ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

First Order Logic in Semantic Tableau and VAMPIRE

Rania Mahmoud*, Ismail Amr Ismail,Ahmed Fahim
*Faculty of Science,Suez University,Suez-Egypt
Faculty of Information Technology,October 6 University,Cairo-Egypt
Faculty of Science,Suez University,Suez-Egypt
rania_cs11@yahoo.com

Abstracts

Semantic tableau is a proof system used to prove the validity of a formula using Prolog, it can also be used to prove if a formula is a logic consequence of a set of formulas. Semantic tableau is used in both propositional and predicate logic. Tableau is a disjunction normal form. VAMPIRE is a high-performance theorem prover for first-order logic with or without equality. In this paper, a modified version of semantic tableau is presented, experimental results proved the validity of modified tableau. Also, it is shown how Tableau proof system and VAMPIRE can be used in predicate logic.

Keywords: Predicate Logic, Semantic Tableau, VAMPIRE..

Introduction

Since the days of Turing theorem has been so important especially in the case of long proofs proving which if done manually could be very much liable to human errors, Therefore automatic proving has been a very important replica. An automatic theorem prover for first order logic(FOL) is a procedure or program that can be used to show that some goal formula is implied by some first order theory, usually represented by a finite set of formulas. A semantic tableau is a tree representing all the ways the conjunction of the formulas at the root can be true. We expand the formulas based on the structure of the compound formulas. This expansion forms a tree. If all branches in the tableau lead to a contradiction, then there is no way the conjunction of the formulas at the root can be true. A path of the tree represents the conjunction of the formulas along the path[2].

VAMPIRE is a system for proving theorems in first order logic with equality. VAMPIRE developed in a resolution-based theorem prover. VAMPIRE is the first theorem prover that can be used for proving and generating program properties automatically.

Semantic tableau was invented by E.W. Beth and J. Hintikka (1965).
The first version of VAMPIRE was implemented in 1993, it was then rewritten several times. The implementation of the current version started in 2009. It is written in C++ and comprises about 152,000 SLOC. It was mainly implemented by Andrei Voronkov and Krystof Hoder. Many of the more recent developments and ideas were contributed byLaura Kovacs. Finally, recent work on SAT solving and bound propagation is due to Ioan Dragan[3].

Problem formulation

First Order Predicate in Tableau

A semantic tableau is a proof system used to:

1. Test a formula A for validity.
2. Test whether B is a logical consequence of A_1,..., A_k.
3. Test A_1,..., A_k for satisfiability.

Definition 1:A path of a tableau is said to be closed if it contains a conjugate pair of formulas, i.e. if some formula A and ~ A appear in the same path. A path of a tableau is said to be open if it is not closed. A tableau is said to be closed if each of its paths is closed[8].
We will see how tableau can be used to prove the validity of formula:

1. To test a formula A for validity, from a tableau starting with ~A. If the tableau closes off then A is logically valid.
2. To test whether B is a logical consequence of A_1,..., A_k, form a tableau starting with A_1,..., A_k, ~B, If the tableau closes off then B is a logical consequence of A_1,..., A_k.
3. To test A_1,..., A_k for satisfiability, form a tableau starting with A_1,..., A_k, If the tableau closes off then A_1,..., A_k is not satisfiable. If the tableau does not close off then A_1,..., A_k is satisfiable,

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

and from any open path we can read off an assignment satisfying $A_1,..., A_k$.

There are 7 rules used to construct the tableau in the propositional logic as shown in the following figure

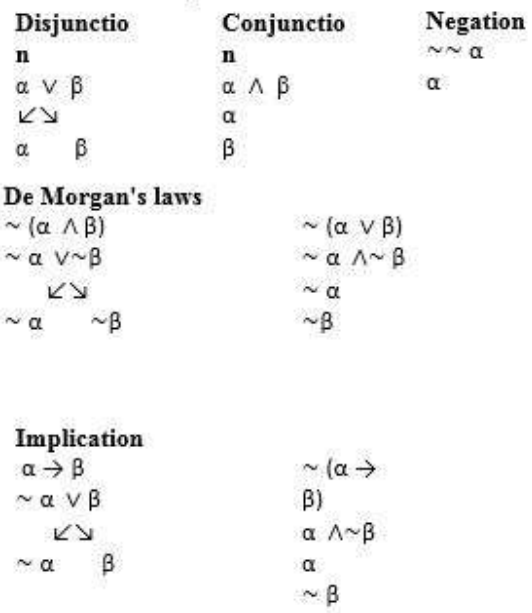

Fig 1: Tableau Rules used in Propositional Logic

Tableau proof is used also in predicate logic by adding rules to cope with the universal and existential quantifiers i.e. (∀, ∃).

There are additional 6 rules used in predicate logic are shown as:

Rule 1: (∀x) α(x)
α(t),
for any ground term or constant term t, where t is a term free from variables.

Rule 2: ~{(∀x) α(x) }
~α (c),
for any new constant c not occurring in α.

Rule 3: Consider A(x, t) to be any an atomic formula with x as a variable and t an any ground term.
~ (∀x) A(x, t)
~A(f(t),t),
where f is some function of t.

Rule 4: Consider A(x, t) to be any an atomic formula with x as a variable and t an any ground term.
(∃ x) A(x, t)
A(g(t),t),
where g is some function of t.

Rule 5: (∃x)α(x)
α(c), for any new constant c.

Rule 6: ~{(∃ x)α(x)}
~α(t), for any ground term t.

The following example shows how tableau proof system is used in proving

Tableau root
~ [(∃x) (P(x) ∨ Q(x)) →((∃x) P(x) ∨ (∃x) Q(x))] (1)
(∃x) (P(x) ∨ Q(x)) (2)
~ [(∃x) P(x) ∨ (∃x) Q(x)] (3)
~ (∃x)P(x) (4)
~ (∃x)Q(x) (5)
P(a) ∨ Q(a) , for some constant a (6)
~P(t1) , for any ground term t1 (7)
~Q(t2) , for any ground term t2

P(a) Q(a)
closed{P(a), ~P(t1)} closed{Q(a), ~Q(t2)}

In this example all the branches are closed. Therefore, the original formula is valid, where {a, t1} and {a, t2} are identical by applying the unification algorithm

Substitution and Unification

Suppose we have two terms t and u, each containing variables. How do we decide whether there are any substitutions that make t and u identical?. A substitution θ is called a unifier of a finite set S of literals if Sθ is a singleton set[9].

A unifier θ for S is called a most general unifier (mgu) for S if for every unifier α of S there exists a substitution σ such that α = θσ. We want to find a way to construct an mgu for any set of literals.

If S is a set of literals, then the disagreement set of S is constructed in the following way.

1. Find the longest common substring that starts at the left end of each literal of S.
2. The disagreement set of S is the set of all the terms that occur in the literals of S that are immediately to the right of the longest common substring.

Unification Algorithm (Robinson)

Input: A finite set of atoms.

Output: Either a most general unifier for S or a statement that S is not unifiable

1. Set k = 0 and $\theta_0 = \varepsilon$, and go to Step 2.
2. Calculate $S\theta_k$. If it's a singleton set, then stop (θ_k is the mgu for S).

Otherwise, let D_k be the disagreement set of S, and go to Step 3.

3. If D_k contains a variable υ and a

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

term t, such that υ doesn't occur in t, then calculate the composition θ_k+1 = θ_k {υ/t}, set k:= k+1, and go to Step 2. Otherwise, stop (S isn't unifiable).

A Tableau Implementation

In this section, we discuss an implementation of the free variable first order tableau. We apply all Tableau Expansion Rules before proving branch closure. Finally, all Tableau Substitution Rule applications will apply most general unifiers to close branches. Correspondence between formulas and their types is summarized in Table1. We use α for formulas of conjunctive type, β for formulas of disjunctive type, γ for quantified formulas of universal, and δ for quantified formulas of existential type. In the case of γ - and δ -formulas the variable x bound by the (top-most) quantifier is made explicit by writing γ(x) and γ_1(x) (resp δ(x) and δ_1(x)); accordingly γ_1(t) denotes the result of replacing all occurrences of x in γ_1 by t. Associativity of∧ and∨ justifies conjunctive and disjunctive formulas with an indefinite number of arguments [4].

Table 1 Correspondence between formulas and their types.

α	$\alpha_1, \dots, \alpha_n$
$\varphi_1 \wedge \dots \wedge \varphi_n$	$\varphi_1, \dots, \varphi_n$
$\sim(\varphi_1 \vee \dots \vee \varphi)$	$\sim\varphi_1, \dots, \sim\varphi_n$
$\sim\sim\varphi$	Φ

β	$\beta_1, \dots, \beta_n$
$\varphi_1 \vee \dots \vee \varphi_n$	$\varphi_1, \dots, \varphi_n$
$\sim(\varphi_1 \wedge \dots \wedge \varphi_n)$	$\sim\varphi_1, \dots, \sim\varphi_n$

δ	δ_1
$\sim(\forall x)(\varphi(x))$	$\sim\varphi(x)$
$(\exists x)(\varphi(x)$	$\varphi(x)$

γ	γ_1
$(\forall x)(\varphi(x))$	$\varphi(x)$
$\sim(\exists x)(\varphi(x))$	$\sim\varphi(x)$

$\frac{\gamma}{\gamma(x)}$	$\frac{\delta}{\delta(f(x_1, \dots, x_n)}$
(for an unbounded variable x)	(for f new Skolem and $x_1, \dots, x_n$ all the used free variables)

If δ occurs on a branch, we add $\delta(f(x_1, \dots, x_n)$ to the branch end, where x must be a free variable that doesn't also occur bound in the tableau; f must be a new Skolem function symbol, and $x_1, \dots, x_n$ all free variables occurring on the branch. We take a Skolem function symbol to be fun(n), where n is a number that is increased by 1 at each δ rule application. We introduce a predicate, funcount, whose purpose is to save the current Skolem function number. Then each time the δ rule is applied, the function newfuncount is called upon to increase the funcount by 1.

The Q-depth is a pre-set bound, representing quantifier depth. The γ rule is applied the maximum allowed number of times, as specified by the value of Q-depth. In this way a complete tableau expansion for a given Q-depth can be constructed in a finite number of steps, and we can then go on to the closure testing stage. Proofs are finite objects so if a sentence X is provable; it has a proof in which some finite number of γ rule applications has been made. Consequently, if X is valid, it will be provable at some Q-depth. Being invalid is equivalent to being unprovable at every Q-depth, so that we will change the Q-depth; by increasing or decreasing it until get closure.

We want to ensure that if there are several γ rules on a branch. We do not apply the Q-depth of γ rule applications on a single formula so that we treat each branch as a priority queue. When working with a branch, we work from the top-down. We apply the rule to the uppermost negation or α formula. If we have a formula that is not a γ formula, we remove it from the branch and add its components or an instance. On other hand, if we have aγ formula in our problem, we remove it, add an appropriate instance to the branch top, and add γ to the branch end. We work as long as apply γ rule on all branches of problem.

First Order Predicate in VAMPIRE

VAMPIRE is an automatic theorem prover for first-order logic. VAMPIRE implements symbol elimination, which allows to automatically discover first-order program properties, including quantified ones. VAMPIRE is the first theorem prover that can be used for proving and generating program properties automatically. VAMPIRE is fully compliant with the first-order part of the TPTP syntax[1]:a Prolog-like

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

syntax allowing one to specify input axioms and conjecture for theorem provers, used by nearly all first-order theorem provers. It was the first ever first-order theorem prover to implement the TPTP if-then-else and let-in formula and term constructors useful for program analysis[3].

In this section, we describe a simple way of using VAMPIRE for proving a formula. Writing the formula as a problem in the TPTP syntax and applying VAMPIRE on this problem. VAMPIRE is completely automatic.

Table 2 Correspondence between the first-order logic and TPTP notations

First-Order Logic	TPTP
$\bot, \top$	$false, $true
$\neg F$	~ F
$F_1 \wedge ... \wedge F_n$	F_1&... &F_n
$F_1 \vee ... \vee F_n$	F_1 |... | F_n
$F_1 \rightarrow Fn$	F_1=>F_n
$F_1 \leftrightarrow F_n$	F_1<=> F_n
$(\forall x_1)...(\forall x_n)F$	![x_1,...,x_n]:F
$(\exists x_1)...(\exists x_n)F$	?[x_1,...,x_n]:F

Refutation in VAMPIRE

VAMPIRE is applied to some axioms and conjecture of first order predicate to obtain a refutation. We can prove the refutation of a formula by adding the negation of the formula and checking if the resulting set of formulas is unsatisfiable. If it is, then the resulting formula is a logical consequence of the axioms.

Definition 2: An *inference rule* is an n-arty relation on formulas, where $n \geq 0$. The elements of such a relation are called *inferences* and usually written as

$$\frac{A_1, \ldots, A_n}{A}$$

The formulae $A_1, \ldots, A_n$ are called the *premises*, and the formula A the *conclusion*, of this inference. An *inference system* is a set of inference rules. An *axiom* of an inference system is any conclusion of an inference with 0 premises.

Each formula (or inference) in the proof is obtained using one or more inference rules. They are shown in brackets, together with parent numbers. Some examples of inference rules in this proof are *superposition*, *inequality splitting* and *skolemisation*. There are several kinds of inference rules. Some inferences, marked as input, introduce input formulae. There are many inference rules related to preprocessing input formulae, for example ennf [*equivalence negation normal form*] transformation and cnf [*conjunction normal form*] transformation. The input formulas are finally converted to clauses, after which VAMPIRE tries to check the unsatisfiability of the resulting set of clauses using the resolution and *superposition inference system*. The *superposition calculus* rules are divided in to two parts *generating* and *simplifying* ones.

Formulas and clauses having free variables are considered implicitly universally quantified. Normally, the conclusion of an inference is a logical consequence of its premises. Inference rules used by VAMPIRE guarantee soundness, which means that an inference cannot change a satisfiable set of formulas into an unsatisfiable one.

At the end of any run to VAMPIRE, there are statistics about the proof, including the overall running time, used memory, and the termination reason (for example, refutation found). In addition, it contains information about the number of various kinds of clause and inferences. VAMPIRE is based on a complete inference system, if the problem is unsatisfiable, enough time and space are given to find a refutation. When VAMPIRE cannot find a refutation, there are many ways try to find solution, increasing time limit and so on.

Preprocessing

Given a problem, VAMPIRE works as follows:

- Input the formula.
- Determine proof-search options to be used for this formula.
- Preprocess the formula.
- Convert the formula into *equivalence negation normal form* (ennf).
- Remove if-then-else and let-in connectives.
- Apply pure predicate elimination.
- Use a naming technique to replace some sub formulas by their names.
- Skolemise the formula.
- Transform the formula into its *conjunctive normal form* (cnf).
- Function definition elimination (optional).
- Apply inequality splitting (optional).
- Remove tautologies.
- Apply pure literal elimination (optional).
- Remove clausal definitions (optional).

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

- Report the result, maybe including a refutation.

Superposition Inference System

The *superposition inference system* depends on a simplification ordering and selection function. It consists of the following rules [7]:

Resolution. $\frac{A \vee C_1 \quad \neg A' \vee C_2}{(C_1 \vee C_2)\theta}$,

where θ is a most general unifier of the atomic formula A and A'.

Factoring. $\frac{A \vee A' \vee C}{(A \vee C)\theta}$,

where θ is a most general unifier of the atomic formula A and A'.

Equality Resolution. $\frac{s \neq t \vee C}{C\theta}$,

where θ is a most general unifier ofs and t.

Functionality of VAMPIRE

Features of VAMPIRE related to its functionality, including the splitting rule, simplification orders, simplification rules, clause and literal selection, and reasoning with limited time.

Saturation Algorithm in VAMPIRE

To saturate a set of clauses S with respect to an inference system, we need a saturation algorithm. At every step of the saturation algorithm, we select an inference, apply this inference to S, and add the conclusion of the inferences to the set S. If the initial set is unsatisfiable, then we will prove the refutation of clauses[5].

Clause selection in VAMPIRE is dependent on two factors: the age and the weight priorities of a clause, where the age is used as numbering of clauses. Each clause has a unique number in increasing order, older clauses have smaller numbers. Each clause has a weight equal to its size, which is the total number of symbols in it. The clauses are selected from the age and weight priority queues using an *age-weight ratio*, a pair of non negative integers (a, w). If the *age-weight ratio* is (a, w), then of each a + w clauses, a oldest and w lightest clauses are selected. The age-weight ratio is specified by the user using the command --age_weight_ratio.

Splitting

VAMPIRE implements the splitting without backtracking; the splitting rule is specified as follows:

$$\frac{S \cup \{C \vee D\}}{S \cup \{C \vee p, D \vee \neg p\}}$$

where S is a set of clauses, the clauses C and D have no common variables, and p is a new propositional symbol. This rule is used in splitting of the clause store into two new stores, as shown in the β-rule in semantic tableau[5]

$$S \cup \{C \vee D\}$$
$$\swarrow \quad \searrow$$
$$S \cup \{C\} \qquad S \cup \{D\}$$

Colored Proofs and Interpolation

Colored proofs are used in a program when some (predicate and/or function)symbols are declared to have colors. In colored proofs every inference and every term or atomic formula can use symbols of at most one color. We call a symbol, term and clause *colored* if it uses a color, otherwise it is called *transparent*[6].

Let L and R be two closed formulae, have no free variables. A formula I is called an *interpolant* of formulae L and R if the following conditions are satisfied:
(1) ⊢ L → I;
(2) ⊢ I → R;
(3) I uses symbols occurring in both L and R.

where the existence of an *interpolant* implies that ⊢ L → R; R is a logical consequence of L.

Simulated output result

The method of tableau works by starting with the initial set of formulae and then adding to the tableausimpler formulae until contradiction is shown in the simple form of opposite literals. Since the formula represented by a tableau is the disjunction of the formulae represented by its branches, contradiction is obtained when every branch contains a pair of opposite literals.

Once a branch has contained a literal and its negation, its corresponding formula is unsatisfiable. As a result, this branch can be now "closed", as there is no need to further expand it. If all branches of a tableau are closed, the formula represented by the tableau is unsatisfiable; therefore, the original set is unsatisfiable as well. Obtaining a tableau where all branches are closed is a way for proving the unsatisfiability of the original set.

In Tableau system, we prove the validity of a formula for a specific Q-depth. If a formula can't be first order at a specific Q-depth, we increase it to get refutation. The following example is compiled in SWI-Prolog(w64pl-2013-11-06)

Example: Check the validity of the formula(∃ x)(p(x) ∧ q(x)) → (∃ x)p(x) ∧ (∃ x)q(x)
If we check at Q-depth 1, we find that the formula not first order tableau at 1.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

```
test(X, Qdepth) :-reset, branch(Notated,[neg
X]), notation(Notated, []),
expand([Notated],Qdepth,Tree),
if_then_else(closed(Tree),yes(Qdepth),
no(Qdepth)).
yes(Qdepth):-write('First-order tableau theorem
at Q-depth'), write(Qdepth), write(.) ,nl.
no(Qdepth):-write('Not First-order tableau
theorem at Q-depth'), write(Qdepth), write(.), nl.
?-test(some(x, p(x) and q(x)) imp (some(x , p(x))
and some(x, q(x))),1).
Not First-order tableau theorem at Q-depth 1.
true
```

We modify the program as shown as

```
test(_ ,5).
test(X, Qdepth) :-reset, branch(Notated,  [neg
X]), notation(Notated, []),
expand([Notated], Qdepth, Tree),
if_then_else(closed(Tree), yes(Qdepth),
no(Qdepth)),
NewQdepth is Qdepth+1, test(X, NewQdepth).
yes(Qdepth):-write('First-order tableau theorem
at Q-depth '), write(Qdepth), write(.) ,nl.
no(Qdepth):-write('Not First-order tableau
theorem at Q-depth'), write(Qdepth), write(.), nl.
?-test(some(x, p(x) and q(x)) imp (some(x, p(x))
and some(x, q(x))),1).
Not First-order tableau theorem at Q-depth 1.
First-order tableau theorem at Q-depth 2.
First-order tableau theorem at Q-depth 3.
First-order tableau theorem at Q-depth 4.
true.
```

By increasing the Q-depth (optional), we can prove that the formula is first order tableau.
VAMPIRE is high-performance theorem prover for first-order logic , based on resolution and superposition. We are using VAMPIRE to obtain a refutation of first order formula. We write the axioms and conjecture in the TPTP-syntax, if we find that the formula is unsatisfiable, so we are getting a refutation. When running the same examples which applied in tableau, we have the same result (where the input formula is satisfiable or unsatisfiable).

```
vampire(option,show_interpolant,on).
vampire(symbol, predicate, q, 1, left).
vampire(symbol,predicate,p,1, left).
vampire(left_formula)
fof(l,
axiom,~(?[X]:(p(X)&q(X))=>(?[X]:p(X)&?[X]
:q(X)))).
vampire(end_formula).
 vampire(right_formula).
fof(r, conjecture, $false).
 vampire(end_formula).
```

This example contains first order problem (one axiom and conjecture) written in TPTP syntax. We need to prove a refutation. The first three declarations after command shown the example denote that q and p are a unary predicate symbol colored in the left color. The declarations fof(...) are TPTP declarations for introducing formulae. The VAMPIRE declarations *left_formula*, *right_formula* and *end_formula* are used to define L and R. The letters l and r are chosen to denote the name of this axioms. The user can choose any name. Names of input are ignored by VAMPIRE. The conjecture is keyword *$false* to denote that the formula ~(?[X]:(p(X)&q(X))=>(?[X]:p(X)&?[X]:q(X)))) has a refutation. The expression ~(?[X]:(p(X)&q(X))=>(?[X]:p(X)&?[X]:q(X)))) is to prove the refutation. We save a problem in a file and run VAMPIRE using the command (Vampire Filename). The output is in steps. Every formula is assigned a unique number. The proof consists of inferences. Each inference infers a formula, called the conclusion of this inference, from a set of formulas, called the premises of the inference. The output of the last example is shown as:

Example

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

```
Refutation found. Thanks to Tanya!
16. $false (0:0) [sat splitting refutation 15,11,13,12,14]
 14. $false {0} (1:0) [resolution 12,9]
 9. p(sK0) (0:2) [cnf transformation 8]
 8. (p(sK0) & q(sK0)) & (! [X1] : ~p(X1) | ! [X0] : ~q(X0)) [skolemisation 7]
7. ? [X2] : (p(X2) & q(X2)) & (! [X1] : ~p(X1) | ! [X0] : ~q(X0)) [rectify 6]
 6. ? [X0] : (p(X0) & q(X0)) & (! [X2] : ~p(X2) | ! [X1] : ~q(X1)) [ennf transformation 4]
 4. ~(? [X0] : (p(X0) & q(X0)) => (? [X2] : p(X2) & ? [X1] : q(X1))) [rectify 1]
1. ~(? [X0] : (p(X0) & q(X0)) => (? [X0] : p(X0) & ? [X0] : q(X0))) [input]
 12. ~p(X1) {0} (0:2) [sat splitting component]
 13. ~q(X0) {2} (0:2) [sat splitting component]
 11. ~q(X0) | ~p(X1) (0:4) [cnf transformation 8]
15. $false {2} (1:0) [resolution 13,10]
10. q(sK0) (0:2) [cnf transformation 8]
Interpolant: $false
```

In VAMPIRE, we get a statistical report about the proof is shown as:

```
Version: Vampire 3.0 (revision 2069)
 Termination reason:  Refutation
 Active clauses: 4
 Passive clauses: 4
 Generated clauses: 7
Final active clauses: 3
Input formulas: 2 Initial clauses: 3
Binary resolution: 2
Split clauses: 1
Split components: 2
SAT solver clauses: 4
 SAT solver unit clauses: 2
 SAT solver binary clauses: 1
SAT solver learnt clauses: 1
Sat splits: 1
Sat splitting refutations: 2
Memory used [KB]: 255
Time elapsed: 0.182 s
```

Conclusion

For a long time, by applying the examples of tableau in VAMPIRE, we can save time and memory, and also have a statistical report about the proof. We can use the result of
VAMPIRE to increase the Q-depth in tableau, where if we get that the formula is not first order formula (satisfiable) at specified Qdepth, we can increase the depth until we get a refutation (unsatisfiable).

References

1. *G. Sutcliffe. The TPTP Problem Library - http://www.cs.miami.edu/~tptp/, 2013.*
2. *M. C. Fitting. First-Order Modal Tableau, Journal of Automated Reasoning, vol 4, pp 191-213.*
3. *L. Kovacs, A. Voronkov. Vampire Web Page -http:// vprover.org2013*
4. *M. C.Fitting.First-Order Logic and Automated Theorem Proving,Springer-Verlag (1990).*
5. *A. Riazanov and A.Voronkov. The Design and Implementation of Vampire. AI Communications, 15(2-3):91–110, 2002.*
6. *L. Kovacs, A. Voronkov. Interpolation and Symbol Elimination. In Proc. of CADE 2009.pp 199–213. Springer, Heidelberg (2009)*
7. *L. FRIBOURG, A SUPERPOSITION ORIENTED THEOREM PROVER(1984).*
8. *S. Kaushik.Logic and Prolog Programming, New Delhi(2002)*
9. *James L.Hein. Theory of Computation, 1996.*