2009-01-01

Machine Learning for Automated Theorem Proving

Aman Kakkad
University of Miami, addys06@gmail.com

UNIVERSITY OF MIAMI

MACHINE LEARNING FOR AUTOMATED THEOREM PROVING

By

Aman Kakkad

A THESIS

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Master of Science

Coral Gables, Florida

August 2009

UNIVERSITY OF MIAMI

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science

MACHINE LEARNING FOR AUTOMATED THEOREM PROVING

Aman Kakkad

Approved:

Geoff Sutcliffe, Ph.D. Associate Professor of Computer Science	Terri A. Scandura, Ph.D. Dean of the Graduate School
Dilip Sarkar, Ph.D. Associate Professor of Computer Science	Mirsolav Kubat, Ph.D Associate Professor of Electrical and Computer Engineering

KAKKAD, AMAN (M.S., Computer Science)
Machine Learning for Automated Theorem Proving (August 2009)

Abstract of a thesis at the University of Miami.

Thesis supervised by Professor Geoff Sutcliffe.
No. of pages in text. (87)

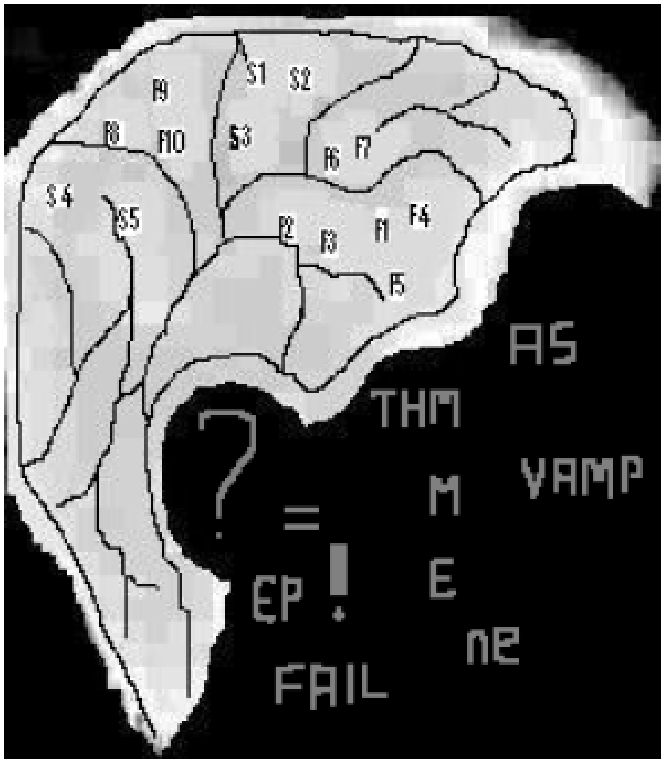
Developing logic in machines has always been an area of concern for scientists. Automated Theorem Proving is a field that has implemented the concept of *logical consequence* to a certain level. However, if the number of available axioms is very large then the probability of getting a proof for a conjecture in a reasonable time limit can be very small. This is where the ability to learn from previously proved theorems comes into play. If we see in our own lives, whenever a new situation S(NEW) is encountered we try to recollect all old scenarios S(OLD) in our neural system similar to the new one. Based on them we then try to find a solution for S(NEW) with the help of all related facts F(OLD) to S(OLD). Similar is the concept in this research.

The thesis deals with developing a solution and finally implementing it in a tool that tries to prove a *failed conjecture* (a problem that the ATP system failed to prove) by extracting a sufficient set of axioms (we call it *Refined Axiom Set (RAS)*) from a large pool of available axioms. The process is carried out by measuring the similarity of a failed conjecture with *solved theorems* (already proved) of the same domain. We call it "process1", which is based on syntactic

selection of axioms. After process1, RAS may still have *irrelevant axioms*, which motivated us to apply semantic selection approach on RAS so as to refine it to a much finer level. We call this approach as “process2”. We then try to prove failed conjecture either from the output of process1 or process2, depending upon whichever approach is selected by the user.

As for our testing result domain, we picked all FOF problems from the TPTP problem domain called SWC, which consisted of 24 broken conjectures (problems for which the ATP system is able to show that proof exists but not able to find it because of limited resources), 124 failed conjectures and 274 solved theorems. The results are produced by keeping in account both the broken and failed problems. The percentage of broken conjectures being solved with respect to the failed conjectures is obviously higher and the tool has shown a success of 100 % on the broken set and 19.5 % on the failed ones.

MACHINE LEARNING FOR AUTOMATED THEOREM PROVING



Acknowledgements

I would like to thank Dr. Geoff Sutcliffe, for giving me an opportunity to work under him and be a part of his research group (ARTists). I learned a lot by communicating with every team member of the research group and would also like to thank each one of them for effectively providing me a feedback related to my research in every group meeting.

I want to thank Dr. Dilip Sarkar and Dr. Miroslav Kubat for serving on my M.S. thesis committee.

I am very grateful to Dr. G.S. Kakkad and all my family members who always motivated me to achieve what I desire.

I am thankful to Miss Ashima Anand, for helping me proof read my thesis.

I would also like to thank all my friends and colleagues.

Table of Contents

List of Tables

List of Figures

Terminology

Assurance: Output produced by any ATP system for a given conjecture, which signifies that there exist a proof but cannot be found because of limited resources.

ATP systems: Computer programs capable enough of finding a solution for a given conjecture with the help of an axiom set (for example, EP [25], E, Metis [26] etc.)

Available axioms: Axioms listed in the corresponding problem file of a conjecture (can be include file axioms).

Axioms: Set of statements which are self evident and necessary for the ATP system to prove a conjecture.

Axiom Domain: Set of all axioms listed for a particular problem domain.

Axiom Refining Strategies [AReS]: Syntactic approaches for refining axiom list are known as axiom refining strategies. There are 4 such strategies that are discussed in Chapter 3 of this thesis.

Broken file: File containing assurance as output produced by any ATP system for a given conjecture.

Broken Problems: Problems which have already been tried proving by any ATP system, and have received assurance as an output.

Closest axiom set: Refined axiom sets are sorted with the help of a tool called SortByUsefulInfoField [23]. After sorting, the top most value depicts closest axiom and bottom value depicts the **farthest axiom** with respect to a given conjecture.

Completeness: A system is called complete when it is able to find a proof for all logical consequence of a set of axioms.

Conjecture: A statement which not proved yet.

Desperate axiom set: Axiom set produced by considering axioms from an include file of a failed conjecture. Formation of such an axiom set is discussed in Chapter 3.

Failed conjecture: Conjecture for which some ATP systems are not able to find a proof from the available list of axioms is called failed conjecture.

Failed file: File containing output produced by any ATP system for all failed problems.

Failed problems: Problems which were already tried by some ATP system for the purpose of proving and ATP system was not able to find proof for them with the provided list of axioms.

First Order Logic: One kind of a language used to write problems for ATP systems. For detailed description see Section 1.2.

First Order Formulae (FOF) problems: A problem is a list of formula and FOF problems consists of number of axioms and a conjecture written in first order logic.

Irrelevant axioms: Axioms which do not create any difference in the proof attempt if they are not present in the refined axiom set.

Irrelevant files: Any file existing in the problem domain but is not useful for the purpose of tool run. They may be present because of the carelessness of user.

Logical consequence: A conjecture is a logical consequence of a set of axioms if every *model* of the axioms is a model of the conjecture.

MLAR: Implementation name of this research.

Model: An interpretation is a model of the formula if the formula is true in that interpretation

Model finder: ATP system which is selected for the purpose of finding models between a set of axioms and a failed conjecture.

Proof attempt: An attempt to prove a failed conjecture with refined axiom set by using any ATP system.

Problem domain: It comprises of all files corresponding to broken problems, failed problems, and solved problems.

Proof output: Proof produced by an ATP system for a given theorem.

Proof tree: Graphical representation of the proof output produced by any ATP system for a given conjecture is called proof tree.

Problem files: Files (containing conjecture) which are given to any ATP system for the purpose of finding a solution for them.

Rating of problem: It shows the hardness of a problem. It’s depicted from value zero to one. Bigger the value, harder is the problem.

Refined axiom set: Axiom list generated by any AReS is known as refined axiom set.

Relevance: Measure by which a conjecture is interrelated to an axiom or a solved theorem (converted to axiom).

Relevance set: Batch of all solved theorems with same relevance.

Relevant axioms: Axioms required for making a successful proof attempt on a failed conjecture.

Solved problems: Problems which were already tried by some ATP system for the purpose of proving and ATP system was able to prove them.

Solution file: File containing an output (for a problem file) produced by any ATP system.

Selected problem set: Set of problems selected from problem domain.

Solved theorems: Conjecture listed in any solved file is called as solved theorem

Theorem: A solved problem is also known as theorem.

Theorem trying time: Time allotted to any ATP system for the purpose of finding a solution for a failed conjecture.

Turned axioms: Solved theorems which are converted into axioms for the purpose of finding relevance are called turned axioms.

Used axioms: Axioms used in a proof of some solved theorem and are extracted with the help of a tool called ProofSummary [24].

Chapter 1

Introduction

In the year 1956, Artificial Intelligence (AI) came up with the big thought of implementing human level intelligence in machines. To conceptualize this, research in AI began to capitalize with many gradual developments like the Turing test [5], neural networks [6], expert systems [7], etc. As the research got intensified, various subfields were formed, one of which is Automated Reasoning (AR).

Reasoning, as in context of human beings, refers to the process of finding reasons so as to validate their actions and beliefs. For instance consider **example 1.1** discussed below:

Situation A: Suppose there is a person named Sam whose family tree is given below:

<u>Example 1.1</u> <u>Sam's family tree comprises of following facts:</u>

Fact AF1: **Sam** is a child of person **Ran** and person **Lee**, **Ran** is father and **Lee** is mother

Fact AF2: **Ran** is a child of person **Can** and person **Bee**, **Can** being father and **Bee** is mother

Fact AF3: Person **Lee** is a child of Person **Pan** and Person **Vee**, **Vee** is father and **Pan** being mother

If given all these facts we humans can automatically conclude many things like:

Conclusion AC1: Sam's grandfather is **Can**

Conclusion AC2: Sam's grandmother is **Bee**

The figure 1.1, shows a tree structure of example 1.1.

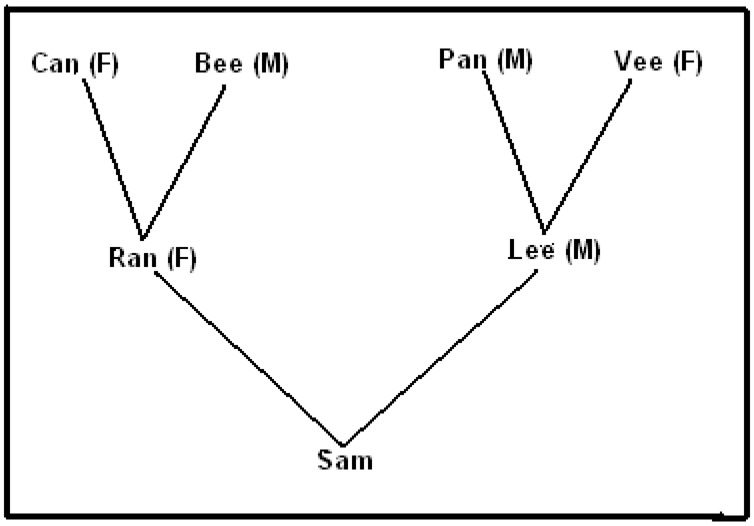


Figure 1.1 - FamilyTreeStructre1

Now by observing above example, we can directly conclude AC1 and AC2 from facts AF1 and AF2 because we are aware of the fact that a person's father is a grandfather of his son, and a person's mother is a grandmother of his son. In the same manner, if we want computer programs to draw some conclusions from given facts then we need to make sure that they have enough information to draw such conclusions. In the Automated Theorem Proving (ATP) world, we call such pieces of information or knowledge *axioms*, and ATP systems are the computer programs specifically designed to prove *theorems* with the help of *axioms*.

For instance, an axiom for the **situation A** will be:

Axiom1 AH1: for all person M, if {(person M is a child of mother N and father O)

and (person P is a child of person M)

and (person M is a father of person P) }

then {(person P's grandmother is person N)

and (person P's grandfather is person O) }

Note: In the axiom above M. N, O, P and H are called variables.

Considering a different example:

Situation B: Suppose there is a person named **Ben** and part of his family tree is given below:

<u>Ben's family tree comprises of following facts:</u>

Fact BF1: Ben is a child of person **Han** and **Kee**, **Han** being father **Kee** being mother.

Fact BF2: Ben's wife is person **Dee**.

Fact BF3: Dee is a child of person **Can** and **Bee**, **Can** being father and **Bee** is mother.

Now for the above mentioned facts BF1, BF2 and BF3 if we provide our ATP system with the axiom BH1 (listed below) then the ATP system would be able to draw the conclusions BC1 and BC2 (listed below).

Axiom BH1: for all person G, if {(person G is a child of mother K, father L)

and (person G has a wife H)}

then {(person H's mother in law is person K)

and (person H's father in law is person L)}

Conclusion BC1: Ben's mother in law is **Bee**

Conclusion BC2: Ben's father in law is **Can**

Thus, AR is a field that brings together the study of all kinds of reasoning which are implemented as computer programs. There are various subareas of AR, out of which the most developed subareas are automated theorem proving and automated proof checking [9]. Work has also been done in reasoning by analogy [28], induction [29] and abduction [30].

There are various ATP systems working efficiently in the field of AR, but they do face some practical challenges. For example, if we consider example 1.1 and **situation A** then to draw conclusions AC1 and AC2 from the given facts we only need axiom AH1. But there can be a case where numerous other axioms are provided to the ATP system for a particular set of problems. For instance say for above example we have:

200 axioms (AH1 ……….. AH200) and

400 conjectures (AC1 ……… AC400)

Now for a particular theorem like AC1 and AC2 we might need only axiom AH1 for it to get proved, but for an ATP system to find axiom AH1 from the list of 200 axioms becomes a tedious job at times. In this research we present a solution for this problem (when the axiom list is too large for the ATP system or it becomes difficult for the ATP system to find relevant axioms so as to prove a failed conjecture). The solution describes four axiom refining strategies that select axioms for a failed conjecture in different manner. All the strategies are discussed in Chapter 3.

1.1 Automated Theorem Proving

ATP started off by the implementation of computer programs for proving mathematical theorems. As the research and funding grew, the field gained its importance in industrial projects as well. Now the technology is not restricted to mathematical theorems but also deals with the implementation of computer programs for proving any given conjecture from a given set of axioms.

Despite the fact that ATP systems are highly successful, we cannot blindly trust the output generated by them. Therefore, we need to have a check on certain things like:

- Whether the axioms are consistent? We need to have a check on the given set of statements for the purpose of determining if they are true in their domain or not? For example, statements like "person Jim Alberto of Moscow, living near Crescent Street is cycling and person Jim Alberto of Moscow, who lived near Crescent Street died 10 years ago" cannot be true if we do not provide more information regarding Jim Alberto. We call this check 'MFCheck1'.
- Should the conjecture be really concluded from the axiom set? For instance, considering example 1.1 the ATP system will not be able to prove statements like Person X has a brother Q. The property is called soundness. We call this check as MFCheck2.

Thus, for the same reason we have a concept called **Model Finding** which helps in checking all these constraints for the ATP systems and allows the research to get reliable results from the technology.

As for any other technology, there are some limitations associated with each one of the ATP systems. The most common ones are:

- Some ATP systems are good in proving a particular set of theorems, like theorems with equality and others are efficient in proving different kind of theorems.
- As described earlier, if large number of axioms are given to any ATP system, then it becomes difficult for that ATP system to select right axioms to prove the given conjecture.

The theorem or conjecture which is not proved by the ATP system because of any of the associated limitation is called a *failed conjecture*. The work that we present here is all about finding solutions for these failed conjectures, with the help of axiom refinement strategies.

Now keeping in mind the first limitation of the ATP systems, a user is given an option of selecting more than one ATP system (discussed in Section 3.2.3). For the second limitation, this research presents various *axiom refinement strategies* using syntactic approaches (discussed in Section 3.3). Also for the purpose of *MFCheck1* and *MFCheck2,* we introduce another approach by using semantic selection of axioms (detailed description in Chapter 4).

1.2 Logic

As we need a language to communicate facts among ourselves, in the same manner there should be a common language for the ATP systems to get universally accepted. The language used to describe axioms and conjectures is called as *logic*. Propositional logic marked the beginning for ATP systems as a language, followed by First Order Logic (FOL) and Higher Order Logic (HOL).

The need of FOL was very much essential when the propositional logic started to struggle with the challenges like use of universal quantification. Considering example 1.1, we cannot define axioms AH1 and BH1 in propositional logic since they have been defined universally. Another challenge which propositional logic faces is the lack of syntax for representing objects, as it only allows the use of statements.

There have been many successes in the development of ATP systems. At the first order level some well known and successful ATP systems are Otter [14] and E [15]. For the purpose of dealing with theorems under higher order logic, some of the successful systems are Nqthm [16], LEO [22], and Isabelle [12].

First Order Logic

Problems in FOL are written with the help of three types of symbols:

1. *Variables* (they are represented with the first letter as capital, don't have a defined value).
2. *Functors* (they are denoted with the first letter as lower case. Every functor has their own arity which means number of arguments, and functors with arity 0 are called constants).
3. *Predicates* (every predicate has their own arity, and those with arity 0 are called propositions).

In FOL, *formulae* are written using *atoms* (a predicate symbol with an arity 0 or with an appropriate number of *terms* (a functor of arity zero is a term, a variable is a term, and a functor with the appropriate number of terms as arguments, is a term) as arguments are called atoms) and various connectives like conjunction, disjunction, negation etc.

The various connectives used in FOL are:

- Disjunction: represented by |
- Conjunction: represented by &
- Negation: represented by ~
- Implication : represented by =>
- Equivalence : represented by <=>
- Exclusive: represented by <~>
- Universal quantification: represented by ∀
- Existential quantification: represented by ∃

In FOL formulae, precedence value of the above connectives is:

∀, ∃, ~, |, &, =>, <=>. Precedence value of <~> is same as |

Some examples of formulae represented in FOL are:

Formula 1	:	∀ X: (father(X) => male(X)).
Formula 2	:	∀ X: (mother(X) => female(X)).
Formula 3	:	∀ X, Y: (zero(X) => (divide(X, Y) = X).

First order logic has a sub-language called Clause Normal Form (CNF). It simplifies FOL formulae to CNF and for ATP systems it becomes easier to work with CNF. In CNF *literals* are atoms and negation of atoms. *Clause* is an expression of form Li_1 | Li_2 ………Li_n where Li $_{(1\text{-}n)}$ are literals. Clauses that are not used in finding a proof for a given conjecture are known as *irrelevant clauses*. Some of the previous work which was designed to deal with huge axiom sets, for example 'light weight relevance filtering' [20] works by filtering irrelevant clauses. FOL can be easily converted to CNF, for instance consider example 1.2.1(a), where FOL 'Formula 1' is converted to CNF form:

Example 1.2.1(a):-

1	FOL formula	:	∀ X (father(X) => male(X))
2	Simplification	:	∀ X (~father(X) | male(X))
3	Move negations in	:	∀ X (~father(X) | male(X))
4	Move quantifiers out	:	∀ X (~father(X) | male(X))
5	Sokelemization	:	~father(X) | male(X)
6	Distribute disjunction	:	~father(X) | male(X)
7	CNF form	:	{~father (X) | male(X)}

1.3 Large Theories (LT)

1.3.1 What are they?

ATP systems have already made their grounds in applications like software design, software verification etc., but for the survival of any new technology it has to deal with the large databases. As far as ATP systems are concerned, a database is a set of axioms.

Now when this set of axioms becomes so large that the ATP systems are not able to select necessary axioms (axioms with which some ATP system is able to find a proof for a given conjecture), it becomes a threat to this technology. As a matter of fact, work produced by W. Reif and G. Schellhorn in "Theorem Proving in Large Theories" [1] clearly shows that the real world application does have a huge list of axioms for the ATP systems, and most of the ATP systems are not capable enough to handle these sets in an efficient manner. One of the software verification tools that deal with huge sets of axioms is KIV [2], [3], [4].

As a matter of fact 'ATP problems may contain unnecessary axioms, either because some of the axiomatization of the theory is irrelevant to the particular theorem, or because of the axiomatization is redundant by design' [11]. The study of how to deal with these huge set of axioms in the field of ATP is called Large Theories. There are many examples of such large theories like: YAGO [18], and CYC [17].

1.3.2 Problems presented by LT

The main challenge that large theories present to ATP systems is the huge search space. Previous work by W. Reif and G. Schellhorn shows the example of KIV tool where the ATP systems got lost in the search space and were even misled in the process of finding correct axioms (axiom set from which it's possible to find a proof for failed conjecture) for the conjecture.

Now if we consider the *proof tree* then for a particular conjecture (say ConP1), when the tree is built some axioms are used more than once (we call these axioms EsAx), some of them are used once (we call these axioms ImAx), and some are not used to find a proof of ConP1 (we call these axioms IrAx), which clearly shows that some axioms are irrelevant to the conjecture (IrAx). Since examples of large theories are comprised of very many axioms, the chances of irrelevant axioms for the given conjecture, increases to a high value. Besides the above mentioned problems, each application of any technology has some practical constraints associated with it (for example time).

Keeping all the above facts in mind, it's clear that with a huge axiom set, ATP systems are not very efficient in finding a proof for a given conjecture (say GC1). This motivates the development of a new strategy, which is capable of selecting a subset of axioms from a large axiom set for the purpose of proving such conjectures (like GC1).

The above mentioned issues were the prime motivators of this research. The work presented here tries to select an adequate subset of axioms from a large axiom set, for every failed conjecture. With this subset of axioms, a proof attempt is made on a failed conjecture. Solved theorems from the same domain are considered for the purpose of making a subset of axioms (detailed description in Chapter 3). This research assumes that, the closer the solved theorem is to a failed conjecture, the higher will be the probability of finding a solution for a failed conjecture.

1.3.3 Relevance

Relevance is defined as a measure that inter-relates a failed conjecture to an axiom or to a solved theorem. It's calculated with the help of Prophet [8]. Prophet uses a syntactic relevance measure to calculate a relevance measure between a conjecture and an axiom.

It assigns a numeric value (say Pvalue) to axioms with respect to a failed conjecture (for detailed description of Prophet please see Chapter 2). The output from Prophet is not in sorted order of relevance value, as a result of which we use another tool called SortByUsefulInfoField [23] to sort the list produced by Prophet.

Prophet gives a relevance value for an axiom with respect to a failed conjecture, which is later used in this research for the purpose of establishing similarity between a failed conjecture and an axiom. As discussed in Chapter 3, the similarity between a failed conjecture and solved theorems is also needed, which can be determined by converting solved theorems into axioms (these are called *turned axioms*). From the output of Prophet, similarity is directly proportional to the relevance value, which means that higher the relevance value more similar a turned axiom is to a failed conjecture.

For instance consider this example 1.3.3a:

List of failed conjecture

Failed conjecture 1	:	FC1
Failed conjecture 2	:	FC2

List of turned axioms

Turned Axiom 1	:	ST1
Turned Axiom 2	:	ST2
Turned Axiom 3	:	ST3
Turned Axiom 4	:	ST4
Turned Axiom 5	:	ST5

Now if we give FC1, ST [1-5] to Prophet it will produce a numeric value for all ST [1-5] in relation to FC1. Suppose Prophet generates following list:

```
==================
 FC1     [relevance: 1.00]
 ==================
ST1      [relevance: 1.00]
ST2      [relevance: .77]
ST3      [relevance: .1.00]
ST4      [relevance: .99]
ST5      [relevance: .77]
==================
```

As we can clearly see that the output received from Prophet is not sorted, therefore we use SortByUsefulInfoField to get the desired sorted list. The above list will now become:

Sorted Conjecture list	**Name**	**Relevance (with FC1)**
Solved conjecture turned axiom 1	ST1	1.00
Solved conjecture turned axiom 3	ST3	1.00
Solved conjecture turned axiom 4	ST4	0.99
Solved conjecture turned axiom 2	ST2	0.77
Solved conjecture turned axiom 5	ST5	0.77

Table 1.3.3: Sorted conjecture turned axiom list

Relevance plays an important role in this research as it forms the basis of finding a similarity measure between solved theorems and a failed conjecture, which in turn initiates the learning process.

1.4 Statement of contribution

Contribution towards technology:

- This work will play a role in the research of “Axiom selection strategies”.
- The work is designed to deal with the problem of large theories in the field of ATP.
- This work is an additional piece of work in the machine learning area of ATP after MaLARea by Josef Urban [10].
- The work also defines new strategies of axiom refinement. The strategies are: Sorted solved conjecture relevance set, Axioms relevance set, Sorted solved conjectures by average axiom relevance, One axiom one time (all these techniques are discussed in Chapter 3).
- The work clearly separates broken problem set and failed problem set from the problem domain and shows that the chances of broken problem set to get solved are higher than failed problem set based on results that we have achieved.

Contribution towards research:

- Implementation of tool.
- Tested the tool against provided test set.
- Analysis of test results.

1.5 Structure of thesis

The thesis has been divided into various Chapters.

Chapter 1 begins by defining Automated Reasoning followed by the concept of ATP. It then explains the need for ATP systems, the basic idea behind their implementation, their applications and limitations. Finally, focusing on the most concerned limitation of ATP systems called large theories.

Chapter 2 reviews the work done on previous approaches of axiom selection. There are three approaches namely: Syntactic selection approaches (SySA), Machine learning approaches (MLA) and Semantic selection approaches (SeSA). We discuss some of the work done in each one of them like MaLARea by Josef Urban [10] for MLA, SRASS by G. Sutcliffe [21] under SeSA, and Prophet [8] under the section of SySA. Their similarities and differences from this research have also been briefed.

Chapter 3 explains the idea of Nearest Neighbor Learning which forms the basis of this thesis. While discussing the main concept we also define our approach of how the axioms are ordered and relevance sets are made with respect to the given conjecture by using syntactic approaches. It also shows that TPTP problem library [13] consist of three kinds of problems namely broken problems, failed problems and solved problems. Finally, it defines the complete algorithm without semantic selection approach.

Chapter 4 states a semantic selection approach to filter the axioms from already refined axiom set. It describes the concept of model finding, which is helpful in refining the axiom set to a much finer level. Finally, a complete algorithm with semantic selection approach is defined.

Chapter 5 describes all the implementation details of the tool followed by the test results. User's perspective, test results for the broken problem set, results for completely failed problems, observation and analysis with both the result sets along with all the hardware details, time constraints and other essential requirements are also mentioned.

Finally, **Chapter 6** draws the conclusion of the work done and presents the future aspects and improvements for this thesis.

Chapter 2

Literature

2.1 Overview of approaches to Axiom selection

When it comes to large theories, some work has already been done in finding different strategies for axiom selection. Three main approaches are:

- Syntactic Selection Approach (SySA): here the focus is on selecting relevant *clauses* with the help of counting function symbols [19], relevance distance [20] etc. Some of the related works to this research are:

  -Prophet [8]

  -Lightweight relevance filtering for machine generated resolution problems [20].

- Machine learning (ML): it is the most recent approach that lay emphasis on learning from previous results in the same *problem domain* (to which the selected conjecture belongs to), related work to this research is:

  -MaLARea [10].

- Semantic Selection Approach (SeSA): here the focus is on the selection of relevant axioms with respect to a given conjecture, which is determined by establishing that a given conjecture is the *logical consequence* of the set of axioms. Related work to this research is:

  -SRASS [21]

2.2 Syntactic Selection Approaches

2.2.1 Prophet

Working:

Prophet works on the principle of finding *relevance* between a given conjecture and available axioms on the basis of common symbols.

Relevance is calculated through prophet with the underlying formula:

Relevance formula = (Common symbols) / (Total number of symbols used)

For instance consider **example 2.2.1(a)**:

Suppose conjecture FC1 has following symbols (i, and p) and

Axiom Ax1 consists of (u, m, n, i, and p).

Variables are not considered here because they can be generalized to any specific symbol. Thus, considering variables will not exactly define how far the conjecture and axioms really are.

Prophet works by finding specific symbols between formulae. Now considering example 2.2.1(a), conjecture FC1 has two symbols and axiom Ax1 has five symbols.

Also FC1 and Ax1 have 2 symbols in common, and the total number of common symbols between FC1 and Ax1 are 5. Therefore, by applying relevance formula we get:

Relevance = 2 / 5 = 0.4.

Therefore the relevance measure between FC1 and Ax1 is .4.

Relevance value produced by prophet lies between 0.00 and 1.00, it also signifies that higher the value closer the conjecture and axiom.

Relation to this work:

As defined in section 1.3.3, this work uses Prophet to find relevance value between *failed conjecture* and *solved theorems*. Based on the relevance value, different axiom sets are formed with the help of various *axiom refinement strategies* (detailed description can be found in Chapter 3). Prophet plays a vital role in this research and provides a platform from which we can establish similarities between solved theorems and failed conjecture.

2.2.2 Light Weight Relevance Filtering (LWRF)

In this work, Jia Meng, Lawrence Paulson [20] laid emphasis on the filtering of *irrelevant clauses*. To filter these clauses they presented a relevance filtering approach by counting function symbols in clauses. In LWRF a clause is considered close enough to existing set of clauses based on the relevance value it receives. Similar to our approach, in LWRF higher the relevance value closer is the clause to existing ones. The process begins by adding clauses from a conjecture to an empty set (which they call ‘pool of relevant clauses’). Process then iterates by adding closest clauses based on their relevance measure with respect to the pool of relevant clauses, and it terminates when no new clauses are accepted.

This approach is a bit different from that used in Prophet. Prophet is generally useful in finding relevant axioms from the domain with the help of relevance filtering. LWRF is more helpful in discarding irrelevant clauses in between the process when ATP system is trying to prove the given conjecture by using resolution techniques.

LWRF is applicable to situations where completeness is less important than achieving a high success rate with limited processor time [20]. This is where it is related to our work as we give less importance to fulfill *completeness* and lay more emphasis on solving the failed conjectures with limited processor time in each *proof attempt*.

2.3 Machine Learning Approaches

2.3.1 MaLARea

Research by Josef Urban closely relates to our research. According to MaLARea [10], the goal of learning could be stated as creating an association of some features (in the machine learning terminology) of conjecture formulas (or even of the whole problems when speaking in generally) with proving methods which turned out to be successful when those particular features were present [10]. Basic functioning of MaLARea is depicted in following underlying steps:

- Select a problem domain.
- Extract all problems.
- Extract all available axioms.
- Order axioms according to their expected relevancy
- Try proving the problem from selected ATP systems with most relevant axioms (from the ordered list) and lowest allowed time limit.
- From all the conjectures which are newly solved, learning (defined above) is immediately applied to the problem domain and it is believed that every new solution adds some new knowledge to the domain which is helpful to prove other conjectures.
- If there is no success, double the axiom limit and try proving.
- If still there is no success, time limit is quadrupled.

Similarities to this research:

- Complete axiom set is not picked rather small sub-sets of axioms are made every time with the help of already solved conjecture.
- Both the research areas are related to large theories in ATP world.
- Both the research filter axioms with the help of semantic approach.

Differences from this research:

- We try to prove every failed conjecture with different axiom sub-sets.
- In MaLARea, as soon as a new problem is solved, the knowledge is immediately added to the problem domain which is true in this research. But in this research new knowledge only becomes applicable when MLAR is done applying *relevance*

restriction strategies followed by *proof attempts* on a user selected set of *failed theorems*. It is because the tool gives user an option of selecting number of conjectures to be attempted before updating the database.

- In MaLARea, time limit is not increased automatically where as in our research user has to give a time limit.
- In MaLARea, axiom set is doubled where as in our research new axioms get added based on the next relevance set.

2.4 Semantic Selection Approaches

2.4.1 SRASS

SRASS, also deals with large theories, Figure 2.4.1 depicts the working of SRASS. It works in an iterative manner by finding models of a selected set of axioms with a failed conjecture and carrying on the process till there are no models left. Then it becomes obvious that the given conjecture is a *logical consequence* of a set of axioms. In case of SRASS, available axioms are first ordered based on their relevance value with respect to the conjecture. As the process continues, most syntactically relevant axioms (with respect to conjecture) are selected until the ordered axiom list is empty.

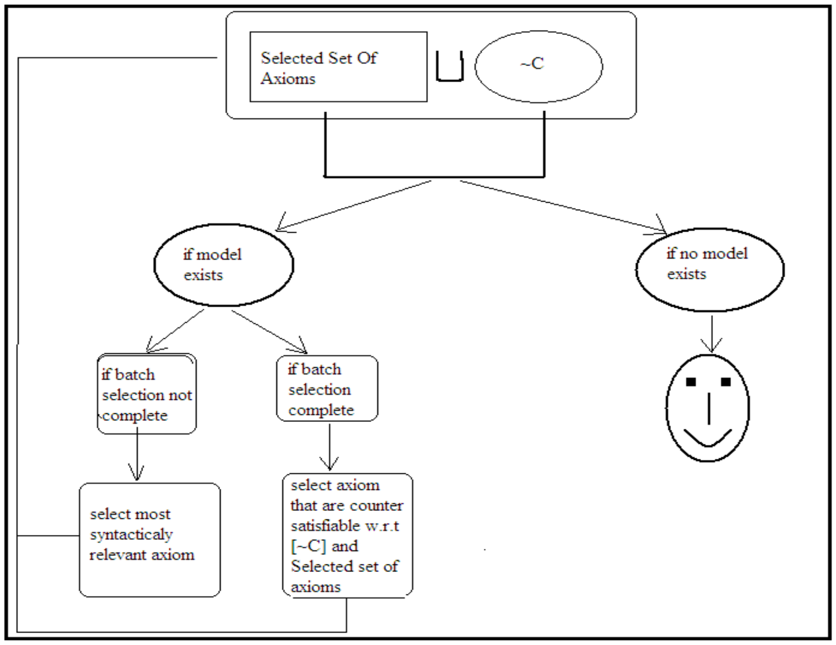

Selected Set Of Axioms
⊔
~C
if model exists
if no model exists
if batch selection not complete
if batch selection complete
select most syntacticaly relevant axiom
select axiom that are counter satisfiable w.r.t [~C] and Selected set of axioms

Figure 2.4.1: SRASSWorking

Similarities to this research:

- In both the research, sorted list of axioms are created syntactically before finding models.
- SRASS filters the axiom set by throwing away those axioms which are not *logical consequence* of a failed conjecture, which is also true for this research.

Dissimilarities from this research:

- In our research, we also refine the axiom set based upon the syntactic relevance measure of solved theorems w.r.to failed conjecture. This is not the case with SRASS.
- This research laid emphasis on learning by believing that newly solved theorems add certain knowledge to the problem domain, which can eventually be useful for some failed theorems that are closely related in terms of relevance to the newly solved theorems (see Chapter 3). This is again not true for SRASS. .

Chapter 3

Nearest Neighbor Learning

3.1 Basic Idea

As discussed in Chapter 1, if the number of available axioms is very large for any failed conjecture, then the probability of getting a proof for that failed conjecture (in a reasonable time limit) is very small. That's where the ability to learn from previously solved theorems comes into play. If we see in our own lives, whenever a new situation S(NEW) is encountered we try to recollect all old situations S(OLD) (which are similar to S(NEW)) in our neural system. Based on S(OLD), we then try to recollect all old facts F(OLD) related to S(OLD) and then try to find a solution for S(NEW). Figure 3.1(a) shows this scenario. The concept of nearest neighbor learning is similar, where we try to find similar solved theorems with respect to a failed conjecture.

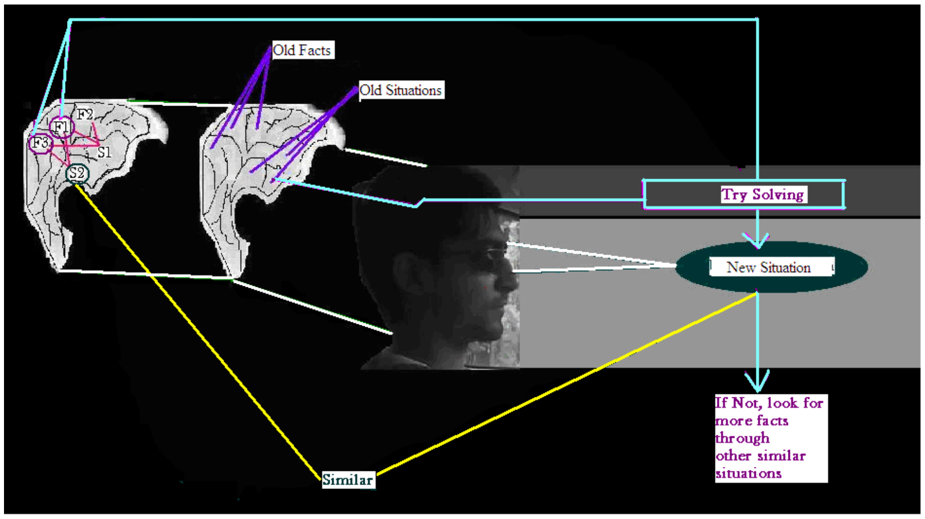


Figure 3.1(a) - Human logic for new situation

The basic idea of Nearest Neighbor Learning (NNL) is to compare an object in the domain of interest from the ones that are closely related. In this research, we apply the concept of finding closely related (in terms of relevance) solved theorems with respect to a failed conjecture. Figure 3.1(b) correlates the idea of human logic (mentioned in figure 3.1(a)) with the concept of NNL, which is implemented in this research.

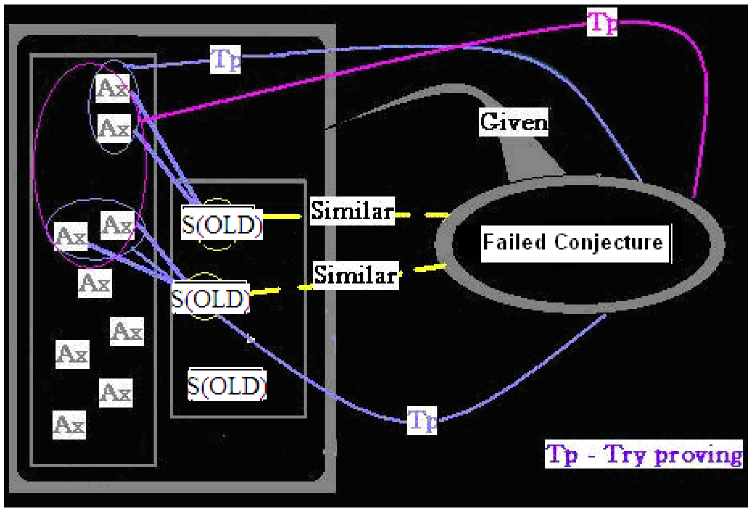


Figure 3.1(b) - Programming logic for new theorem

The process begins by assuming that there exist some *solved theorems* and *failed conjectures* in the problem domain. Learning then begins by picking one failed conjecture, and finding similar solved theorems with respect to the failed conjecture. The process then makes a list of sorted solved theorems (see Section 1.3.3) in order of *relevance* (calculated by Prophet, see Section 2.2.1) w.r.to the failed conjecture. Four *axiom refinement strategies* are introduced in this research for the purpose of making different axiom sets based on the following approaches:

- Sorted Solved Conjecture Relevance Set (SSCRS)
- Axiom Relevance Set (ARS)
- Sorted Conjectures by Average Axiom Relevance (SCAAR)
- One Axiom One Time (OAOT)

All the above mentioned approaches are implemented as an individual algorithm. The user is provided with an option of selecting the desired algorithm on the command line (see Chapter 5 for command line options). Detailed descriptions of all the above mentioned algorithms are given in Section 3.3.

3.1.1 Relevance Set

As described in Section 1.3.3, the relevance measure is first calculated between a failed conjecture and turned axioms. Taking a closer look in example 1.3.3a, we will observe that many turned axioms have the same relevance value. Therefore, in this research different batches are formed for a failed conjecture, by grouping all turned axioms with same relevance value in one batch. These batches are termed as *relevance sets*.

From example 1.3.3a, it also becomes clear that Prophet might assign different relevance values to turned axioms for a failed conjecture, and similar turned axioms are grouped in one relevance set. Thus for a failed conjecture, number of relevance sets is equal to the number of different relevance values produced by Prophet.

For instance in example 1.3.3a, there are three different relevance values for failed conjecture FC1:

First value	-	1.00
Second value	-	.99
Third value	-	.77

Therefore, for FC1 we have three Relevance Sets [ReS] with values 1.00, .99 and .77.

From the same example 1.3.3a, now it's obvious that the list of turned axioms in each relevance set will be:

-ReS 1.00 will contain ST1 and ST3

-ReS .99 will contain ST4

-ReS .77 will contain ST2 and ST5

There is a major limitation associated with this formation of relevance sets:

- Every failed conjecture will have various relevance sets associated with it (values ranging from 0.00 to 1.00). As it becomes obvious from Chapter 1 that highest relevance set value would mean the most similar relevance set for a failed conjecture. Therefore relevance sets with lower value might be of no use in the process of finding a solution for a failed conjecture (discussed later in this Chapter), and using these relevance sets might just be a waste of time. For the purpose of dealing with this limitation, the user is provided with an option of setting a threshold on relevance values, which we call the *relevance set limit*. All the relevance sets whose relevance value falls below relevance set are not used in the process of finding a solution for a failed conjecture.

For instance consider this **example 3.1.1a**

Failed conjecture list	:	FC1
Turned axioms list	:	ST1, ST2, ………………… , ST105

Now, suppose after giving FC1 and ST [1 - 105] to prophet we get 42 relevance sets:

1.	ReS[1.00]	:	ST1,ST2
2.	ReS[.80]	:	ST3,ST4
3.	ReS[.60]	:	ST5,ST6
4.	ReS[.40]	:	ST7
5.	ReS[.39]	:	ST8,ST9
.			
42.	ReS[0.00]	:	ST[60-105]

Note: Relevance sets from 6 to 42 are of value between 0.39 and 0.00.

By observing the above output, we can clearly see that the closest relevance set values with respect to the FC1 are much less compared to the ones that are farthest, and we also know that lower relevance set values might be of no use to FC1. Therefore, we should try avoiding as many lower values as possible. In the example above, if the user provides relevance set limit of .40, it will take into account top four closest relevance sets for FC1, and will avoid the 38 remaining relevance sets. This will save huge lot of time. For now the user will have to select the relevance set limit on his own, and intelligence for selecting relevance set limit should be considered as a future work for this research (listed in Chapter 6). Selecting '.40' is considered as a safe relevance set limit in this research, as it takes into account the top 60% of the relevance value for a failed conjecture.

3.1.2 Axioms Extraction

After making relevance sets of turned axioms, the next step is to make *refined axiom sets*. The process begins by converting turned axioms into solved theorems and then iteratively extracting *used axioms* from all *solved theorems* belonging to a particular relevance set.

For instance in example 1.3.3a, we have five turned axioms named ST[n] (where n = 1 to 5). Let's assume that by turning them into solved theorems, we get:

Solved theorem from ST1	:	SST1
Solved theorem from ST2	:	SST2
Solved theorem from ST3	:	SST3
Solved theorem from ST4	:	SST4
Solved theorem from ST5	:	SST5

The next step would be to extract the used axioms for these solved theorems. Suppose we receive following list (shown in Table 3.1.2) of used axioms for the solved theorems listed above:

Relevance Set	Solved Conjecture	Axioms used for proving it.
ReS 1.00	SST1	Ax1,Ax2,Ax3,Ax4,Ax5
ReS 1.00	SST3	Ax51,Ax3,Ax2
ReS .99	SST4	Ax6,Ax8,Ax99,Ax67,Ax4
ReS .77	SST2	Ax,5,Ax2,Ax4,Ax9
ReS .77	SST5	Ax8,Ax2,Ax99

Table 3.1.2 : Axiom Extraction

From the table above, it becomes clear that the total number of axioms in the relevance set with value 1.00 (ReS 1.00) will be 6 and the axiom list (say AxL1) for relevance set [ReS 1.00] will be Ax1, Ax2, Ax3, Ax4, Ax5 and Ax51. Similarly, we will also have axiom lists (AxL[1-n] where n is the total number of relevance sets) for the remaining relevance sets.

As by now we know that in the process of finding a solution for a failed conjecture, we first pick the top most relevance set, and then extract all axioms for it. For example in above case first picked relevance set will be [ReS 1.00] and extracted axioms will be Ax1, Ax2, Ax3, Ax4, Ax5 and Ax51. Therefore these six axioms will form the first axiom list, which will then be given to the *axiom refining strategy* (discussed in Section 3.3) for refining axioms, and making *proof attempt* on FC1. If the proof is not found by [ReS 1.00], then the next relevance set will be picked, which will be [ReS .99] for the example above. The union of the axioms from [ReS 1.00] and [ReS .99] will now be done so as to avoid duplicate axioms, and the union list (for the example above, union list will be: Ax1, Ax2, Ax3, Ax4, Ax5, Ax6, Ax8, Ax51, Ax67, Ax99) is given to the axiom refining strategy.

For the above mentioned process, there is a major limitation associated with it. In the ATP world, for any failed conjecture there is a list of *available axioms*. For a failed conjecture all axioms other than the available axioms cannot be used and are called *out of box axioms*. The union list produced from the above process might contain out of box axioms, as the union list is formed with the help of used axioms from different solved theorems. Considering the issue, we need to make sure that axioms in the union list do not contain any out of box axioms. The out of box axioms issue is discussed below in more detail:

Out of Box Axioms (OBAx)

Process of making *relevant axiom sets* involves the extraction of used axioms from all solved theorems. Now, consider Figure 3.2.5, where C2 is the failed conjecture, S3 and S5 are the solved theorems in some relevance set (say [ReS .84]). As the figure shows, the outer circle at the top right consists of all the available axioms for the solved theorem S3 (Axioms3 (all)), and the inner circle on the top right shows all the used axioms in the proof of solved theorem S3 (Axioms3 (used)). Similarly, the bottom right circles shows all available axioms (Axioms 5 (all)) and used axioms (Axioms 5 (used)) for solved theorem S5. The left most circle shows all the available axioms for a failed conjecture C2 (Axioms2 (all)). Thus, for the purpose of proving C2, any axiom which is not available, but exists in the used axiom list ((Axioms3 (used)) and (Axioms5 (used))) of solved theorems (S3 and S5), are out of box axioms. In Figure 3.2.5, the dark area represents out of box axioms for C2. All these axioms are not used in the process of finding a proof.

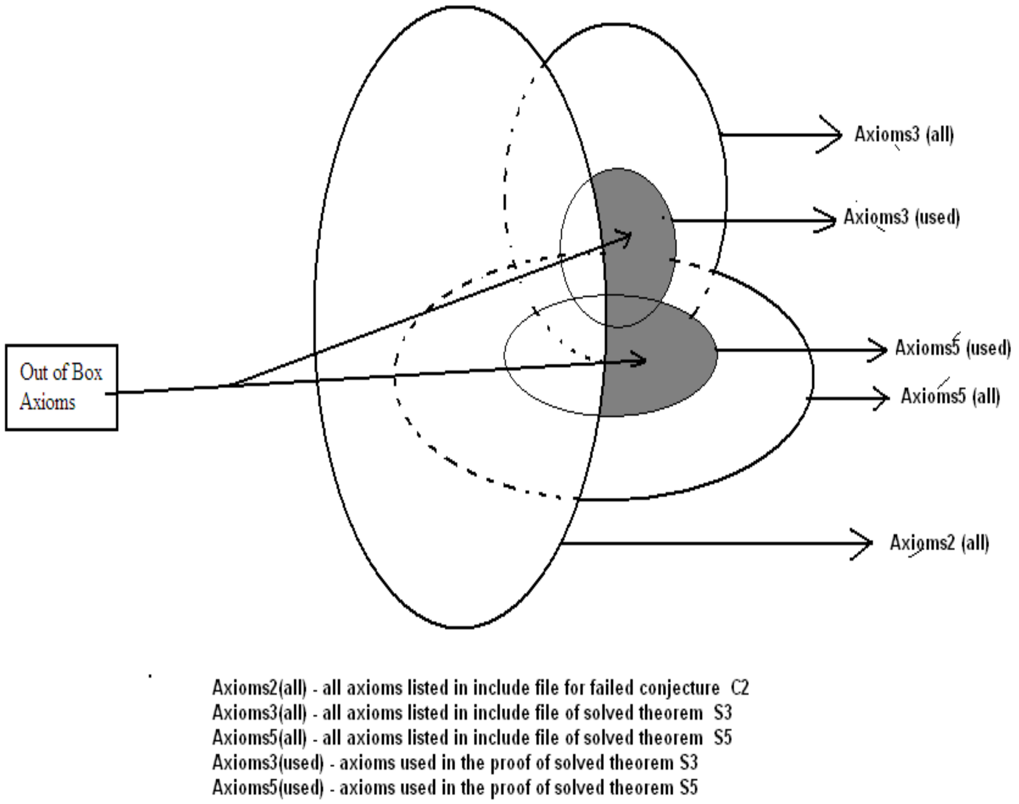

Out of Box Axioms
Axioms3 (all)
Axioms3 (used)
Axioms5 (used)
Axioms5 (all)
Axioms2 (all)
Axioms2(all) - all axioms listed in include file for failed conjecture C2
Axioms3(all) - all axioms listed in include file of solved theorem S3
Axioms5(all) - all axioms listed in include file of solved theorem S5
Axioms3(used) - axioms used in the proof of solved theorem S3
Axioms5(used) - axioms used in the proof of solved theorem S5

Figure 3.2.5: Out of Box Axioms

Therefore, for the purpose of making a proof attempt on any failed conjecture we define *Relevant Axiom Set (RAxS)* as:

Relevant axiom set = Intersection of [Union of (previous relevance axiom set, current relevance axiom set] and [available axioms of the failed conjecture].

3.1.3 The complete - Basic Learning Process (BLP)

The complete process of learning from solved theorems and making relevant axiom sets for a failed conjecture can be defined as: Generation of relevance sets with the help of solved theorems, which are then used to make relevant axiom sets so as to make the proof attempt on a failed conjecture.

The process begins by extracting all solved theorems from solution files, and converting them into turned axioms. Then relevance sets are generated using the relevance measures of turned axioms with respect to a failed conjecture. Finally, relevance sets are picked iteratively and turned axioms are converted back into solved theorems for the purpose of making relevant axiom sets. A proof attempt is then made on the failed conjecture with the relevant axiom set, and if proof is found for the failed conjecture then the problem domain is updated, and a new solved theorem gets added to the domain. This new solved theorem adds some more knowledge to the domain, and this knowledge will become applicable in the next MLAR run (discussed in more detail in Section 3.2.4). Therefore, to take advantage of the expanded domain of solved problems, the user has to restart MLAR. Now, if proof is not found and the relevance set limit is reached, then the process terminates itself for the current failed conjecture, and the next failed conjecture is picked. We call this process BLP. Figure 3.1.1 below shows its pictorial representation:

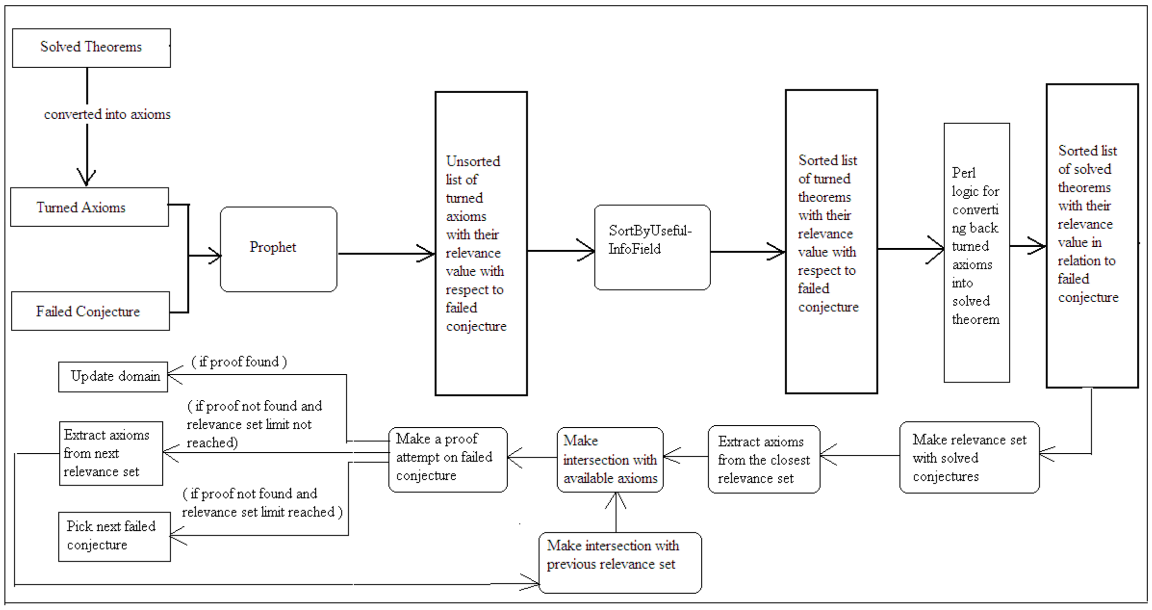

Solved Theorems
converted into axioms
Turned Axioms
Failed Conjecture
Prophet
Unsorted list of turned axioms with their relevance value with respect to failed conjecture
SortByUseful-InfoField
Sorted list of turned theorems with their relevance value with respect to failed conjecture
Perl logic for converting back turned axioms into solved theorem
Sorted list of solved theorems with their relevance value in relation to failed conjecture
Make relevance set with solved conjectures
Extract axioms from the closest relevance set
Make intersection with available axioms
Make a proof attempt on failed conjecture
Make intersection with previous relevance set
(if proof found)
Update domain
(if proof not found and relevance set limit not reached)
Extract axioms from next relevance set
(if proof not found and relevance set limit reached)
Pick next failed conjecture

Figure 3.1.3: BasicLearningProcess

The relevant axiom sets generated in the process of BLP can be refined to a finer level by using axiom refinement strategies (SSCRS, ARS, SCAAR and OAOT), which are discussed in Section 3.3. Now for the purpose of BLP consider **example 3.1.3.1a** mentioned below.

Example 3.1.3.1a:-

List of failed conjectures : FC2, FC3

Suppose for a failed conjecture FC2 there exist five relevance sets.The list of axioms for all relevance sets for FC2 is shown in table 3.1.4 below:

S.No	Relevance set	Extracted axioms for corresponding relevance set
1	ReS 1.00	Ax1,Ax2,Ax4

S.No	**Relevance set**	**Extracted axioms for corresponding relevance set**
2	ReS .99	Ax3,Ax5,Ax7
3	ReS .40	Ax1,Ax2
4	ReS .30	Ax9,Ax5
5	ReS .10	Ax8,ax2

Table 3.1.4 – BLP

Now suppose the relevance set limit = .50

BLP process will be:

1	Picked relevance set 1	:	ReS 1.00
2	Formed relevant axiom set 1	:	Ax1, Ax2, and Ax4
3	Proof attempt		
4	Output – not proved		
5	Relevance set limit not reached		
6	Picked next relevance set 2	:	ReS .99
7	Formed next relevant axiom set	:	Ax1, Ax2, Ax3, Ax4, Ax5 and Ax7
8	Proof attempt		
9	Output - not proved		
10	Relevance Set limit reached.		
11	Pick next failed conjecture: FC3		
12	Process continues ……		

3.2 Fine Tuning

3.2.1 Broken vs. Failed Problems

By analyzing the problem domain, it became clear that some of the solution files corresponding to each problem file do not contain proof output, instead they contain assurance as an output. As already defined, this research treats all such solution files (which have assurance as output) as broken problems. The above fact also made us realize that if in a particular problem domain, there exist failed problems and broken problems, and then it is more likely that broken problems may be solved more quickly than the failed ones, as the assurance of proof already exists. Thus, making a proof attempt on these broken problems before the failed ones increases the chances of adding some more solved problems to the domain. This might eventually lead to the solution of more failed problems (from the same domain), as it might be a case that many of the failed problems are closely related to some of the broken problems.

3.2.2 Same axiom set

In the process of BLP a special case may occur, when the recently generated list of extracted axioms is same as the previous one. For instance consider the example below: Suppose for a failed conjecture FC2 there exists five relevance sets. Table 3.2.2 shows these relevance sets with their corresponding axioms:

S.No.	**Relevance set**	**List of axioms**
1	ReS 1.00	Ax1,Ax2,Ax4

S.No.	Relevance set	List of axioms
2	ReS 0.99	Ax3,Ax5,Ax7
3	ReS 0.60	Ax1,Ax2
4	ReS 0.50	Ax9,Ax5
5	ReS 0.30	Ax8,ax2

Table 3.2.2 – Same Axiom Set

Now suppose the BLP process for FC2 is:

1	Picked relevance set 1	:	ReS 1.00
2	Formed relevant axiom set 1	:	Ax1, Ax2, and Ax4
3	Proof attempt		
4	Output – not proved		
5	Picked next relevance set 2	:	ReS 0.99
6	Formed next relevant axiom set	:	Ax1, Ax2, Ax3, Ax4, Ax5 and Ax7
7	Proof attempt		
8	Output - not proved		
9	Picked next relevance set 2	:	ReS 0.60
10	Formed next relevant axiom set	:	Ax1, Ax2, Ax3, Ax4, Ax5 and Ax7

Now after the 10th step it makes no sense to make a proof attempt for FC2 with the current relevant axiom set. Simply proceed to the next relevance set.

3.2.3 Multiple ATP systems.

As discussed before, different ATP systems are capable of proving different problems. For example, some of them might be good with equality problems but others may not. Considering the case above, this research integrates the use of multiple ATP systems for the purpose of making a proof attempt on a failed conjecture.

Three ATP systems are required to be specified in the implementation, which can be changed as per user requirements. For the purpose of changing ATP systems in the implementation please see the *used ATP system* subsection under Section 5.1. After specifying the ATP system in the implementation, the user is given an option of making a maximum of three proof attempts with the help of the different ATP systems on a failed conjecture, from same axiom set. The number of ATP systems to be used can be specified from the command line options (see Chapter 5).

Since there are three ATP systems specified in the implementation, we therefore have seven possible ways of selecting different ATP systems. The example below, explains how this becomes possible:

If we call the three ATP systems specified in implementation as:

ATPsystemOne

ATPsystemTwo

ATPsystemThree

Then the different combination of above three ATP systems will provide user with following options to choose from:

-using only ATPsystemOne

-using only ATPsystemTwo

-using only ATPsystemThree

-using ATPsystemOne and ATPsystemTwo

-using ATPsystemTwo and ATPsystemThree

-using ATPsystemThree and ATPsystemOne

-using ATPsystemOne, ATPsystemTwo and ATPsystemThree

Suppose user picked an option of using ATPsystemOne and ATPsystemThree. Then there will be two proof attempts on a failed conjecture with the same axiom set. A proof attempt will be first made by using ATPsystemOne. If no proof is found then the control flows to ATPsystemTwo for the next proof attempt. Figure 3.2.3 below shows the pictorial representation of the control flow for this option.

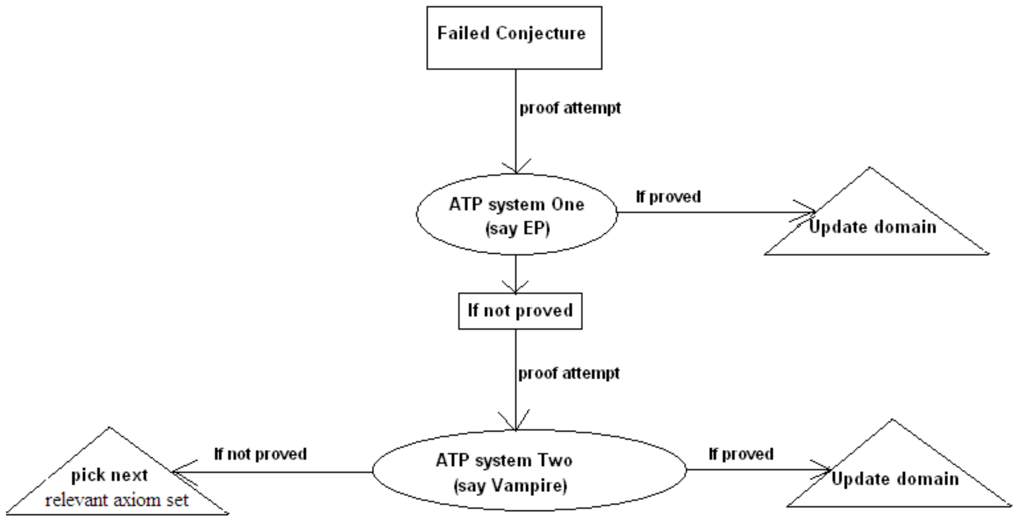


Figure 3.2.3: Multiple ATP system

3.2.4 Update Domain.

This research extensively depends on solved theorems for the purpose of finding a solution of a failed conjecture. It also believes that every addition of a solved theorem to the problem domain increases the knowledge base for the remaining failed conjectures, and thereby increasing the probability of them getting solved. Therefore, it becomes essential to update the problem domain with every new solved theorem.

In this research, whenever a failed conjecture is proved, its proof output is immediately added to the corresponding solution file in problem domain. Also, different statistics (like time limit in which it got proved, number of axioms given to ATP system for finding proof) related to the newly solved theorem are updated in a separate file (the location of this file in the implementation is described in Chapter 5).

Though it is true that every new solved theorem adds some knowledge to the problem domain, at the same time we also need to consider a fact that if the number of failed problems is huge then adding one new solved theorem might not make a big difference. Thus for the same purpose, the user is provided with an option of specifying the number of failed problems to be attempted before using the updated knowledge from domain. For using the knowledge from expanded domain of solved theorems, the user has to restart the tool, and specify the number of problems that he/she wants to attempt. An option is provided on the command line, which is discussed in Chapter 5.

3.2.5 Desperate attempt to prove

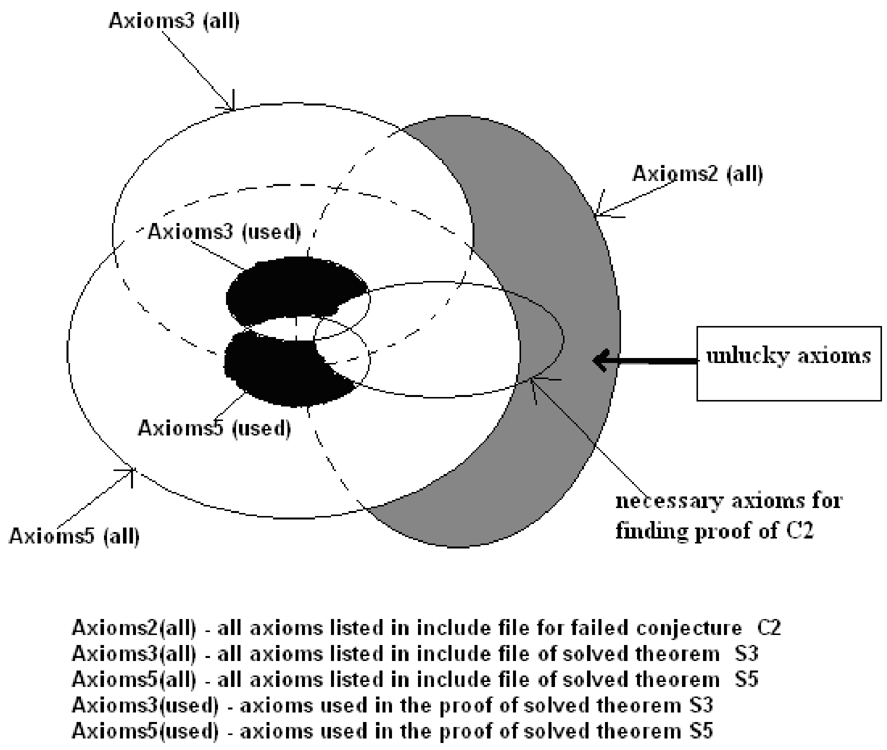
As by now we know that for any failed conjecture, relevant axiom sets are formed with the help of used axioms from solved theorems. In the ATP world, used axioms for any solved theorem are the subset of its available axioms. Every failed conjecture (say C2) also has a predefined list of available axioms. Now it can be the case that some available axioms of a failed conjecture (C2) are not available for any solved theorem (we call this set of axioms the *unlucky axioms*). Unlucky axioms can never exist in any relevant axiom set and can never be used in a proof attempt of C2. At the same time, it can be true that some of the unlucky axioms are required to find a proof for C2. Thus we consider all unlucky axioms in the last proof attempt for C2, and we call this last attempt as *desperate attempt to prove*. The user is provided with an option of switching on or off, the desperate attempt to prove.

Now consider figure 3.2.6, we have:

Failed conjecture	:	C2
Solved theorems	:	S3, and S5

In the Figure 3.2.6 Axioms3 (all) and Axioms5 (all) are the available axioms for solved theorems S3 and S5 respectively. Similarly Axioms3 (used), and Axioms5 (used) are the used axioms for S3 and S5 respectively. For the failed conjecture C2, Axioms2 (all) depict it's available axioms. Taking a closer look in the figure, we observe that the area depicting the *necessary axioms for finding the proof for C2* falls out of Axioms3 (all) and Axioms5 (all). This means that some of the necessary axioms for C2 are not available for

S3 and S5, therefore a proof of C2 cannot be found by using used axioms of S3, and S5. To avoid such a case we consider all unlucky axioms in the last proof attempt for C2.



Axioms2(all) - all axioms listed in include file for failed conjecture C2
Axioms3(all) - all axioms listed in include file of solved theorem S3
Axioms5(all) - all axioms listed in include file of solved theorem S5
Axioms3(used) - axioms used in the proof of solved theorem S3
Axioms5(used) - axioms used in the proof of solved theorem S5

Figure 3.2.6: Desperate attempt to prove

A desperate attempt to prove can increase the number of axioms to a very large value. Now, as discussed in Chapter 4, semantic selection of axioms works by picking one axiom at a time, therefore if the axiom list is very large then the semantic approach will take huge amount of time.

3.3 Axiom Refining Strategies

As discussed before we have defined four axiom refinement strategies:

1 Sorted solved conjecture relevance set.

2 Axioms relevance set.

3 Sorted solved conjectures by average axiom relevance.

4 One axiom one time.

The user is provided with an option to select any one of the above mentioned strategies so as to make a refined axiom set for the purpose of finding a solution for a failed conjecture. The selection of an algorithm is done through the command line, which is explained in Chapter 5. The sole purpose of introducing all these strategies is to refine the relevant axiom set produced by BLP, and make it a refined axiom set before making a proof attempt. All these strategies are discussed below in more detail.

3.3.1 Sorted Solved Conjecture Relevance Set (SSCRS).

This axiom refining strategy produces refined axiom set based on the relevance value of solved theorems with respect to a failed conjecture.

As discussed in BLP, relevance sets are generated based on sorted solved theorems, followed by extraction of axioms in these relevance sets. After the axioms are extracted, duplicate entries are avoided as discussed in Section 3.1.2. As soon as the duplicity is removed, SSCRS then sorts this axiom list based on their relevance value (calculated through prophet) w.r.to the failed conjecture (discussed in Section 1.3.3).

Figure 3.3.1(a) shows the pictorial representation:

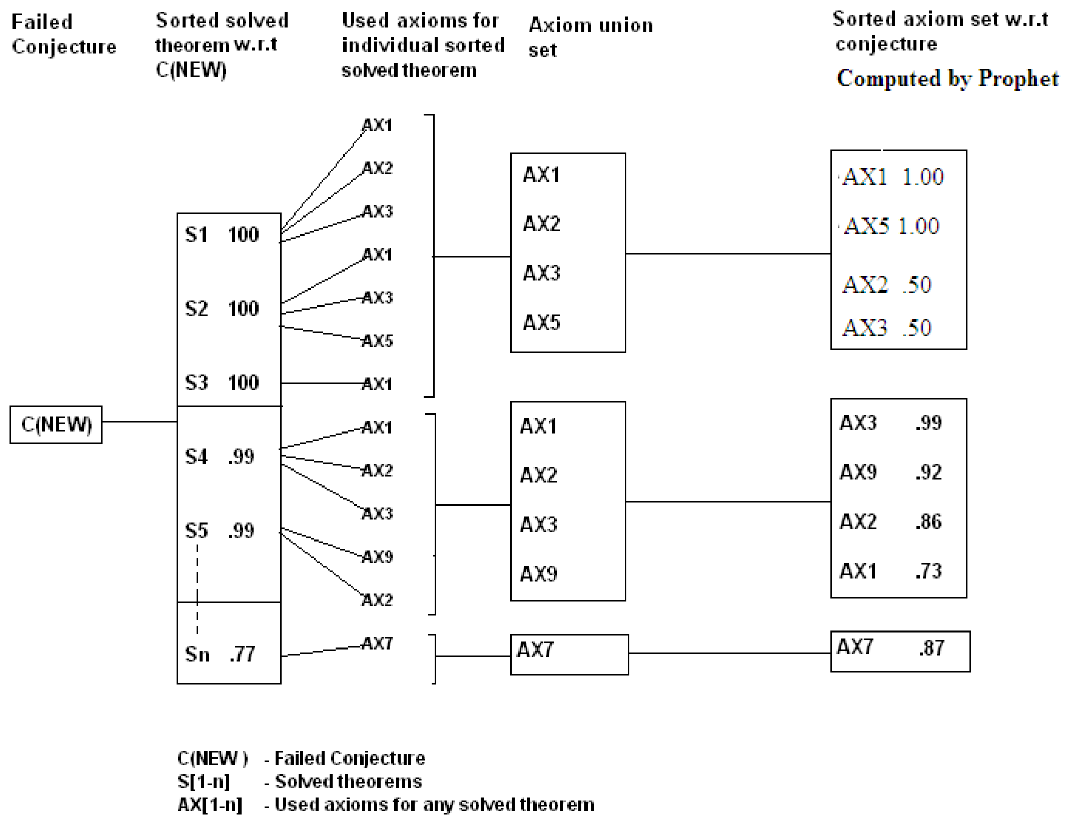


Figure 3.3.1(a): Graphical SSCRS

As shown in Figure 3.3.1(a), C(NEW) is picked as a failed conjecture, and the *axiom union sets* (the fourth column in Figure 3.3.1(a)) are generated (to generate axiom union set, SSCRS uses the strategy discussed in Section 3.1.3). This axiom set is then sorted (with the help of tool called SortByUsefulInfoField) with respect to C(NEW). The sorted axiom set is not used directly for the purpose of making a proof attempt on a failed conjecture rather it is generated as an input for the semantic selection approach (discussed in Chapter 4). In SSCRS, this sorted axiom set is the refined axiom set.

3.3.2 Axioms Relevance Set (ARS).

This axiom refining strategy produces refined axiom sets based on the relevance values of used axioms with respect to the failed conjecture.

The process in ARS remains same as SSCRS until the generation of the *sorted axiom union set* (the fourth column in Figure 3.3.2(a)). As soon as a sorted axiom set is generated, different *sorted axiom relevance sets* (the fifth column in Figure 3.3.2(a)) are formed based on the relevance value of axioms in the sorted axiom union set w.r.to C(NEW). The sorted axiom relevance sets are generated by grouping axioms with same relevance in same set, and then sorting relevance sets with the help of the tool SortByUsefulInfoField.

Figure 3.3.2(a) below shows the pictorial representation:

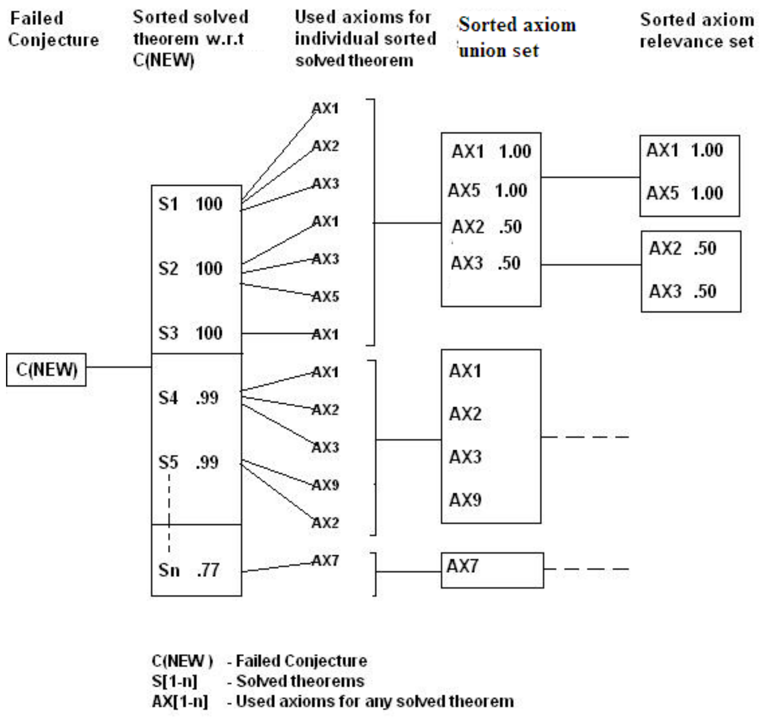


Figure 3.3.2(a): Graphical ARS

As shown in figure, C(NEW) is picked as a failed conjecture. By using the BLP process, the *used axioms for individual sorted conjecture* (extracted axioms list) are generated with the help of *sorted solved conjecture list* w.r.to *C(NEW)*. Duplicity is then removed and relevance sets are formed with respect to axioms as shown in *sorted axiom union set*. Axioms with the same relevance value are grouped together in *sorted axiom relevance sets*. In ARS, the sorted axioms relevance sets are the refined axiom sets.

3.3.3 Sorted Conjectures by Average Axiom Relevance (SCAAR).

This axiom refining strategy produces refined axiom sets by sorting the solved theorems list in a particular relevance set based on the average relevance value of their used axioms with respect to the failed conjecture.

The process of SCAAR remains same (w.r.to ARS and SSCRS) until the extraction of used axioms, followed by the extraction of their relevance value with respect to the failed conjecture. After establishing the relevance values of used axioms, solved theorems in a particular relevance set are picked iteratively and the average of their used axioms is calculated.

For example, in Figure 3.3.3(a), consider first relevance set with relevance set value 1.00. This relevance set contains solved theorems S1, S2 and S3. Used axioms for S1 are AX1, AX2 and AX3. Similarly, used axioms for S2 are AX1, AX3 and AX5 and used axioms for S3 is AX9. Now suppose for S1, the relevance values for its used axioms AX1, AX2, and AX3 are 1.00, .50, and 1.00 respectively.

Therefore total number of used axioms for S1 = 3 and

Summation of relevance value = 1.00 + .50 + 1.00 = 2.5

The average value for S1 = 2.5 / 3 = .83

Similarly, the average value for solved theorems S2, and S3 is 1.00, and 0.7 respectively. Thus, by sorting the solved theorems (S1, S2, and S3) in relevance set with value 1.00, we get *sorted solved theorem w.r.t avg. axiom relevance* (the sixth column in Figure 3.3.3(a)). As soon as the list of sorted solved theorem is formed, refined axiom sets are generated by picking one solved theorems at a time (top to bottom approach), followed by extracting its used axioms. For example in figure 3.3.3(a), the sorted list of solved theorem is: S2 – S1 – S3. Thus S2 is picked first, followed by extraction of its used axioms AX1, AX3, and AX5. These used axioms of S2 will form the first refined axiom set in SCAAR. The second refined axiom set in SCAAR is the union of the used axioms from S2 and S1.

Figure 3.3.3(a) shows the pictorial representation of SCAAR:

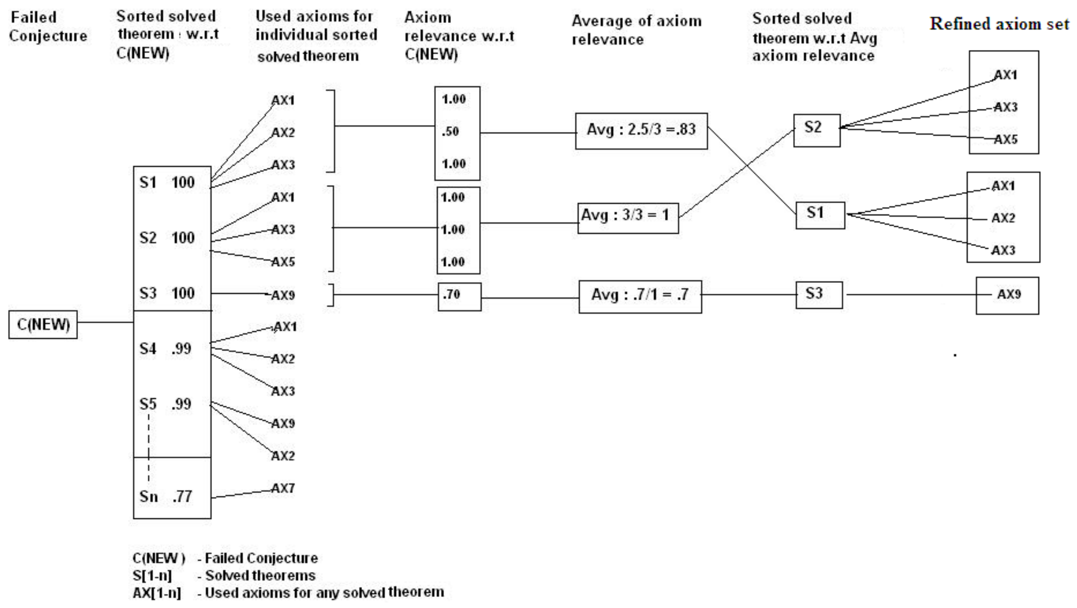

Failed Conjecture
Sorted solved theorem w.r.t C(NEW)
Used axioms for individual sorted solved theorem
Axiom relevance w.r.t C(NEW)
Average of axiom relevance
Sorted solved theorem w.r.t Avg axiom relevance
Refined axiom set
C(NEW)
S1 100
S2 100
S3 100
S4 .99
S5 .99
Sn .77
AX1
AX2
AX3
AX5
AX9
AX7
1.00
.50
.70
Avg : 2.5/3 =.83
Avg : 3/3 = 1
Avg : .7/1 = .7
S2
S1
S3
C(NEW) - Failed Conjecture
S[1-n] - Solved theorems
AX[1-n] - Used axioms for any solved theorem

Figure: 3.3.3(a)- Graphical SCAAR

3.3.4 One Axiom One Time (OAOT).

As discussed above, SSCRS generates sorted used axiom list based on their relevance value w.r.t a failed conjecture. SSCRS makes this complete sorted list as a refined axiom set, whereas OAOT generates refined axiom sets by picking one axiom (from top to bottom order) every time, and taking a union of this axiom with previously generated refined axiom set.

Figure 3.3.4(a) below shows the pictorial representation of OAOT:

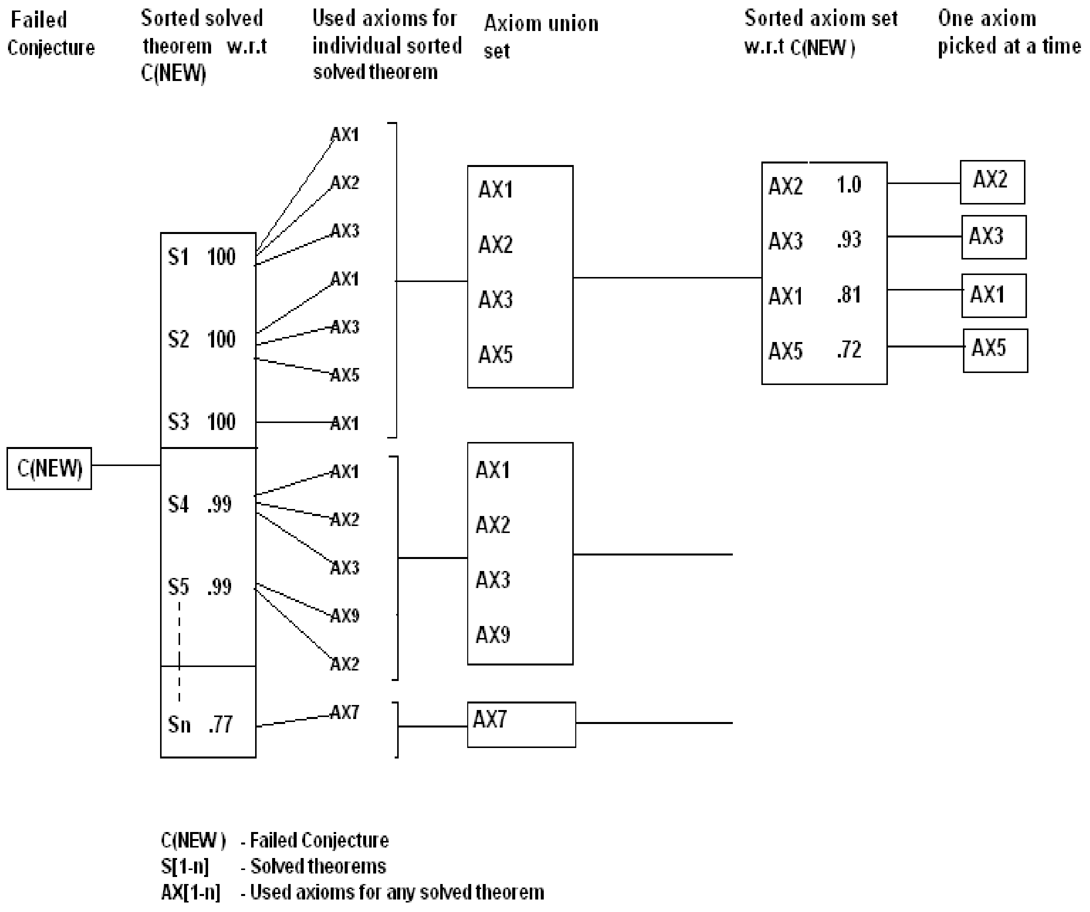

Failed Conjecture
Sorted solved theorem w.r.t C(NEW)
Used axioms for individual sorted solved theorem
Axiom union set
Sorted axiom set w.r.t C(NEW)
One axiom picked at a time
C(NEW)
S1 100
S2 100
S3 100
S4 .99
S5 .99
Sn .77
AX1
AX2
AX3
AX5
AX9
AX7
AX2 1.0
AX3 .93
AX1 .81
AX5 .72
C(NEW) - Failed Conjecture
S[1-n] - Solved theorems
AX[1-n] - Used axioms for any solved theorem

Figure: 3.3.4(a)- Graphical OAOT

For the example in Figure 3.3.4(a), the OAOT process remains same as SSCRS, until the formation of the *sorted axiom set w.r.t C(NEW)* (the fifth column in Figure 3.3.4(a)). As soon as the above mentioned axiom list is generated, refined axiom sets are generated by picking one axiom every time. For example in Figure 3.3.4(a), the first refined axiom set will have axiom AX2, second refined axiom set will have AX2, and AX3. Similarly the third will have AX2, AX3, and AX1, and so on.

When the next relevance set is picked in OAOT, the axiom list is first made. It is generated by making the union of axioms from the previous relevance set, and the current relevance set. It is then sorted to make a refined axiom set as discussed in the case of first relevance set (with value 1.00). For example in Figure 3.3.4(a), after the relevance set with value 1.00, we pick the second relevance set with value 0.99, followed by extracting axioms AX1, AX2, AX3, and AX9 from second relevance set. We then make a union of axioms from second relevance set and first relevance set, as a result of which we get: AX1, AX2, AX3, AX5, and AX9 (call this list SecRlist). Now for the purpose of making second refined axiom set, OAOT will sort SecRlist, and will start making refined axiom sets (as discussed in the case of first relevance set) by picking one axiom at a time from the sorted list.

3.4 Complete Algorithm

This section now presents a complete syntactic refinement algorithm, which is defined with the help of the following terminology:

- *Domain Analysis*: The process of extracting all broken problems, solved problems, failed problems and storing them in a broken problem set, solved problems set and failed problem set respectively is known as domain analysis.
- *Update Domain*: The process of updating the problem domain with newly solved theorems is known as update domain.
- *Batch Problems Set*: The number of failed conjectures selected by user for the purpose of proving before updating the domain. They are in order of their existence in problem domain.
- *Relevance Set list*: The list containing all relevance set values for a selected failed conjecture. Every value in it points to all solved theorems related to that value.

The complete algorithm for syntactic selection approaches is shown below:

```
1.   domain analysis
2.   while  batch problem set not empty.
3.       extract one failed conjecture from batch problem set;
4.       extract all solved theorems from problem domain into solved conjecture list;.
5.       convert all solved theorems into turned axiom and store them in turned
         axiom list;
6.       for all axioms in turned axiom list
7.           find relevance through prophet with respect to failed conjecture and store
             in solved theorem set;
8.       end
9.       sort solved theorem set by using tool SortByUsefulInfoField;
10.      make relevance sets from sorted solved theorem set and store them in
         relevance set   list;
11.      while relevance set limit not reached
12.          pick top most relevance set from relevance set list;
13.          generate refined axiom set from the selected axiom refining strategy;
14.          make union of current refined axiom set with previous
15.          make a proof attempt on failed conjecture with refined axiom set;
16.          if proof attempt successful
17.              update domain;
18.          else if proof attempt unsuccessful and relevance set limit not reached
19.              remove current relevance set from relevance set list;
```

```
        else if proof attempt unsuccessful and relevance set limit reached
            if user requested for desperate attempt to prove
                make desperate attempt to prove on selected failed conjecture;
                if proof found
                    update domain;
                else
                    print “desperate attempt failed”;
                end
            else
                print “[failed conjecture name] not proved”;
            end
        end
    end
end
```

Chapter 4

Use of Semantic Selection

4.1 Basic Idea

Although the axiom selection processes described in Chapter 3 gives a refined axiom set, it can still contain some irrelevant axioms. This gave the motivation for going down to much finer level in the process of axiom selection. As a result, this thesis implements semantic axiom selection approaches on the refined axiom set (generated by axiom refining strategies, discussed in Chapter 3).

In the *refined axiom set* received by any *axiom refinement strategy* (say SCAAR) described in Chapter 3, there exist a minimal subset of selected axioms (say AxLC), which might show that a failed conjecture is a *logical consequence* of them. In this research, we call AxLC a *perfect set*.

Remaining axioms (other than those in perfect set) in the refined axiom set are known as *discarded axioms*.

Discarded axiom set = [refined axiom set] - [perfect set]

The semantic approach is introduced for the purpose of selecting this perfect set of axioms. The axiom set produced by applying the semantic approach to the *refined axiom set* is known as a *semantically refined axiom set*.

4.1.1 Model Finding Strategy:

This strategy assumes that there exists a sorted refined axiom set generated by some axiom refining strategy (discussed in Chapter 3). In the process, one axiom is then picked iteratively (from top to bottom order) from the sorted refined axiom set and is treated as a conjecture within the process. This picked axiom is called a *checking axiom*. In the model finding strategy, we negate the failed conjecture and call it a *negated conjecture*. Semantic selection process then begins, and for the purpose of making a perfect set, a Model Finder (MF) tries to establish non logical consequence between negated conjecture, checking axiom as conjecture, and the perfect set.

By assuming that the perfect set is empty in beginning, it becomes clear that the union of the perfect set and a checking axiom as conjecture do not matter in the first run. There, can be three cases for checking axiom, based on the output produced by MF:

- Case 1: If the model finder is able to establish logical consequence, then checking axiom is called *discarded axiom*.
- Case 2: If the model finder is able to establish non logical consequence, then checking axiom is called *Perfect Axiom (PA)*.

- Case 3: If for any reason model finder is not able to establish either of the above two cases then the checking axiom is called a *Confused Axiom (CA),* the set of confused axioms is called the *confused set*. This may happen because of the limited resources available to MF, like a time constraint.

The goal for this approach is to make a perfect set. The process begins by assuming that the perfect set is empty. The MF is then provided with following list (we call it the *MF list*):

- Perfect set (which is initially empty)
- Picked checking axiom as conjecture.
- Negation of a failed conjecture as negated conjecture.

From the above list, MF will produce one of the following outputs:

- If with MF list MF is able to establish non logical consequence, then the checking axiom is added to the perfect set.
- If with MF list, MF is able to establish logical consequence, then the checking axiom is discarded.
- If with MF list, MF is not able to establish either of the above two cases, then checking axiom is added to confused set.

As soon as the above process is completed for the refined axiom set, the first proof attempt is made on the failed conjecture with the perfect set. If a proof is not found then a second proof attempt is made using the union of the confused set, and the perfect set.

Now as discussed before, different ATP system have different computational powers. Thus for the semantic selection approach, two model finders are considered. The user can specify model finders in the implementation of this research. For details of specifying model finders please see Chapter 5.

Figure 4.1.1 shows the process described above, with following consideration:

- Logical consequence is supposed to be established with MF list, when any of the MF produces Theorem (THM) as output.
- Non logical consequence is supposed to be established with MF list, when any of the MF gives CounterSatisfiable (CSA) as output.
- If both the above cases does not occur, and MF gives Time Out (TMO) as output then checking axiom is called as confused axiom.

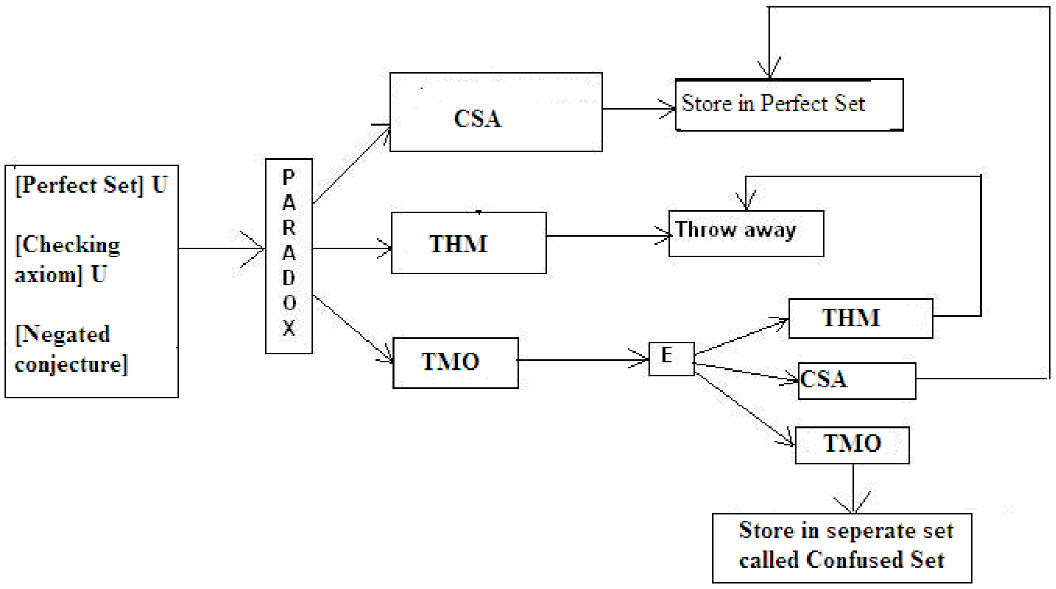


Figure 4.1.1: MF Strategy

4.2 Modified Algorithm with Semantic Selection

Below is the modified algorithm (discussed in Chapter 3) with semantic axiom refinement strategy:

```
1.  domain analysis
2.  while batch problem set not empty.
3.      extract one failed conjecture from batch problem set;
4.      extract all solved theorems from problem domain into solved conjecture list;
5.      convert all solved theorems into turned axiom and store them in turned
        axiom list;
6.      for all axioms in turned axiom list
7.          find relevance through prophet with respect to failed conjecture and
            store in solved theorem set;
8.      end
9.      sort solved theorem set by using tool SortByUsefulInfoField;
10.     make relevance sets from sorted solved theorem set and store them in
        relevance set list;
11.     while relevance set limit not reached
12.         pick top most relevance set from relevance set list;
13.         generate refined axiom set from the selected axiom refining strategy;
14.         make union of current refined axiom set with previous.
15.         while refined axiom set not empty
16.             apply model finding strategy;
17.         end
18.         Store all perfect axioms generated by model finding strategy into
            perfect set;
19.         Store all confused axioms generated by model finding strategy into
            confused set;
20.         make a proof attempt on failed conjecture with perfect axiom set;
21.         if proof attempt successful
22.             update domain;
23.         else
24.             make a proof attempt on failed conjecture with confused axiom set;
25.             if proof attempt successful
26.                 update domain
27.             else if proof attempt unsuccessful and relevance set limit not
                reached
28.                 remove current relevance set from relevance set list;
29.             else if proof attempt unsuccessful and relevance set limit reached
30.                 if user requested for desperate attempt to prove
31.                     make desperate attempt to prove on selected failed
                        conjecture;
```

```
                    if proof found
                        update domain;
                    else
                        print "desperate attempt failed";
                    end
                else
                    print "[failed conjecture name] not proved";
                end
            end
        end
    end
end
```

Chapter 5

Implementation and Testing

5.1 Implementation and User Perspective

The tool has been implemented for the purpose of proving failed conjectures with the help of Syntactic Axiom Refinement Strategies (SARS (see Chapter 3)), and semantic axiom selection (see Chapter 4). Syntactic axiom refinement strategies are implemented independent of each other, but semantic axiom selection is dependent on SARS.

The implementation is done in Perl, so the user system should support Perl scripts. This tool runs some of the shell commands from the main Perl script hence it's important that the interface, and location from which user is trying to run the tool, should support shell scripts. As an initial step, tool tries to analyze the problem domain, and extract axioms by using tools from TPTP world. Thus, the user has to make sure that the execution of TPTP commands (running the ATP system, model finder etc.), and use of tools (Prophet [8], SortByUsefulInfoField [23], ProofSummary [24]) is possible without any interruption of user privileges, and access permissions.

The tool comes with one main directory (the user is allowed to change name of this directory and set the path for installation directory in the code file (discussed below)). In this research, the installation directory is called `Work`, which contains two sub-directories `Domain`, and `FilesUsedInCode`, and one code file `LearnAndProve.pl`. Figure 5.1, shows the tree structure for the installation:

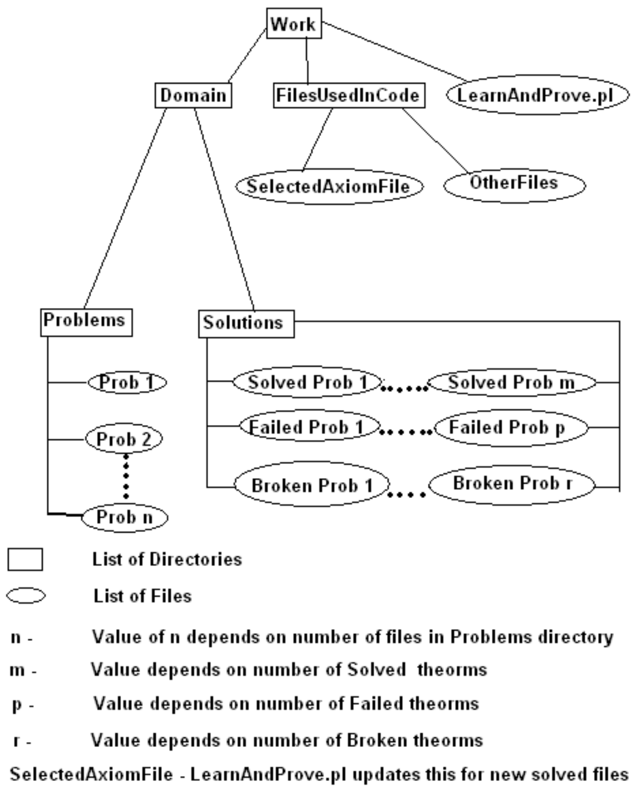

Work
Domain
FilesUsedInCode
LearnAndProve.pl
SelectedAxiomFile
OtherFiles
Problems
Solutions
Prob 1
Prob 2
Prob n
Solved Prob 1
Solved Prob m
Failed Prob 1
Failed Prob p
Broken Prob 1
Broken Prob r
List of Directories
List of Files
n - Value of n depends on number of files in Problems directory
m - Value depends on number of Solved theorms
p - Value depends on number of Failed theorms
r - Value depends on number of Broken theorms
SelectedAxiomFile - LearnAndProve.pl updates this for new solved files

Figure 5.1: ToolFileSystem

The `Domain` directory contains two more sub-directories called `Problems`, and `Solutions`. The sub-directory `Problems` should contain all the original *problem files* from the selected TPTP problem domain. `Solutions` sub-directory should contain *solution*, and *failed files* for the corresponding problem files. `FilesUsedInCode` sub-directory contains necessary files, which are updated or modified during tool execution. The user is advised not to delete any file from the `FilesUsedInCode` sub-directory. The code file should also be updated before use. Below is the list of updates required in the code file:

- **Installation Directory**

Since the tool uses files from the directory called `FilesUsedInCode`, therefore it's important to know whether the installation directory has been changed from what is listed in the code file. For the purpose of changing installation directory, user will have to change the variable value called `$InstallationDirectory` to the location where code file, and the sub-directories `Domain`, and `FilesUsedInCode` are located. Current value for above variable in the code looks like:

```
$InstallationDirectory = "~/Desktop/Work";
```

- **Service Tools**

Service tools used by MLAR from TPTP world are listed below:

1. `Prophet`
2. `SortByUsefulInfoField`
3. `ProofSummary`
4. `tptp4X`

User will have to make sure that the tool knows the exact location for all of them. For the same purpose, following variables have to be changed in the code file:

```
$ToolForRelevance = "/home/graph/tptp/ServiceTools/prophet -p";
$SortByRelevance = "/home/graph/tptp/ServiceTools/JJUsers/SortByUsefulInfoField relevance -f";
$ToolPath_ProofSummary = "/home/graph/tptp/ServiceTools/ProofSummary -f parents";
$tptp4X_path = "/home/graph/tptp/ServiceTools/tptp4X";
```

- **System on TPTP**

MLAR uses `SystemOnTPTP` in between tool run. For the purpose, it's again essential to make sure that we have correct path for `SystemOnTPTP` listed in the code file. The variable which needs to be changed is listed below with current location:

```
$SystemOnTPTP_path = "/home/graph/tptp/SystemExecution/SystemOnTPTP";
```

- **Used ATP system**

The tool uses three ATP systems for the purpose of theorem proving and 2 ATP systems for the purpose of model finding. If user wants to change the version of ATP system or the ATP system itself, he can do so by changing following variable:

ATP systems for theorem proving:-

```
$TheoremChecker1 = "SPASS---3.01";
$TheoremChecker2 = "Vampire---SUMO";
$TheoremChecker3 = "EP---1.1pre";
```

ATP systems for model finding:-

```
$TheoremDisprover1 = "E---1.1pre";

$TheoremDisprover = "Paradox---3.0";
```

5.1.1 Command line:

```
perl LearnAndProve.pl [-t]  [-m]  [-r]  [ -n]  [-a]  [-p]
```

-t : ATP system time limit

[Default is 180]

[Reasonable range can be from 0-600]

-m : Model finder time limit (as per my observation, setting it more than 30 will cause huge time wastage and very little effect on output.)

[Default: 30]

[Reasonable range can be from 0-600]

-r : Relevance set limit

[Default: .40]

[Range: 0.00 – 1.00]

-n : Number of failed theorems user want to try at once before updating the problem domain

[Default: 10]

[Range: depending upon how big is the problem domain]

-a : Algorithm for refining the set of axioms

[Default: `RelevanceSet`]

[Options: `RelevanceSet, AxiomsRelevanceSet, SortedConjByAxiomRelevance, OneAxiomOneTime`]

- p : ATP system to be used while testing

[Default: `FirstTC`]

Example:

```
perl LearnAndProve.pl –t 30 –m 30 –r .40 –n 20 –a RelevanceSet –p FirstTC
```

NOTE: Turning semantic selection ON or OFF: If for any purpose user wish to disable the process of semantic selection, it can be done by setting the value of –m to zero.

5.1.2 Running the tool:

Running the tool for proving the failed theorem set is an easy 5 step process:

- **Command line**

  Type in the command line. For command line options, and usage please see Section 5.1.1.

- **Analyzing the Problem Domain.**

  As soon as the user types in the correct command line options, the tool will try to analyze problem domain. It will extract the total number of failed problems, and broken problems existing in the problem domain.

- **Selecting the Problem Set**

  As soon as the tool is done analyzing the problem domain, which may take few seconds, it will show the number of failed problems and broken problems in the problem domain. It will then prompt user to choose one of them.

- **Union of Axioms from Include File**

  There can be a special case when a failed conjecture is not proved after the relevance limit is reached, this is where the concept of a desperate attempt to prove can be used (for details please see Section 3.2.5). The user can turn on or off this attempt, but this has to be done before starting of the main process. Therefore the tool will (after the user made a selection of problem set) prompt the user, asking for turning on or off this attempt.

- **Leave it by itself**

  As soon as the user gives input for an desperate attempt to prove, the tool gets initialized and will start picking up one problem at a time, and based on the options selected (like algorithm selection, time limits, theorem set, ATP system and number of theorems to be attempted at once) by the user it will start the process of making refined axiom sets, and proof attempts on failed conjectures.

Thus, it's the time for the user to sit, and relax if he has provided a huge time limits and huge number of problems to be attempted at once (before updating the domain). If that is the case then, user might have to keep himself occupied in some other stuff for may be a week or more.

5.1.3 Output:

- When the tool is running it will keep printing on your screen the selected axioms as a step by step process for each failed theorem. It will also keep informing what exactly it's currently doing.

  For example:

  -If it's finding model between selected axiom list and failed conjecture, display would be:

```
=================================================================================
Finding models through Paradox (ATP system will be the one selected by user) ……
=================================================================================
```

  -If it's trying to prove a failed theorem with the selected list of axiom, display would be:

```
=================================================================================
Trying to prove with EP (ATP system will be the one selected by user)….
=================================================================================
```

- If the tool is stuck at some point for more than 5 – 10 minutes and has printed out anything (keeping in mind that you have not provided big time limits), then either the computer system has a too low configuration for the tool or it's serving a huge number of big processes simultaneously. Thus, if time is a main factor, then you will have to take care of this issue by yourself.
- The tool will also update `SelectedAxiomFile` file in folder `FilesUsedInCode`. For the purpose of analyzing all newly solved problems, the user can open `SelectedAxiomFile` from `FilesUsedInCode` sub-directory. This file will contain the following for each problem which was proved:

- Problem name.
- List of axioms used in proving that problem.
- Total number of axioms used in proving that problem.
- Total number of axioms actually listed in the include file for that problem.
- Time taken by the ATP system to prove that problem from refined axiom set.
- Time listed for this problem in which it failed previously.

5.2 Testing and Results

5.2.1 Problems used

For our purpose of testing the tool, we picked all FOF problems from SWC category of TPTP problem library [13] version 3.3.0. There were a total 422 problems. We then picked the corresponding solution files produced by the ATP system `EP-0.99` for all the 422 problems.

For the above mentioned problem domain, we received the following statistics (listed in Table 5.2.1) ,which made it a perfect test case for our tool as it contains a huge axiom set, consists of both broken, and failed problems, and the number of solved theorems is large enough to initiate *basic learning process*.

No.	STATISTICS - NAME	VALUE
1	Selected problem domain	SWC
2	Total theorems	422
3	Failed theorems	124

No.	STATISTICS - NAME	VALUE
4	Broken theorems	24
5	Solved theorems	274
6	Selected ATP system for solutions	EP---0.99
7	Number of axioms listed for the problem domain (axiom set)	95
8	CPU time limit for each theorem	600s

Table 5.2.1: ProblemStatistics_A

5.2.2 Hardware Configuration

All the test cases have been performed with the following computer system configuration.

Processor name	:	Intel(R) Core(TM) 2 CPU 6600 @ 2.40GHz
CPU MHz	:	2394.713
Cache size	:	4096 KB
Address size	:	36 bits physical, 48 bits virtual
RAM size	:	2059904 kB
Operating system	:	Fedora release 8
Hard disk size	:	235GB

5.2.3 Results

Generation of each result set may take hours of operation. As showed in Section 5.1.1, the user is provided freedom to choose options like ATP system time limit, selection of algorithm, relevance set limit etc. Thus the success of our system largely depends upon the combination of user inputs. Full testing of all possible parameters will take too long, therefore some successful test results are presented in this section. Many such combinations were applied to the system while in testing phase, therefore following test results are been generated by an experienced user. These are not necessarily an optimal solution, but they illustrate the capabilities of our system.

In the generation of testing results, the sole purpose was to prove the failed conjectures and not increase the CPU time limit. Justification of this motive is also shown in "Evaluating general purpose automated theorem proving systems" [27], which clearly shows that in TPTP world the ability to solve more problems is better determined by the theorem proving technique rather than by increasing the CPU time.

All the results generated here are not independent of each other, as after solving some failed conjectures and updating the problem domain, it is believed that some knowledge is been added, which might be useful for the remaining failed conjectures.

In all the result sets (Result Set [1-5]) listed below, column names of the tables depict following:

1) *Problem Name*	:	New solved problem name.

2) *Problems Rating & System of TPTP version*: Rating of problem and the version of `SystemOnTPTP` from which rating is taken.

3) *Number of Axioms which proved* : Total number of axioms in last refined axiom set for this problem.

4) *Number of Axioms actually listed* : Total number of axioms listed in the include file of the solved problem.

5) *CPU Time in last run* : CPU time taken by the ATP system, for proving the problem with the last refined axiom set.

6) *Previous CPU time* : CPU time, in which EP---0.99 gave TMO for the same problem.

<u>Result set 1</u>

This set has been formed by using following set of options, and Table 5.2.3(a) shows the output:

ATP system time limit	:	60s
Model finder time limit	:	30s
Relevance set limit	:	.40
Selected algorithm	:	RelevanceSet
Selected problem set	:	BrokenProblems
Selected ATP system	:	EP---0.99
Use of desperate axiom set	:	NO

No.	Problem Name	Problem Rating and SystemOnTPTP version	Number of Axioms which proved	Number of Axioms actually listed	CPU Time in last run	Previous CPU time
1	SWC021	0.84 v3.3.0	38	95	58.1s	481.1s
2	SWC031	0.84 v3.3.0	38	95	30.0s	422s
3	SWC055	1.00 v2.4.0	29	95	10.0s	376.1s
4	SWC056	1.00 v2.4.0	29	95	10.0s	395.9s
5	SWC077	0.84 v3.3.0	38	95	51.8s	348.1s
6	SWC078	0.95 v3.3.0	22	95	0.9s	417.7s
7	SWC093	0.79 v3.3.0	22	95	0.9s	438.7s
8	SWC119	0.95 v3.3.0	22	95	0.9s	421.7s
9	SWC228	0.79 v3.3.0	28	95	0.4s	365.9s
10	SWC237	0.84 v3.3.0	28	95	0.4s	364.5s
11	SWC245	0.84 v3.3.0	26	95	0.4s	421.4s
12	SWC250	0.79 v3.3.0	28	95	0.4s	369.9s
13	SWC341	1.00 v3.3.0	24	95	10.0s	402.2s
14	SWC363	0.79 v3.3.0	22	95	0.9s	417.1s
15	SWC388	0.95 v3.3.0	24	95	1.1s	451.9s
16	SWC390	1.00 v2.6.0	24	95	24.0s	383.4s
17	SWC412	0.79 v3.3.0	13	95	10.0s	461.4s

Table 5.2.3(a): ResultSet1

Result set 2

After Result Set 1, this set has been formed by using underlying set of options, and Table 5.2.3(b) shows the output:

ATP system time limit	:	180s
Model finder time limit	:	30s
Relevance set limit	:	.40
Selected algorithm	:	RelevanceSet
Selected problem set	:	BrokenProblems
Selected ATP system	:	EP---0.99
Use of *desperate axiom set*	:	NO

No.	**Problem Name**	**Problem Rating and SystemOnTPTP version**	**Number of Axioms which proved**	**Number of Axioms actually listed**	**CPU Time in last run**	**Previous CPU time**
1	SWC091	0.84 v3.3.0	42	95	108.2s	336.9s
2	SWC099	0.79 v3.3.0	41	95	104.2s	473.8s
3	SWC214	0.84 v3.3.0	42	95	115.7s	459.6s

Table 5.2.3(b): ResultSet2

Result set 3

After Result Set 1 and 2, this set has been formed by using underlying set of options, and Table 5.2.3(c) shows the output:

ATP system time limit	:	150s
Model finder time limit	:	30s
Relevance set limit	:	.40
Selected algorithm	:	RelevanceSet
Selected problem set	:	FailedProblems
Selected ATP system	:	EP---0.99
Use of *desperate axiom set*	:	NO

No.	**Problem Name**	**Problem Rating and SystemOnTPTP version**	**Number of Axioms which proved**	**Number of Axioms actually listed**	**CPU Time in last run**	**Previous CPU time**
1	SWC066+1	0.95 v3.3.0	41	95	128.2s	600s
2	SWC107+1	1.00 v2.4.0	33	95	107.1s	600s
3	SWC111+1	0.95 v3.3.0	41	95	125.6s	600s
4	SWC125+1	1.00 v2.6.0	41	95	124.2s	600s
5	SWC224+1	0.79 v3.3.0	23	95	0.2s	600s
6	SWC226+1	0.79 v3.3.0	25	95	0.1s	600s
7	SWC234+1	0.84 v3.3.0	25	95	26s	600s

No.	Problem Name	Problem Rating and SystemOnTPTP version	Number of Axioms which proved	Number of Axioms actually listed	CPU Time in last run	Previous CPU time
8	SWC310+1	0.95 v3.3.0	22	95	47.3s	600s
9	SWC316+1	0.74 v3.3.0	22	95	40.5s	600s
10	SWC343+1	0.89 v3.3.0	19	95	0.1s	600s
11	SWC393+1	0.84 v3.3.0	22	95	78.4s	600s
12	SWC394+1	0.95 v3.3.0	21	95	109.3	600s
13	SWC400+1	0.84 v3.3.0	18	95	5.6s	600s
14	SWC401+1	0.84 v3.3.0	23	95	54.4s	600s
15	SWC403+1	0.84 v3.3.0	23	95	54.7s	600s
16	SWC416+1	0.74 v3.3.0	16	95	0.1s	600s
17	SWC420+1	1.00 v3.3.0	16	95	0.1s	600s

Table 5.2.3(c): ResultSet3

Result set 4

After Result Set 1, 2 and 3, this set has been formed by using underlying set of options, and Table 5.2.3(d) shows the output:

ATP system time limit	:	150s
Model finder time limit	:	30s
Relevance set limit	:	.40

Selected algorithm	:	OneAxiomOneTime
Selected problem set	:	FailedProblems
Selected ATP system	:	EP---0.99
Use of *desperate axiom set*	:	NO

No.	**Problem Name**	**Problem Rating and SystemOnTPTP version**	**Number of Axioms which proved**	**Number of Axioms actually listed**	**CPU Time in last run**	**Previous CPU time**
1	SWC002+1	0.68 v3.3.0	1	95	0.1s	600s
2	SWC062+1	1.00 v2.4.0	15	95	0.2s	600s
3	SWC088+1	1.00 v2.4.0	15	95	0.2s	600s
4	SWC152+1	0.95 v3.3.0	10	95	1.9s	600s
5	SWC344+1	0.95 v3.3.0	3	95	0.1s	600s
6	SWC366+1	0.89 v3.3.0	14	95	0.1s	600s
7	SWC383+1	1.00 v3.3.0	24	95	103.7s	600s

Table 5.2.3(d): ResultSet4

Result set 5

After Result Set 1, 2, 3 and 4, this set has been formed by using underlying set of options, and Table 5.2.3(e) shows the output:

ATP system time limit	:	180s
Model finder time limit	:	30s

Relevance set limit	:	.40
Selected algorithm	:	RelevanceSet
Selected problem set	:	BrokenProblems
Selected ATP system	:	EP---1.00
Use of *desperate axiom set*	:	NO

No.	**Problem Name**	**Problem Rating and SystemOnTPTP version**	**Number of Axioms which proved**	**Number of Axioms actually listed**	**CPU Time in last run**	**Previous CPU time**
1	SWC117	0.89 v3.3.0	37	95	94.1s	542.8s
2	SWC123	0.79 v3.3.0	41	95	102.2s	393.6s
3	SWC342	1.00 v2.4.0	21	95	92.2s	391.6s
4	SWC374	0.95 v3.3.0	19	95	66.1s	393.3s

Table 5.2.3(e): ResultSet5

5.3 Analysis and Discussion

Analysis 1

As discussed before, broken problems have a higher probability of getting solved than the failed problems. This is true and can be seen from Result Set 1, Result Set 2 and Result Set 5, which together produce a 100 % result for the broken problems in the selected

SWC problem domain. Broken problems should always be tried first because they may increase the number of solved files in the problem domain, which in turn increases the probability of finding solutions for remaining problems.

Analysis 2

From the result sets, it was realized that five FOF problems from SWC domain (namely: `SWC062+1.p`, `SWC088+1.p`, `SWC107+1.p`, `SWC383+1.p`, `SWC420+1.p`), which have rating of 1.00 (meaning unsolved problems), are only solved by MLAR and not solved by any of the ATP systems before, thus proving that MLAR is capable enough of finding solutions for harder problems.

Analysis 3

Table 5.2.3(f) listed below depicts all problems that are solved by only one ATP system other than MLAR. Column 'Problem Name' in the table below shows the name of problems being solved. Column 'Previous ATP system' shows the ATP system which solved the corresponding problem. Column 'CPU time from previous ATP system' depicts time taken from corresponding ATP system to solve the problem. Column 'CPU time in last run from MLAR' shows CPU time taken to solve corresponding problem by one of the refined axiom sets.

S.No.	Problem Name	Previous ATP system	CPU time from previous ATP system	CPU time in last run from MLAR
1	SWC055	SiNE	73.1	10.0
2	SWC056	SiNE	71.9	10.0
3	SWC111	SiNE	125.6	125.6
4	SWC 125	SiNE	137.3	124.2
5	SWC310	SPASS	22.0	47.3
6	SWC341	SiNE	131.3	10.0
7	SWC342	SiNE	131.6	92.2
8	SWC344	SRASS	74.5	0.1
9	SWC390	SiNE	77.8	24.0

Table 5.2.3(f): Analysis3

Comparison with SRASS

Results were compared with SRASS [21] because it also works on the concept of semantic selection. Out of the 148 problems (124 failed problems and 24 broken problems) SRASS solved only 18 problems (`SWC099+1.p`, `SWC123+1.p`, `SWC149+1.p`, `SWC181+1.p`, `SWC224+1.p`, `SWC226+1.p`, `SWC228+1.p`, `SWC250+1.p`, `SWC264+1.p`, `SWC308+1.p`, `SWC316+1.p`, `SWC343+1.p`, `SWC344+1.p`, `SWC344+1.p`, `SWC366+1.p`, `SWC393+1.p`, `SWC409+1.p`, `SWC412+1.p`, `SWC416+1.p`) where as MLAR solved 48 problems (24 failed problems and 24 broken problems). All the problems solved by MLAR are showed in *Result Sets* above.

5.4 Output by axiom refining strategies:

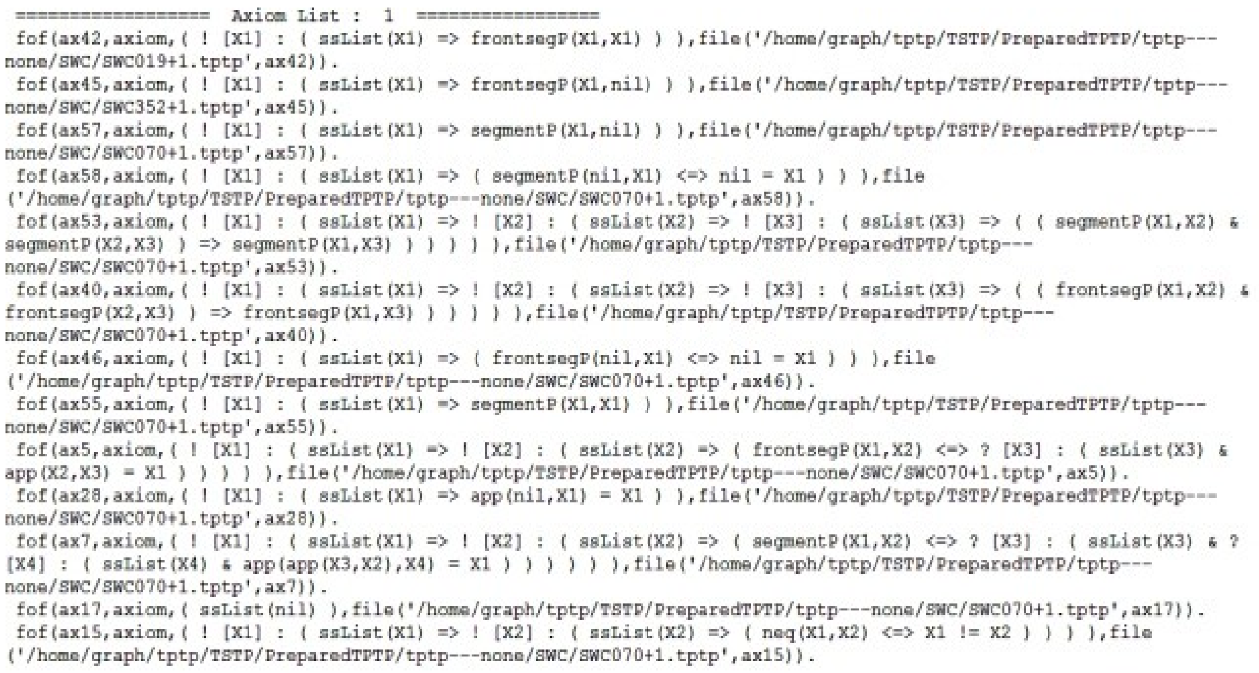
By observing the output from different axiom refining strategies, it was realized that they produce different refined axiom sets for a failed conjecture. As for the purpose of showing refined axiom set produced by all axiom refining strategies, we pick broken problem `SWC021+1.p` from SWC problem domain. Failed conjecture (c021) listed in the above problem is shown in Figure 3.3:

```
%------------------------------------------------------------
fof(c021,conjecture,
    ( ! [U] :
        ( ssList(U)
       => ! [V] :
            ( ssList(V)
           => ! [W] :
                ( ssList(W)
               => ! [X] :
                    ( ssList(X)
                   => ( V != X
                      | U != W
                      | ~ frontsegP(X,W)
                      | ~ strictorderedP(W)
                      | ? [Y] :
                          ( ssList(Y)
                          & neq(Y,nil)
                          & frontsegP(V,Y)
                          & frontsegP(U,Y) )
                      | ? [Z] :
                          ( ssList(Z)
                          & neq(W,Z)
                          & frontsegP(X,Z)
                          & segmentP(Z,W)
                          & strictorderedP(Z) )
                      | ( nil = V
                        & nil = U ) ) ) ) ) ) )).
 %------------------------------------------------------------
```

Figure 3.3: Broken problem 21

Output from SSCRS

Now, if we consider the failed conjecture (c021) from broken problem `SWC021+1.p`, then the first refined axiom list produced by SSCRS for making a proof attempt on c021 is shown in Figure 3.3.1(b):



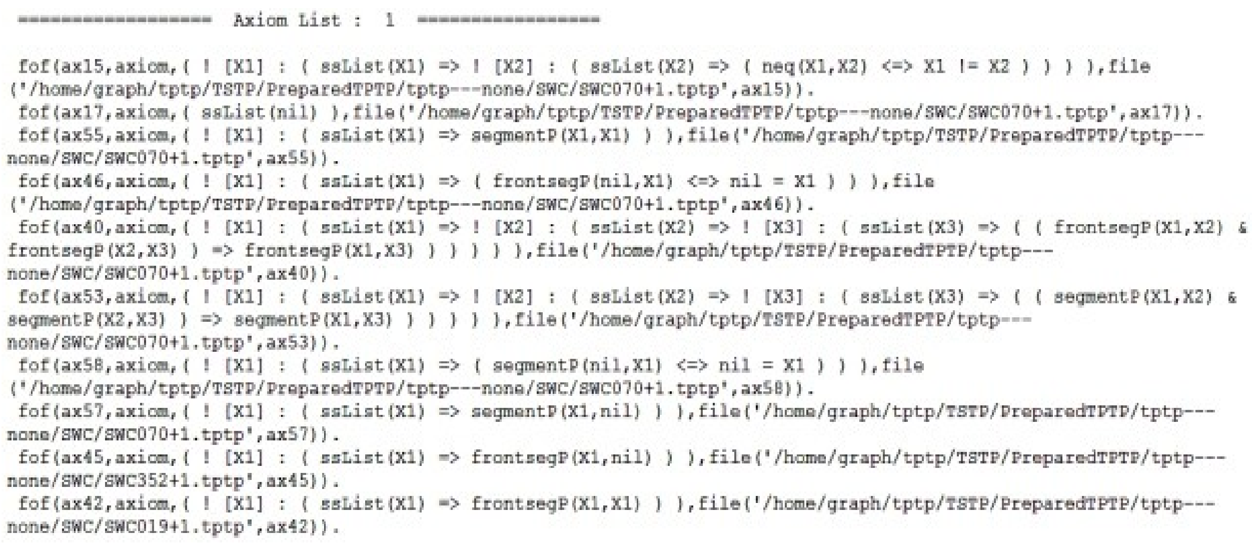
```
 ==================  Axiom List :  1  =================
 fof(ax42,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC019+1.tptp',ax42)).
 fof(ax45,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC352+1.tptp',ax45)).
 fof(ax57,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax57)).
 fof(ax58,axiom,( ! [X1] : ( ssList(X1) => ( segmentP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax58)).
 fof(ax53,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( segmentP(X1,X2) &
segmentP(X2,X3) ) => segmentP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax53)).
 fof(ax40,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( frontsegP(X1,X2) &
frontsegP(X2,X3) ) => frontsegP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax40)).
 fof(ax46,axiom,( ! [X1] : ( ssList(X1) => ( frontsegP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax46)).
 fof(ax55,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax55)).
 fof(ax5,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( frontsegP(X1,X2) <=> ? [X3] : ( ssList(X3) &
app(X2,X3) = X1 ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax5)).
 fof(ax28,axiom,( ! [X1] : ( ssList(X1) => app(nil,X1) = X1 ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax28)).
 fof(ax7,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( segmentP(X1,X2) <=> ? [X3] : ( ssList(X3) & ?
[X4] : ( ssList(X4) & app(app(X3,X2),X4) = X1 ) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax7)).
 fof(ax17,axiom,( ssList(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax17)).
 fof(ax15,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( neq(X1,X2) <=> X1 != X2 ) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax15)).
```

Figure: 3.3.1(b)- SSCRS axiom union list

From the Figure 3.3.1(b), we can clearly see that SSCRS selected 13 axioms for problem `SWC021+1.p` in its first refined axiom set.

Output from ARS

Next axiom refining strategy known as ARS, selected 10 axioms as the first generated axiom list for the purpose of making a proof attempt on c021, and is shown in Figure 3.3.2(b):



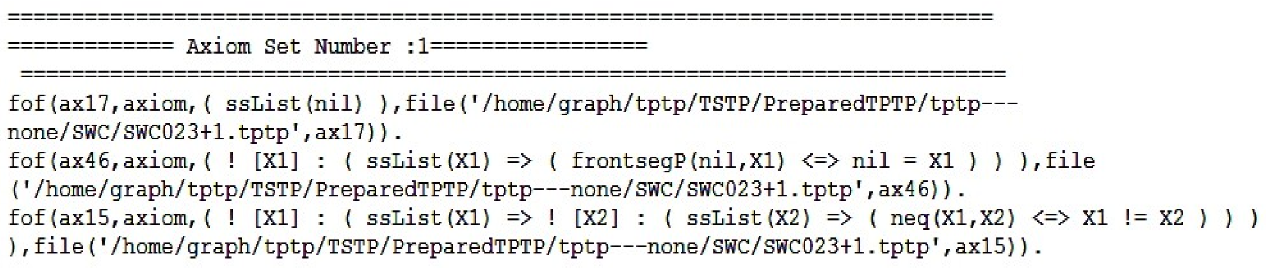
```
 ================== Axiom List :  1  =================

 fof(ax15,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( neq(X1,X2) <=> X1 != X2 ) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax15)).
 fof(ax17,axiom,( ssList(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax17)).
 fof(ax55,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax55)).
 fof(ax46,axiom,( ! [X1] : ( ssList(X1) => ( frontsegP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax46)).
 fof(ax40,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( frontsegP(X1,X2) &
frontsegP(X2,X3) ) => frontsegP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax40)).
 fof(ax53,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( segmentP(X1,X2) &
segmentP(X2,X3) ) => segmentP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax53)).
 fof(ax58,axiom,( ! [X1] : ( ssList(X1) => ( segmentP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax58)).
 fof(ax57,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax57)).
 fof(ax45,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC352+1.tptp',ax45)).
 fof(ax42,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC019+1.tptp',ax42)).
```

Figure: 3.3.2(b)- ARS axiom union list

<u>Output from SCAAR</u>

Now if we consider SCAAR, then for c021 first refined axiom set contains three axioms as shown in Figure3.3.3 (b):

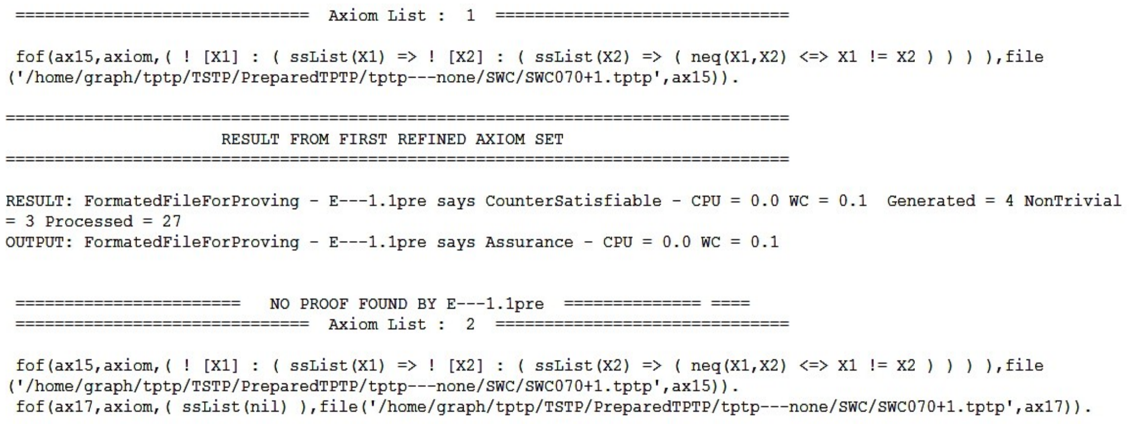


```
===============================================================================
============= Axiom Set Number :1=================
 ==============================================================================
fof(ax17,axiom,( ssList(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC023+1.tptp',ax17)).
fof(ax46,axiom,( ! [X1] : ( ssList(X1) => ( frontsegP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC023+1.tptp',ax46)).
fof(ax15,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( neq(X1,X2) <=> X1 != X2 ) ) )
),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC023+1.tptp',ax15)).
```

Figure: 3.3.3(b)- SCAAR axiom union list

<u>Output from OAOT</u>

Considering the last axiom refining strategy called OAOT, figure 3.3.4(b) shows the first refined axiom set, which contain only one axiom ax15. Output for first refined axiom set by `E---1.1pre`, is also shown. `SWC021+1.p` was not solved by first refined axiom set. Next refined axiom set (containing axioms ax15, and ax17) is then depicted in Figure 3.3.4(b). .



```
 ==============================  Axiom List :  1  ==============================

 fof(ax15,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( neq(X1,X2) <=> X1 != X2 ) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax15)).

================================================================================
                      RESULT FROM FIRST REFINED AXIOM SET
================================================================================

RESULT: FormatedFileForProving - E---1.1pre says CounterSatisfiable - CPU = 0.0 WC = 0.1  Generated = 4 NonTrivial
= 3 Processed = 27
OUTPUT: FormatedFileForProving - E---1.1pre says Assurance - CPU = 0.0 WC = 0.1


 =======================   NO PROOF FOUND BY E---1.1pre  ============== ====
 ==============================  Axiom List :  2  ==============================

 fof(ax15,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( neq(X1,X2) <=> X1 != X2 ) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax15)).
 fof(ax17,axiom,( ssList(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax17)).
```

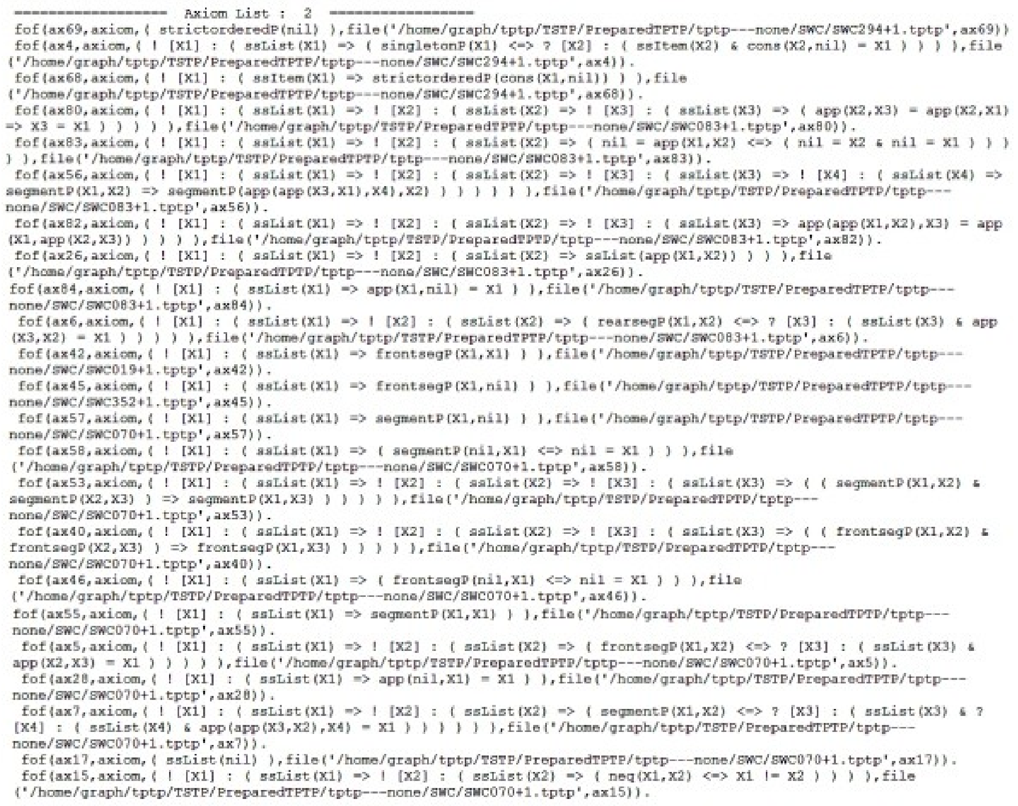
Figure: 3.3.4(b)- OAOT Axiom Union List

<u>Output by semantic selection</u>

Now by considering semantic selection approach, we can observe following:

When we try proving `SWC021+1.p`, figure 4.2(a) below shows the refined axiom list from second relevance set produced by SSCRS strategy. There exist 23 different axioms namely:

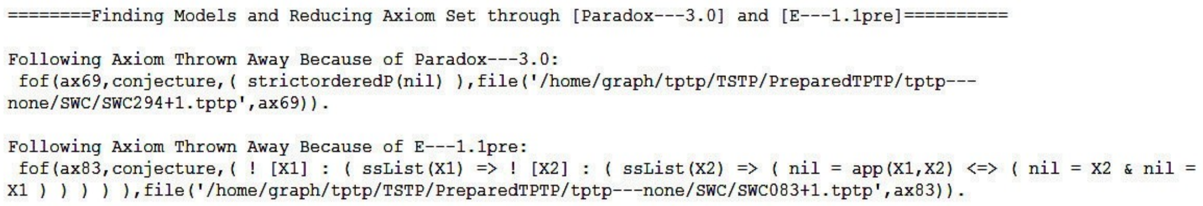
`ax69, ax4, ax68, ax80, ax83, ax56, ax82, ax26, ax84, ax6, ax42, ax45, ax57, ax58, ax53, ax40, ax46, ax55, ax5, ax28, ax7, ax17 and ax15`



```
 ==================  Axiom List :  2  =================
 fof(ax69,axiom,( strictorderedP(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC294+1.tptp',ax69))
 fof(ax4,axiom,( ! [X1] : ( ssList(X1) => ( singletonP(X1) <=> ? [X2] : ( ssItem(X2) & cons(X2,nil) = X1 ) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC294+1.tptp',ax4)).
 fof(ax68,axiom,( ! [X1] : ( ssItem(X1) => strictorderedP(cons(X1,nil)) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC294+1.tptp',ax68)).
 fof(ax80,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( app(X2,X3) = app(X2,X1)
=> X3 = X1 ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax80)).
 fof(ax83,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( nil = app(X1,X2) <=> ( nil = X2 & nil = X1 ) ) )
) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax83)).
 fof(ax56,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ! [X4] : ( ssList(X4) =>
segmentP(X1,X2) => segmentP(app(app(X3,X1),X4),X2) ) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC083+1.tptp',ax56)).
 fof(ax82,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => app(app(X1,X2),X3) = app
(X1,app(X2,X3)) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax82)).
 fof(ax26,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ssList(app(X1,X2)) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax26)).
fof(ax84,axiom,( ! [X1] : ( ssList(X1) => app(X1,nil) = X1 ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC083+1.tptp',ax84)).
 fof(ax6,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( rearsegP(X1,X2) <=> ? [X3] : ( ssList(X3) & app
(X3,X2) = X1 ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax6)).
 fof(ax42,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC019+1.tptp',ax42)).
 fof(ax45,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC352+1.tptp',ax45)).
 fof(ax57,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax57)).
 fof(ax58,axiom,( ! [X1] : ( ssList(X1) => ( segmentP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax58)).
 fof(ax53,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( segmentP(X1,X2) &
segmentP(X2,X3) ) => segmentP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax53)).
 fof(ax40,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( frontsegP(X1,X2) &
frontsegP(X2,X3) ) => frontsegP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax40)).
 fof(ax46,axiom,( ! [X1] : ( ssList(X1) => ( frontsegP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax46)).
 fof(ax55,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax55)).
 fof(ax5,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( frontsegP(X1,X2) <=> ? [X3] : ( ssList(X3) &
app(X2,X3) = X1 ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax5)).
 fof(ax28,axiom,( ! [X1] : ( ssList(X1) => app(nil,X1) = X1 ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax28)).
 fof(ax7,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( segmentP(X1,X2) <=> ? [X3] : ( ssList(X3) & ?
[X4] : ( ssList(X4) & app(app(X3,X2),X4) = X1 ) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax7)).
 fof(ax17,axiom,( ssList(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax17)).
 fof(ax15,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( neq(X1,X2) <=> X1 != X2 ) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax15)).
```

Figure 4.2(a): BP1 - Second Relevance Set

As soon as the syntactic refinement approach is completed, the control shifts to the semantic refinement approach which discards following three axioms from the list: `ax69`, and `ax83`. Figure 4.2 (b) shows the output



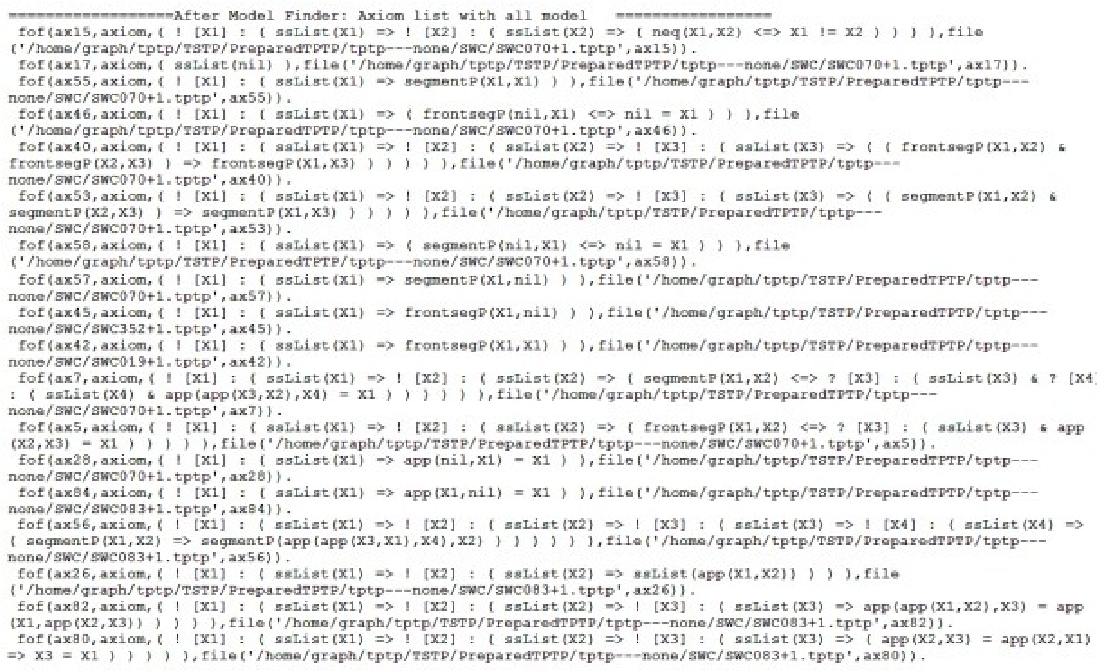
```
========Finding Models and Reducing Axiom Set through [Paradox---3.0] and [E---1.1pre]=========

Following Axiom Thrown Away Because of Paradox---3.0:
 fof(ax69,conjecture,( strictorderedP(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC294+1.tptp',ax69)).

Following Axiom Thrown Away Because of E---1.1pre:
 fof(ax83,conjecture,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( nil = app(X1,X2) <=> ( nil = X2 & nil =
X1 ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax83)).
```

Figure 4.2(b): BP1-Discarded axioms

After discarding the above listed axioms, semantic approach generates the perfect set for `SWC002+1.p`. It is shown in Figure 4.2(c):



```
==================After Model Finder: Axiom list with all model   =================
 fof(ax15,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( neq(X1,X2) <=> X1 != X2 ) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax15)).
 fof(ax17,axiom,( ssList(nil) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax17)).
 fof(ax55,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax55)).
 fof(ax46,axiom,( ! [X1] : ( ssList(X1) => ( frontsegP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax46)).
 fof(ax40,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( frontsegP(X1,X2) &
frontsegP(X2,X3) ) => frontsegP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax40)).
 fof(ax53,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( ( segmentP(X1,X2) &
segmentP(X2,X3) ) => segmentP(X1,X3) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax53)).
 fof(ax58,axiom,( ! [X1] : ( ssList(X1) => ( segmentP(nil,X1) <=> nil = X1 ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax58)).
 fof(ax57,axiom,( ! [X1] : ( ssList(X1) => segmentP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax57)).
 fof(ax45,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,nil) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC352+1.tptp',ax45)).
 fof(ax42,axiom,( ! [X1] : ( ssList(X1) => frontsegP(X1,X1) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC019+1.tptp',ax42)).
 fof(ax7,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( segmentP(X1,X2) <=> ? [X3] : ( ssList(X3) & ? [X4]
: ( ssList(X4) & app(app(X3,X2),X4) = X1 ) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax7)).
 fof(ax5,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ( frontsegP(X1,X2) <=> ? [X3] : ( ssList(X3) & app
(X2,X3) = X1 ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC070+1.tptp',ax5)).
 fof(ax28,axiom,( ! [X1] : ( ssList(X1) => app(nil,X1) = X1 ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC070+1.tptp',ax28)).
 fof(ax84,axiom,( ! [X1] : ( ssList(X1) => app(X1,nil) = X1 ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC083+1.tptp',ax84)).
 fof(ax56,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ! [X4] : ( ssList(X4) =>
( segmentP(X1,X2) => segmentP(app(app(X3,X1),X4),X2) ) ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---
none/SWC/SWC083+1.tptp',ax56)).
 fof(ax26,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ssList(app(X1,X2)) ) ) ),file
('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax26)).
 fof(ax82,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => app(app(X1,X2),X3) = app
(X1,app(X2,X3)) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax82)).
 fof(ax80,axiom,( ! [X1] : ( ssList(X1) => ! [X2] : ( ssList(X2) => ! [X3] : ( ssList(X3) => ( app(X2,X3) = app(X2,X1)
=> X3 = X1 ) ) ) ) ),file('/home/graph/tptp/TSTP/PreparedTPTP/tptp---none/SWC/SWC083+1.tptp',ax80)).
```

Figure 4.2(c): BP1- Semantic Perfect List

If we observe from Figure 4.2(c), we will realize that the list not only excludes two discarded axioms but there are some other axioms also missing from the list namely `ax4`, `ax6`, and `ax68`. These axioms are missing because MF gave TMO as output for them. Therefore a proof attempt is made on `SWC002+1.p` with the above mentioned perfect set. If in case proof is not found then we include `ax4, ax6` and `ax68` to the perfect set and call it as confused set. From this confused set, second proof attempt is made on the problem. If no proof is found, and relevance set limit is not reached then next relevance set is picked and the process for making proof attempt continues till the relevance set limit is reached. As soon as relevance set limit is reached, next failed problem is picked from the list. If there are no more failed problems left to be attempt, then try singing the song "Quit playing games" and terminate the complete process. .

Chapter 6

6.1 Conclusion

This research is applicable to the problem of large theories in ATP world. It is helpful in finding solutions for the failed problems existing in TPTP world. This research tries to find a solution for failed problems by facilitating a learning process for each failed problem with the help of solved theorems. It then refines the axiom set by combining syntactic and semantic approaches of axiom selection. The research looks promising as it solved 48 new FOF problems in the SWC domain. The research gives more importance in finding a solution for a failed problem rather than considering CPU times.

6.2 Future Work

- More intelligence should be added like for the purpose of selecting time limits, relevance limit and ATP systems based upon the analysis of the original problem domain.
- Careful selection of algorithm is very important because some of the problems in given domain may need one axiom, some other may need more than 40 axioms etc. Therefore, intelligence for selecting the algorithm based upon the history and nature of problem file (conjecture) is very important.
- Full analysis of the system should be performed by considering various different parameter combinations.

References

[1] W. Reif and G. Schellhorn. Theorem proving in large theories. In Wolgang bible and Peter H. Schmitt, editors, Automated Deduction A Basis for- Application,volume III. Application, pages 225-240 Kluwer Academic Publishers, 1998.

[2] W. Reif. The KIV-approach to Software Verification. In M. Broy and S. Jahnichen, editors, KORSO:Methods, Lnaguages, and Tools for the Construction of Correct Software – Final Report, LNCS 1009. Springer, Berlin, 1995.

[3] W. Reif, G. Schellhorn, and K. Stenzel. Interactive Correctness Proofs for Software Modules Using KIV. In COMPASS'95 – Tenth Annual Conference on Computer Assurance (Gaithersburg (MD). USA). IEEE press,1995.

[4] W. Reif, G. Schellhorn, and K. Stenzel. Proving System Correctness with KIV 3.0. In 14th International Conference on Automated Deduction. Proceedings. Townsville, Australia, Springer LNCS 1249, pages 69-72, 1997.

[5] A. Turing. (October 1950), Computing Machinery and Intelligence, Mind LIX (236): 433–460, doi:10.1093/mind/LIX.236.433, ISSN 0026-4423.

[6] M. A. Arbib (Ed.) (1995). The Handbook of Brain Theory and Neural Networks.

[7] J. Ignizio (1991). Introduction to Expert Systems. ISBN 0-07-909785-5.

[8] Prophet: http://www.cs.miami.edu/~tptp/Seminars/Relevance/Summary.html

[9] C. Lengauer . A View of Automated Proof Checking and Proving. New Jersey, J. C. Baltzer AG, Science Publishers, 1988.

[10 J. Urban. MaLARea: a Metasystem for Automated Reasoning in Large Theories. ESARLT 2007. In proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, Bremen, Germany, 17th July 2007. Vol-257 (2007).

[11] G. Sutcliffe, A. Dvorský. Proving Harder Theorems by Axiom Reduction. FLAIRS Conference 2003: 108-113

[12] T. Nipkow, L. C. Paulson, M. Wenzel: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. Springer 2002

[13] G. Sutcliffe and C. B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning, 21(2):177-203, 1998.

[14] Otter - ATP system. http://www-unix.mcs.anl.gov/AR/otter/.

[15] S. Schulz(2002): E - A Brainiac Theorem Prover. Journal of AI Communications 15(2/3):111-126, 2002.

[16] R. S. Boyer, M. Kaufmann and J. S. Moore. The Boyer-Moore Theorem Prover and Its Interactive Enhancement. Computers and Mathematics with Applications, 29(2): 27-62, 1995.

[17] CYC –Higer order logic theorem prover. http://www.cyc.com/cyc

[18] F. M. Suchanek, G. Kasneci, G. Weikum. YAGO: A Core of Semantic Knowledge. In proceedings of 16th annual world wide web conference, pages 697-706, 2008.

[19] J. Meng and L. Paulson, Lightweight relevance filtering for machine-generated resolution problems, in: Proceedings of the FLoC'06 Workshop on Empirically Successful Computerized Reasoning, 3rd International Joint Conference on Automated Reasoning, G. Sutcliffe, R. Schmidt and S. Schulz, eds, volume 192 of CEUR Workshop Proceedings, 53-69, 2006.

[20] D. Plaisted and A. Yahya, A relevance restriction strategy for automated deduction, Artificial Intelligence 144 (2003) 59-93.

[21] G. Sutcliffe, Y. Puzis. SRASS – a Semantic Relevance Axiom Selection System. In F. Pfenning, editor, CADE 2007, Lecture Notes in Artificial Intelligence, pages 295-310. Springer, 2007.

[22] C. Benzmuller, L. Paulson, F. Theiss, and A. Fietzke. LEO-II – A Cooperative Automatic Theorem Prover for Higher-Order Logic. In Proceedings of 4th International Joint Conference on Automated Reasoning, volume 5195 of Lecture Notes in Artificial Intelligence, 162-170, 2008.

[23] G. Sutcliffe. SortByUsefulInfoField tool in TPTP world.

[24] G. Sutcliffe. ProofSummary tool in TPTP world.

[25] S. Schulz. E: A Brainiac Theorem Prover. AI Communications, 15(2-3):111-126, 2002.

[26] J. Hurd. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In M. Archer, B. Di Vito, and C. Munoz, editors, Proceedings of the 1st International

Workshop on Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003 212448 in NASA Technical Reports, pages 56-68, 2003.

[27] G. Sutcliffe, C.B Suttner. Evaluating General Purpose Automated Theorem Proving Systems. Journal of Artificial Intelligence, 131(1-2) :39-54,2001.

[28] P.H. Winston. Learning and Reasoning by Analogy: The Details. Cambridge, MIT Artificial Intelligence Lab. Memo, 1979.

[29] A. Feeney, E. Heit. Inductive reasoning: experimental, developmental, and computational approaches. Cambridge, Cambridge University Press, 2007.

[30] J.R. Josephson, S.G. Josephson. Abductive Inference: Computation, Philosophy, Technology. Cambridge, Cambridge University Press, 1995.