Department of Computer Science
Duale Hochschule Baden-Wurttemberg Stuttgart

Mission impossible: Disproving failed conjectures

Bachelor Thesis

Author:	Heba Aamer Anwar Mohamed
Supervisor:	Professor Dr. Stephan Schulz
Submission Date:	XX July, 2015

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Heba Aamer Anwar Mohamed
XX July, 2015

Acknowledgments

First, I have to thank my supervisor Professor Dr. Stephan Schulz for his continuous support and understanding.
Second, I have to thank my whole family for their support and encouraging to achieve this stage of my education.
Third, Many thanks should be said to my friends the ones who travelled with me and the others who were in Egypt for their help, encouraging.

Abstract

In the past few decades the field of automated theorem proving (ATP) has been flourishing and improving a lot by the enormous amount of research devoted to it. That interest came from its importance as well as its various uses in different fields such as mathematical reasoning.

ATP comes along with another process which is model generation/computation/construction from a certain (counter) satisfiable specification/problem. Model generation has usages that ATP alone wont have the effect that it has with it, and that could be noticed in Software/Hardware verification, debugging various systems.

Here in this project we added a model generation technique for a subclass in First Order Logic named Effectively Propositional Calculus in an existing theorem prover "E" where we transform the axioms of the specification into a certain form called range restricted form, and then after reaching saturation, we apply Bachmair and Ganzinger model construction technique to get the model.

Automated theorem proving has applications in mathematics, verification, commonsense reasoning, and many other domains. It can demonstrate the compliance of a system with certain requirements. However, it is often just as important to show that a desired property is not met. This can be done by constructing a counter-model, or, in simpler words, a counter-example. In this talk we describe the implementation of techniques that enable the theorem prover E to find such counter-examples for effectively propositional proof problems, and to give an explicit counter-models to the user.

Keywords: Automated Theorem Proving, Model Construction, Effectively propositional logic, Range Restricting Transformations, Theorem Prover E, Bachmair and Ganzinger Model Construction Technique.

Contents

Chapter 1

Introduction

Constructing Models has a huge importance in many fields specially in debugging tasks. It helps in modelling and highlighting the existence of bugs. And because of this it is used extensively in the field of Software and Hardware verification, also in analyzing and verifying Theorems.

So a concrete example to show its importance is the verification of Timsort, which was explained in details here [1]. Timsort is a hybrid sorting algorithm that was developed in 2002. It combines merge sort and insertion sort. And it was developed in the beginning to be used in python, but later on it was added to java. And today in Open JDK, Sun's JDK, and Android SDK it is the default sorting algorithm since it showed a great performance on real data. That resulted in using it in billions of devices because of the popularity of those platforms. In 2015, a formal verification for Timsort was tried to be done by a team using KeY, a verification tool for java programs that could be found here `http://www.key-project.org/`. And their analysis showed that the TimSort algorithm was broken and they found a bug and they corrected it, then after that they were able to formally verify the correction of the algorithm. And all the happened by the help of KeY. So it really beneficial to have models.

Having that importance in mind, we needed in this project to have an explicit model given to user when running problems on the theorem prover E, specifically for a sub-class of FOL named Effectively propositional calculus (EPR). And in order to achieve that goal, Transformations have been applied to the problem specification to transform it to a certain form, namely range restricted form. Afterwards, Bachmair and Ganzinger Model Construction Technique is applied to extract the explicit model from the saturated set of the problem specification.

So a discussion of the work done will be given in the following order. We will have a background on the topic in chapter 2. Then in chapter 3 we will discuss the methodology followed and the implementation. Afterwards in chapter 4 we will explain the procedures followed to test the accuracy and efficiency of the implemented techniques. Moreover, A discussion of the results and related works will be found in chapter 5. Then a conclusion will be given in chapter 6. And last but not least a discussion for related future work will be in chapter 7.

Chapter 2

Background

This chapter is devoted for introducing and familiarizing the reader to the theoretical concepts behind this project, and give the reader a background on the theorem prover E as well. So it will include the following sections:

- A background on FOL in section
- A background on EPR in section
- A background on ATP in section
- A background on E in section
- A background on Problem set in section

2.1 First order logic (FOL)

FOL, which is also known as first order predicate logic, is an expressive logic that allows us to formulate and encode most of our spoken language sentences in a defined way such that it could be further handled with rules such as simplification, inference rules. That allows automating the reasoning for FOL problems.

In order to have a general background on the topics of FOL, we will devote this section to that mission. So we are going to discuss the following points:

- Summary over FOL and Example on using it
- Decidability of FOL
- Herbrand Universe

2.1.1 FOL overview and Example on it

Overview on FOL

Quantifiers, variables, and functions are what signifies FOL over propositional logic. Being able to formulate "some", "all" is what added a lot to the expressiveness of FOL. So a review on the syntax and the structure of FOL if needed will be found in the appendix here B.

Example on FOL

A very famous example on FOL is the following:

Listing 2.1: Example on FOL

```
We want to formulate the following three sentences in FOL:

  (1) All Humans are mortals.
  (2) Socrates is a human.
  (3) Therefore, Socrates is mortal.

So in FOL syntax they are:

  (1) ∀X(Human(X) → Mortal(X))
  (2) Human(socrates)
  (3) Mortal(socrates)

Where Human and Mortal are predicates here since they represent
    property over the elements of the domain. While socrates is a
    constant or in other words a function symbol with arity 0.
```

2.1.2 Decidability of FOL

The Problem of proving validity, or in other words unsatisfiability, of a formula in First order logic (FOL) had a lot of attention and experiments in the beginning. A lot of trials were made to prove that it is decidable as mentioned in [5]. But those trials were not successful. Later on, in the same year both [31, 6] proved that this problem in general is un-decidable.

Some algorithms were developed such that if the formula is unsatisfiable it will give a refutation, or proof in simpler words, however if it was not unsatisfiable then the algorithm may halt/terminate and give the correct result and may not halt/terminate, so this problem is semi-decidable. An example for such procedure is resolution, with some refinements, in which its basic idea was first developed in [22]. So First order logic (FOL) is semi-decidable logic.

2.1.3 Universe of FOL

2.2 Automated theorem proving (ATP)

2.3 The theorem prover E

E is a saturation-based theorem prover that is concerned with full FOL with equality. It is known to be a fast one because of the unique and various heuristics implemented in it. The current state of E, that it could prove the un-satisfiability of set of axioms with the negation of the conjecture(s) by finding the empty clause, or returning the saturated set if the empty clause was not found and no more new clauses can be inferenced/simplified.

2.3.1 The main proof procedure

This part is devoted from giving a brief on the main proof saturation procedure.
Search state: U P
U contains unprocessed clauses, P contains processed clauses.
Initially, all clauses are in U , P is empty.
The given clause is denoted by g.

```
while U =
g = delete best(U )
g = simplify(g, P )
if g ==
SUCCESS, Proof found
if g is not subsumed by any clause in P (or otherwise redundant w.r.t. P )
P = P  c  P — c subsumed by (or otherwise redundant w.r.t.) g
T = c  P — c can be simplified with g
P = (P  T )  g
T = T  generate(g, P )
foreach c  T
c = cheap simplify(c, P )
if c is not trivial
U = U  c
SUCCESS, original U is satisfiable
```

Remarks: delete best(U) finds and extracts the clause with the best heuristic evaluation (see 3.3) from U . generate(g, P) performs all generating inferences using g as one premise, and clauses from P as additional premises. It uses inference rules (SP) or (SSP), (SN) or (SSN), (ER) and (EF).
simplify(c, S) applies all simplification inferences in which the main (simplified) premise is c and all the other premises are clauses from S. This typically includes full rewriting, (CD) and (CLC). cheap simplify(c, S) works similarly, but only ap-

plies inference rules with a particularly low cost implementation, usually including rewriting with orientable units, but not (CLC). The exact set of contraction rules used is configurable in either case.
Fig. 2. Saturation procedure of E

2.3.2 Latest results

The latest results showed performance approaching 70% over all the CNF and FOF problems and this is according to [24]

2.4 Problem Set

TPTP problems are first order problems, that are considered benchmarks to measure the performance of the different automated theorem provers. A description for TPTP could be found in [28], and the problem set could be downloaded from http://www.cs.miami.edu/~tptp/. TPTP problem set consists of different categories of problems, or divisions in other words, such as : PUZ, NLP, GRP. Those categories depends on the problems scientific meaning, e.g., NLP represents Natural language processing problems.

TPTP has its own language; TPTP for problems and TSTP for solutions. TPTP language could be written in two formats. The first of them is FOF, which is first order form. While the second form is CNF, which is clause normal form. Having a unified language for problems and solutions had a great impact on the field of ATP. And this allows the integration of different ATP systems ,i.e., Automated theorem provers, since they have a specific language that could communicate in.

An example for a problem, i.e., GRP004-1 problem, from the TPTP problem set is given below in CNF format:

Listing 2.2: GRP004-1.p problem

```
%------------------------------------------------------------------------------
% File     : GRP004-1 : TPTP v6.1.0. Released v1.0.0.
% Domain   : Group Theory
% Problem  : Left inverse and identity => Right inverse exists
% Version  : [Cha70] axioms : Incomplete.
% English  : In a group with left inverses and left identity every element
%            has a right inverse.

% Refs     : [Luc68] Luckham (1968), Some Tree-paring Strategies for Theore
%          : [Cha70] Chang (1970), The Unit Proof and the Input Proof in Th
%          : [CL73]  Chang & Lee (1973), Symbolic Logic and Mechanical Theo
% Source   : [Cha70]
% Names    : Example 3 [Luc68]
%          : Example 4 [Cha70]
%          : Example 4 [CL73]
%          : EX4 [SPRFN]
% Status     : Unsatisfiable
% Rating     : 0.00 v5.4.0, 0.11 v5.3.0, 0.10 v5.2.0, 0.00 v2.1.0, 0.00 v2
    .0.0
% Syntax     : Number of clauses       :    5 (   0 non-Horn;   3 unit;   3 RR)
%              Number of atoms         :   11 (   0 equality)
%              Maximal clause size     :    4 (   2 average)
%              Number of predicates    :    1 (   0 propositional; 3-3 arity)
%              Number of functors      :    3 (   2 constant; 0-1 arity)
%              Number of variables     :   15 (   1 singleton)
%              Maximal term depth      :    2 (   1 average)
% SPC        : CNF_UNS_RFO_NEQ_HRN

% Comments : [Luc68] is actually the right to left version.
%------------------------------------------------------------------------------
cnf(left_inverse,axiom,
    ( product(inverse(X),X,identity) )).
cnf(left_identity,axiom,
    ( product(identity,X,X) )).
cnf(associativity1,axiom,
    ( ~ product(X,Y,U)
    | ~ product(Y,Z,V)
    | ~ product(U,Z,W)
    | product(X,V,W) )).
cnf(associativity2,axiom,
    ( ~ product(X,Y,U)
    | ~ product(Y,Z,V)
    | ~ product(X,V,W)
    | product(U,Z,W) )).
cnf(prove_there_is_a_right_inverse,negated_conjecture,
    ( ~ product(a,X,identity) )).
%------------------------------------------------------------------------------
```

Chapter 3

Methodology and Implementation

Here in this chapter we will discuss the methods applied to extract an explicit model from an EPR problem specification. And how they are implemented using E's data structures and configurations.

In order to achieve our goal, we had to transform the problems' clauses, which are the axioms of the specification and the negated conjectures(if found), to clauses in range restricted form through a simplified form of range restricted transformations. The original range restricted transformations and there simplified version will be discussed in section 3.1.

3.1 Transformations

The methods applied in the project to extract the models follows the transformations discussed in [3]. A brief on the original transformations will be given in 3.1.1.

Moreover, as mentioned before that this project is concerned with a sub-class of FOL which is EPR some simplifications to the transformations were made as the removed steps will have no meaning in the context of EPR problems. Those simplifications will be discussed in 3.1.2.

3.1.1 Original transformations

The original procedures discussed in [3] generally work for all sub-classes of FOL. Those procedures should be applied to a given set of axioms in a specific form called implication form, sometimes it is called a sequent as in here —ref—, which is explained here – , and here – to know how to transform to that form. Moreover, those transformations are guranteed to terminate for any given problem set, which is a gain for us since some of the EPR problems were not terminating in the original configurations of E.

What are the Transformations

The Transformations are series of procedures, mainly about changing the clauses to certain form named range restricted form. Since the transformations deal with clauses in implication form, then we could define **Clauses in range restricted form** to be clauses in which all the variables that appear in the succeedent must exist in the anticedent as well.

An example for a range restricted clause is found below:

Listing 3.1: Range Restricted Clause Example

$$\forall X \forall Y \, (P(X) \wedge Q(Y) \longrightarrow R(X,Y)) \, .$$

where the **antecedent** here is $P(X) \wedge Q(Y)$;
whereas the **succeedent** is $R(X,Y)$.

For a reason why this range restricted form will help would be ??

How do the transformations work

Transformations add a domain predicate to the specification that will help in finding the Model by saying what are the elements of the domain, or the elements of the universe in other words.

Moreover, There are three types of procedures in the transformations:

- **Range restricting transformations**
  are the first two transformations, and they are the most important type of them. All the rest were added to enhance and improve the range restricting ones. They are responsible for transforming the input clauses to the range restricted form and enumerating the universe/domain of the problem in a way or another. Only one of the two transformations should applied, since they perform the same functionality but in different ways. The two transformations are:
  - Classical Range Restricting Transformation (CRR): it enumerates the Herbrand Universe in a naive simple way.
  - Range Restricting Transformation (RR): it was introduced to improve the naive implementation of the CRR. So it only adds elements to the domain only when it is needed.

- **Shifting transformations**
  are the second two transformations. They are optional to be used. They complement one another not replace each other. They were introduced mainly to prevent the non-termination of the transformations and to prevent generating and redundant and unpleasant clauses from the steps in RR as well. And the two shifting transformations are:
  - Basic Shifting Transformation (BS)
  - Partial Flattening Transformation (PF)
- **Blocking Transformation (BL)**
  is the last transformation and it is optional as well. And It was introduced to detect periodicity that may occur because of function terms.

Their order of application is:

1. One of the two range restricting CRR or RR
2. The Partial Flattening PF
3. The Basic Shifting BS
4. The Blocking BL

Where the output of a lower number transformation is the input to the higher one. A Flow Chart that summarizes what was explained will be found in Figure 3.1.

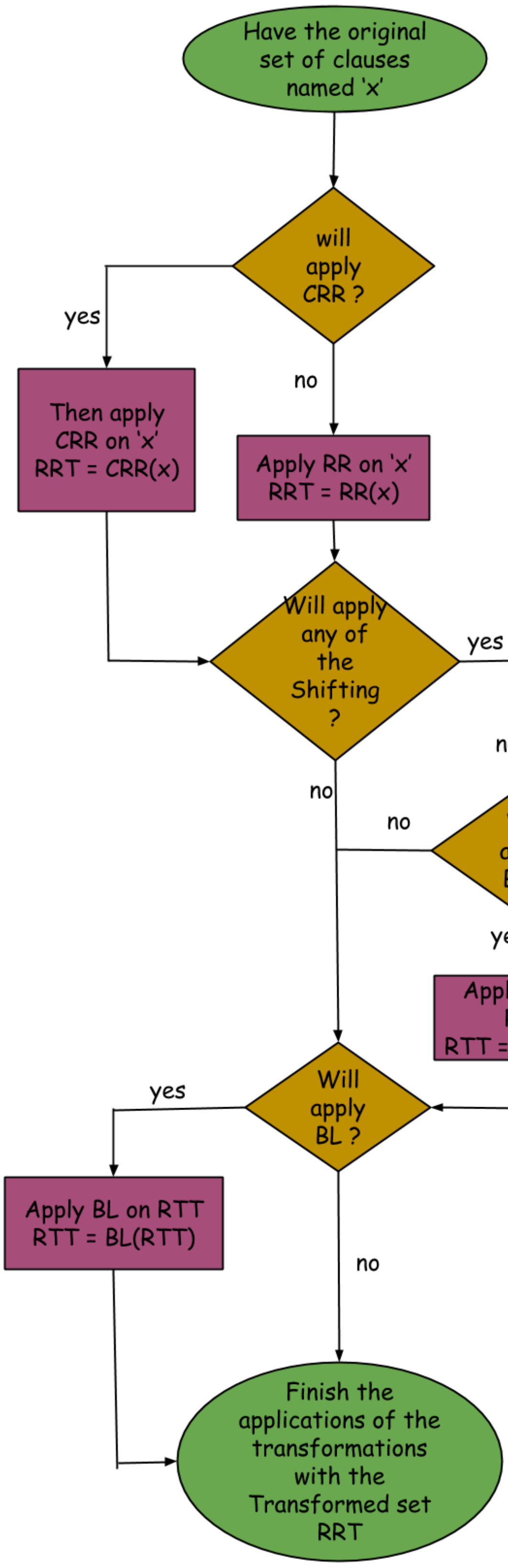

Have the original set of clauses named 'x'
will apply CRR ?
yes
no
Then apply CRR on 'x' RRT = CRR(x)
Apply RR on 'x' RRT = RR(x)
Will apply any of the Shifting ?
yes
Will apply PF ?
yes
Apply PF on RTT RTT = PF(RTT)
no
no
no
Will apply BS ?
yes
Apply BS on RTT RTT = BS(RTT)
yes
Will apply BL ?
Apply BL on RTT RTT = BL(RTT)
no
Finish the applications of the transformations with the Transformed set RRT

Figure 3.1: Flow of the original transformations

3.1.2 Simplified transformations

This subsection is concerned about discussing the simplifications that were added to the original transformations, that was explained in subsection 3.1.1. So this subsection will cover the following points:

- The reason for doing such simplifications.
- What are the Simplifications.
- The resulted simplified transformations.
- Examples for the Output of the transformations, or let's name it Intermediate Output.
- Discussion on the Output of the prover.

The Reason for doing the simplifications

As mentioned before in subsection 3.1.1, the original transformations is generic for all sub-classes of FOL. So naturally it contains many steps that deals with proper function symbols. However, what concerns us in this project is only the EPR sub-class of problems. And by keeping in mind the definition of EPR as mentioned here in **??**, and by knowing that there is no existence of proper function symbols, and that there won't be even after the skolemization step in any EPR problem, So implementing the original transformations, as they were, didn't seem logical as it won't be used in any of the problems, and it would take much more time to implement it practically. So those were the motives for having such simplifications.

What are the steps of the simplifications

The simplifications made were very simple, and they were applied to all procedures of the original transformations. The steps of the simplifications are:

1. Every step that only deals with proper function symbols is removed since they are not existing in EPR. Ex.: some steps in CRR and RR that loops on all the proper function symbols in the given problem.
2. Any step or procedure that was introduced because of the existence of proper function symbols in problems were removed as well. Ex.: BS, PF, and BL.

The resulted simplified transformations

After applying the simplifications to the procedures of the transformations, the following steps were the only needed:

- Some steps in CRR.
- Some steps in RR.

For a chart representing the original CRR and what is removed from it, you can view Figure 3.2, while Figure 3.3 for RR. And for a detailed explanation for the steps, you can check [3].

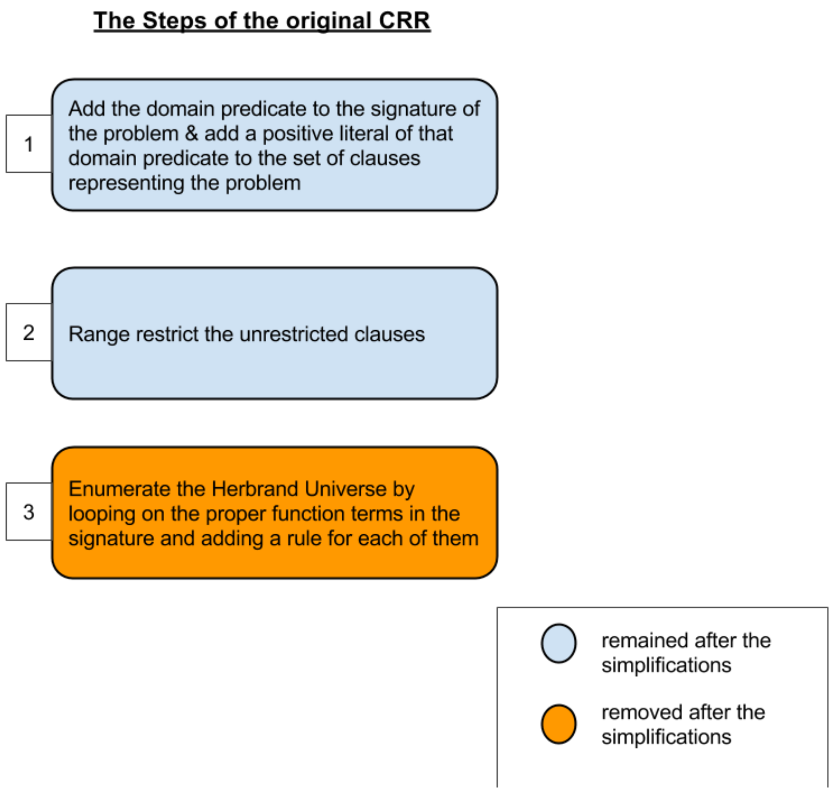

The Steps of the original CRR
1
Add the domain predicate to the signature of the problem & add a positive literal of that domain predicate to the set of clauses representing the problem
2
Range restrict the unrestricted clauses
3
Enumerate the Herbrand Universe by looping on the proper function terms in the signature and adding a rule for each of them
remained after the simplifications
removed after the simplifications

Figure 3.2: Original CRR with the kept and removed steps after the simplifications

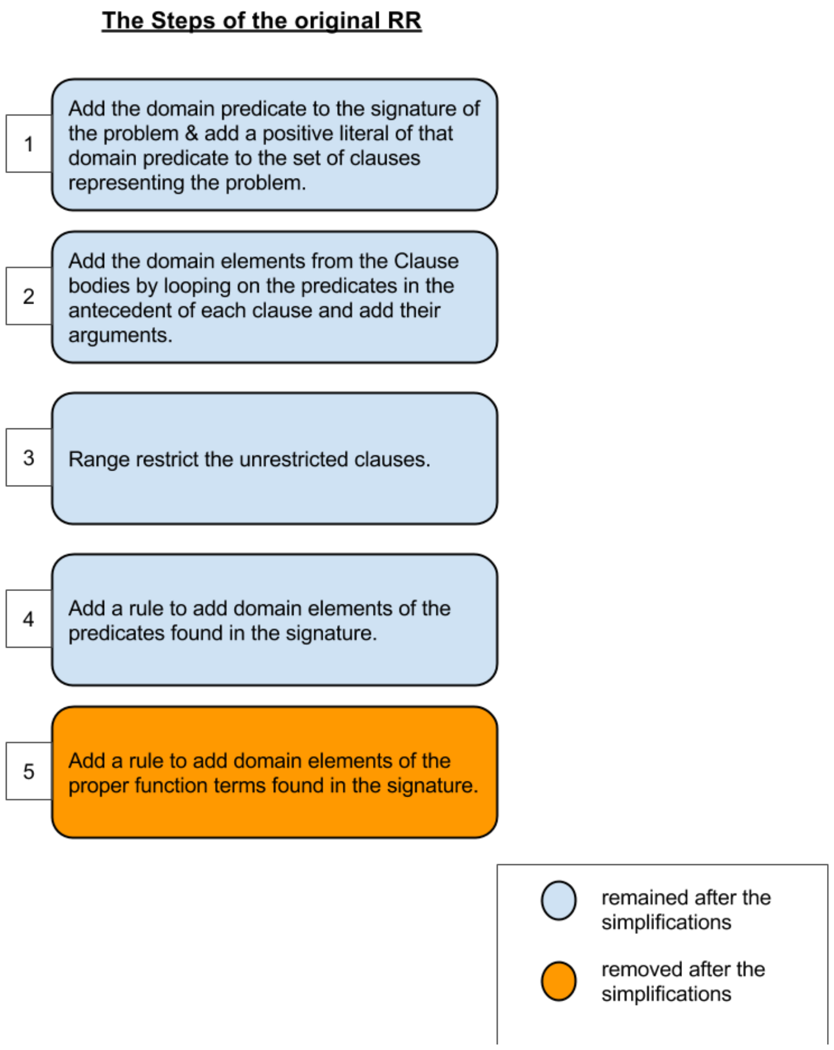

The Steps of the original RR
1
Add the domain predicate to the signature of the problem & add a positive literal of that domain predicate to the set of clauses representing the problem.
2
Add the domain elements from the Clause bodies by looping on the predicates in the antecedent of each clause and add their arguments.
3
Range restrict the unrestricted clauses.
4
Add a rule to add domain elements of the predicates found in the signature.
5
Add a rule to add domain elements of the proper function terms found in the signature.
remained after the simplifications
removed after the simplifications

Figure 3.3: Original RR with the kept and removed steps after the simplifications

The Intermediate output

Here we name the output of applying the simplified transformations on a set of (counter) satisfiable set of clauses an **Intermediate output**, since the the Output should be the returned explicit (counter) model. So In this part we give an example for a problem, and then show what is the Intermediate output of CRR, and RR.

Listing 3.2: Satisfiable CNF problem Example

```
    cnf(agatha,axiom,(~lives(a))).
    cnf(butler,axiom,(dies(X)
                     |dies(Y)
                     |~lives(a)
                     |~lives(Y))).
```

Listing 3.3: CRR output Example

```
cnf(i_0_1,axiom,(~lives(a))).
cnf(i_0_2,axiom,(dies(X1)|dies(X2)
                 |~lives(a)|~lives(X2)
                 |~dom(X1))).
cnf(i_0_3,plain,(dom(a))).
```

Listing 3.4: RR output Example

```
cnf(i_0_1,axiom,(~lives(a))).
cnf(i_0_2,axiom,(dies(X1)|dies(X2)
                 |~lives(a)|~lives(X2)
                 |~dom(X1))).
cnf(i_0_3,plain,(dom(a))).
cnf(i_0_4,plain,(dom(a)|~lives(X1))).
cnf(i_0_5,plain,(dom(a)|~lives(X2))).
cnf(i_0_6,plain,(dom(X4)|~lives(X4))).
cnf(i_0_7,plain,(dom(X5)|~dies(X5))).
```

Discussion on the Output of the prover

After applying the transformations on the clauses of a given problem, then this Intermediate output should be given as input to the normal proof procedure in E. And the final output for a (counter) satisfiable problem should be the saturated set, from which the explicit model should be extracted easily. However, this wasn't the case while running the problems, and the explicit model wasn't clear in the saturated set.

So a Model Construction Technique for saturation-based theorem provers was applied to the saturated set. And that Technique will be discussed in the following section 3.2

3.2 Model Construction

A Model Construction Technique had to be applied to the saturated set of clauses as discussed in the part related to the discussion on the output of the prover when the input was the range restricted version of the original set of clauses, and that was explained in subsection 3.1.2. Here in this section we will cover the following points:

- What is the Model Construction Technique used
- How does it work
- Discussion on the Output

What is the used technique for Model Construction

The Model Construction Technique used is specific for resolution based theorem provers, and it is the ground positive case of Bachmair and Ganzinger Model Construction Technique that was devised here in [15]. This technique originated from the proof of Bachmair and Ganzinger that their resolution based theorem proving technique is complete. That proof will be found in [2].

How does Bachmair and Ganzinger Model Construction Technique works

The chosen and implemented Bachmair and Ganzinger Model Construction Technique works for a saturated (counter) satisfiable set positive ground set of clauses. That has been generated using an ordered resolution system with simplification.

This Technique needs a term ordering that has been lifted to literals then lifted to clauses. That clause ordering should be total on ground clauses, and it should be the same one used in the saturation procedure.

The algorithm works as follows:

1. sort the positive ground clauses using the ordering in an ascending order
2. sort the literals in each clause to define the maximal literal
3. for each of the clauses starting from the smaller in terms of the ordering if not already true by the chosen true literals, then add the its maximal literal to the set of the chosen true literals

A pseudo-code for the implemented version of the algorithm is given below:

Listing 3.5: Ground Positive Case for Bachmair and Ganzinger Model Construction

```
Input: clauses_set % saturated set of clauses
         , ordering % ordering used in the saturation
{
         % model is the set of positive literals in the
         % constructed model, at the beginning it is
         % an empty set of literals
         model = {}

         % this sorts the set of clauses ascendingly
         sort_clauses(clauses_set , ordering)
         % it marks the maximal literal in each clause
         mark_maximal_literals(clauses_set , ordering)

         for clause:clause_set do:
         {
           % here it checks whether the current clause
           % is true by the partial model we have or not
           if not is_clause_true_by_model(clause , model) then:
           {
                % if not true yet the it gets the marked
                % maximal literal and adds it to model
                model = model + get_maximal_literal(clause)
           }
           end_if
         }
         end_for
}
Output: model
```

An Example for applying the Bachmair and Ganzinger Model Construction Technique is given below:

Listing 3.6: Example for applying Bachmair and Ganzinger Model Construction Technique

```
Let the unordered saturated set of clauses be:
{
        R(a,b)|Q(b),
        P(a),
        P(a)|Q(b),
        R(b,b)
}

Let the order of the present clauses:
{
        P(a) < Q(b) < R(a, b) < R(b, b)
}

% the left most literal in each of the
% ordered clauses is the maximal
```

Order	Ordered Clause	Partial Model	Change in Model
(1)	P(a)		P(a)
(2)	Q(b) | P(a)	P(a)	—
(3)	R(a, b) | Q(b)	P(a)	R(a, b)
(4)	R(b, b)	P(a), R(a, b)	R(b, b)

Table 3.1: Table to test captions and labels

```
Then the explicit Model:
{
        P(a),
        R(a,b),
        R(b,b)
}
```

Discussion on the Output

The output of the Model Construction Technique is the final one. It gives back the positive literals in the constructed explicit model. An Example for the output is given below in 3.7.

Listing 3.7: Example for the returned Model

```
dom(t).
bird(t).
fly(t).
```

Moreover, we augmented the output with a visual part. It prints out a dot graph representing the positive clauses linked with the positive literal, that belongs to the model, that makes them true or satisfiable in other words. Example of the returned dot graph will be listed in 3.8, and a rendered Figure for the same graph will be in 3.4.

Listing 3.8: Example of returned dot graph

```
digraph model {
 rankdir=LR;
 subgraph cluster_model {
  label="Model";
  0 [shape=ellipse,fillcolor=lightskyblue1,style=filled,label="
     fly(t)"]
  1 [shape=ellipse,fillcolor=lightskyblue1,style=filled,label="
     bird(t)"]
  2 [shape=ellipse,fillcolor=lightskyblue1,style=filled,label="
     dom(t)"]
 }
 subgraph cluster_clauses {
  label="Positive Ground Clauses";
  3 [shape=box,fillcolor=lightpink1,style=filled,label="cnf(
     i_0_3, plain, (fly(t)))."]
  3 -> 0
  4 [shape=box,fillcolor=lightpink1,style=filled,label="cnf(
     i_0_1, plain, (bird(t)))."]
  4 -> 1
  5 [shape=box,fillcolor=lightpink1,style=filled,label="cnf(
     i_0_2, plain, (dom(t)))."]
  5 -> 2
 }
}
```

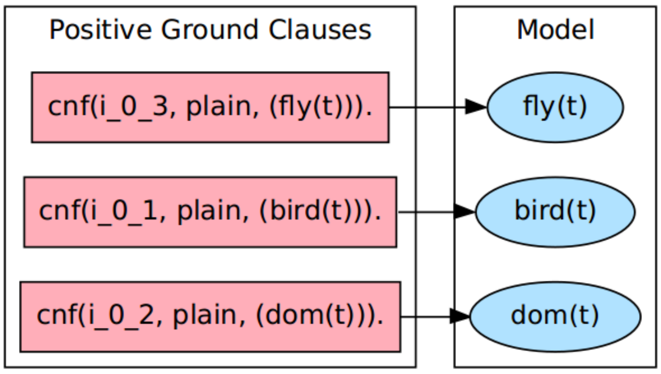

Positive Ground Clauses
cnf(i_0_3, plain, (fly(t))).
cnf(i_0_1, plain, (bird(t))).
cnf(i_0_2, plain, (dom(t))).
Model
fly(t)
bird(t)
dom(t)

Figure 3.4: Rendered dot graph of Model

3.3 Code FLow

So a small summary for the flow of the code is given below, and simplified flow chart is in Figure 3.5:

1. The input is an EPR (counter) satisfiable problem.
2. Simplified Range Restricted Transformations got applied on the input.
3. The intermediate output from the transformations act as input for the normal proof procedure.
4. The output from the prover, which is the saturated set of clauses, is the input for the Bachmair and Ganzinger Model Construction Technique.
5. Then the final output is the positive literals of the explicit Model for the given problem.

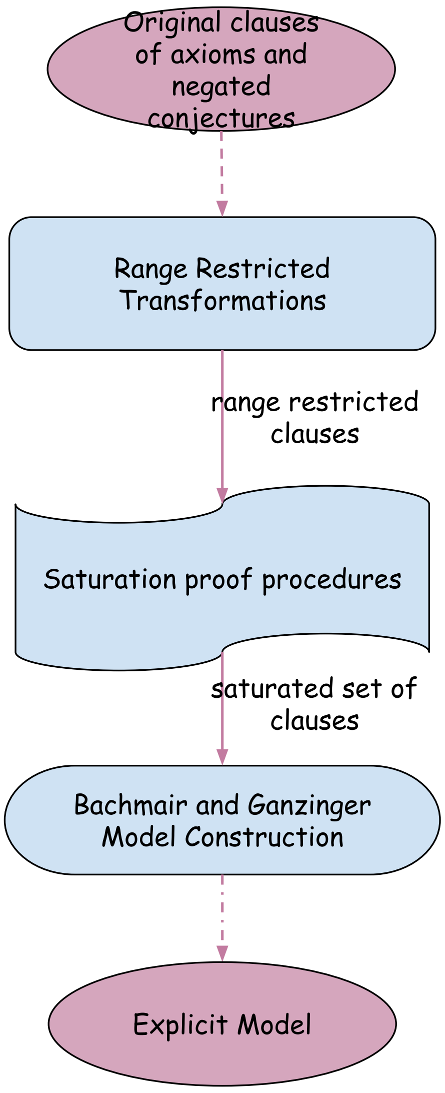

Original clauses of axioms and negated conjectures
Range Restricted Transformations
range restricted clauses
Saturation proof procedures
saturated set of clauses
Bachmair and Ganzinger Model Construction
Explicit Model

Figure 3.5: Simplified Flow chart for the code flow

Chapter 4

Testing and Validation

Testing and Validation

4.1 Efficiency

4.1.1 Memory Efficiency

The first part of the implementation which is related to applying the transformations to the the clause set introduces no memory leaks to the whole program.
Tools to check this:

1. Script implemented in E for giving a summary on the allocated and de-allocated memory structures, and the results showed that they are equal.

   ```
   \\
   Ex.:\\
   \# ————————————————————————————————————————————————————————
   \\
   \# Total SizeMalloc()ed memory: 68536168 Bytes (131507 requests)
   \\
   \# Total SizeFree()ed   memory: 68536168 Bytes (131507 requests)
   \\
   \# New requests:     214 (    197 by SecureMalloc(),      17 by SecureRea
   \\
   \# Total SecureMalloc()ed memory: 277647 Bytes
   \\
   \# Returned:         214 (    214 by FREE(),               0 by SecureReal
   \\
   \# SecureRealloc(ptr):      19 (     17 Allocs,        0 Frees,
   2 Reallocs)
   ```

\\
\# ————————————————————————
\\

2. Tool named ’valgrind’ which also showed the same results as the above script.

4.2 Accuracy of the transformations

The results of ./edisprover part is the same as the author of the paper implemented program in all the tested problems.

Chapter 5

Results and Literature review

Results and Literature review

5.1 Related Work

Automated Theorem Proving has many applications in various fields, e.g., Mathematical Reasoning, Knowledge Representation, Planning and Scheduling. The main process in automated theorem proving is proving the validity of a conjecture given a set of axioms, or if we are talking from the context of refutation-based theorem proving then the main process is proving the unsatisfiability of a given specification. And a lot of work and research was done to develop and enhance different calculi for that process.

The dual task for proving, or process in other words, is disproving, where a counter (model) is returned from the disproving procedure. Despite of the importance of the disproving process, less work ,compared to the proving process, were devoted for its development.

Different theorem provers

A lot of classifications can be done to automated theorem provers. One may target the different calculi applied by each of them such as Model Evolution Calculus as applied in Darwin for example or Instantiation-based theorem provers as in iProver. Other classification may target the output of each, where a distinction between the goal of each prover is considered. E.g., E is a refutation-based theorem prover that proves unsatisfiability, or in simpler words it proves the validity of a conjecture. However, The main goal for iProver is to give a (counter) model for unvalid conjectures. Other classification may exist for hybrid systems that do both jobs effectively.

Model Construction Techniques

Model Construction, or building, can be done using different approaches that depends on the type of logic it relates to. E.g., A very famous procedure for propositional logic is DPLL algorithm, which stands for Davis-Putnam-Logemann-LoveLand algorithm, it is mainly used to decide the satisfiability of a logical formula. In propositional logic, this problem can be reduced to the SAT problem, boolean satisfiability problem. Model evolution calculus is another example that its idea is based on DPLL, however it lifts DPLL in order to be applied for first order logic problems.

Another technique that is applied is Instantiation-based techniques, where it applies grounding on the clauses and then give the grounded set to a SAT solver, if the SAT solver detected the un-satisfiability of the clause set then it terminates, however if not then it generates new clauses using inference rules then repeats the whole process until saturation is reached.

Another set of tool that helps in the process of Model Generation, is Model finders. The functionality of those type of tools is mainly giving back a model for a satisfiable set of clauses with specifying the number of domain elements that you need to find for. Those tools is used for example in some Model evolution based theorem provers such as FM-Darwin.

There are different styles of model finders such as MACE-style and SEM-style model finders. MACE-style model finders work by transforming the first order problem to an equivalent propositional one considering the size of the domain while doing the transformation. And then use a SAT-solver on the transformed set. On the other hand, SEM-style model finders depends on searching and backtracking. An example for a MACE-style model finder is Paradox. However, Finder is an example for a SEM-style model finder. [7]

CASC competition

Nowadays, Automated theorem provers have a yearly competition named CASC competition [18, 26]. It is held on CADE or IJCAR conferences. In CASC different automated theorem provers compete to prove its efficiency over other theorem provers. The CASC competition's problems are chosen from the TPTP problem set. There are various division in the Competition, e.g. FOF, which is first order form non-propositional theorems, another one is SAT, which is clause normal form really non-propositional non theorems. The criteria for judgement is average time needed to solve problems, the number of solved problems, the numbers of problems solved + a solution output . Each division has its own contestants, and its own winners.

An Important division in the CASC competition, that is so related to that topic, is the EPR division, which represents effectively propositional clause normal form theorems and non-theorems. Moreover, EPR division has two sub-divisions, and they are:

- EPT: which have the theorems related problems, or in other words the unsatisfiable set of problems.
- EPS: which have the non-theorems, that is the satisfiable problems.

The latest CASC competition, that was CASC-J7, was held during the 7th International Joint Conference on Automated Reasoning (IJCAR) on 20th July 2014 http://cs.nyu.edu/ijcar2014/. The latest results for the EPR division is summarized in the following Figure 5.1:

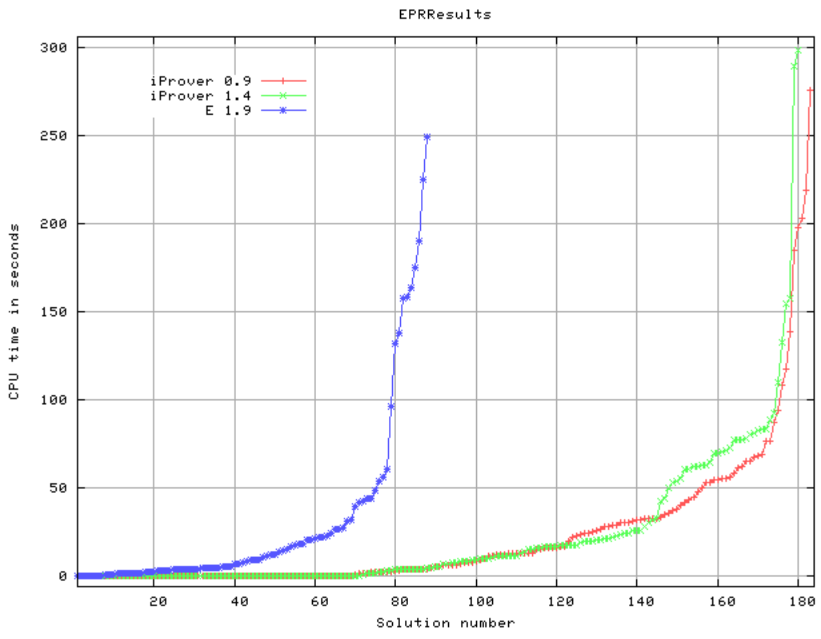
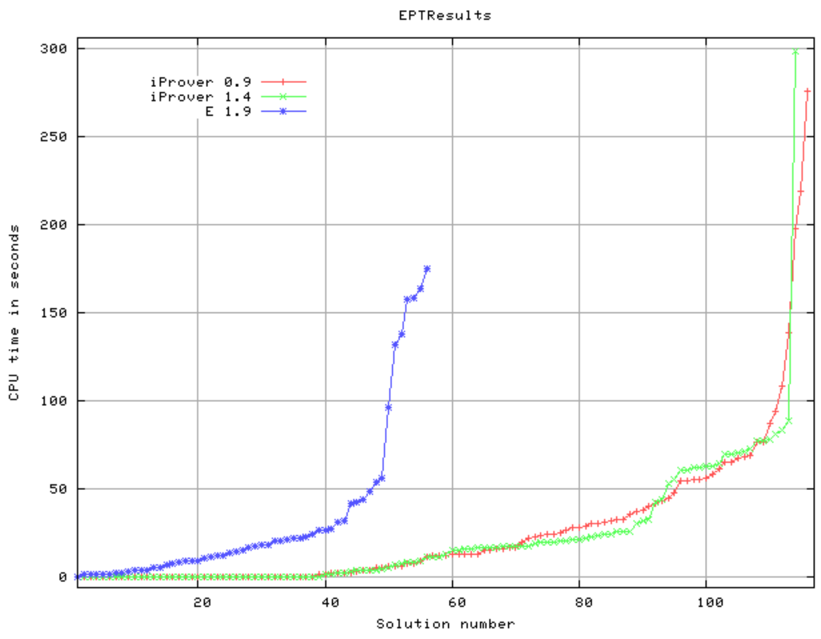
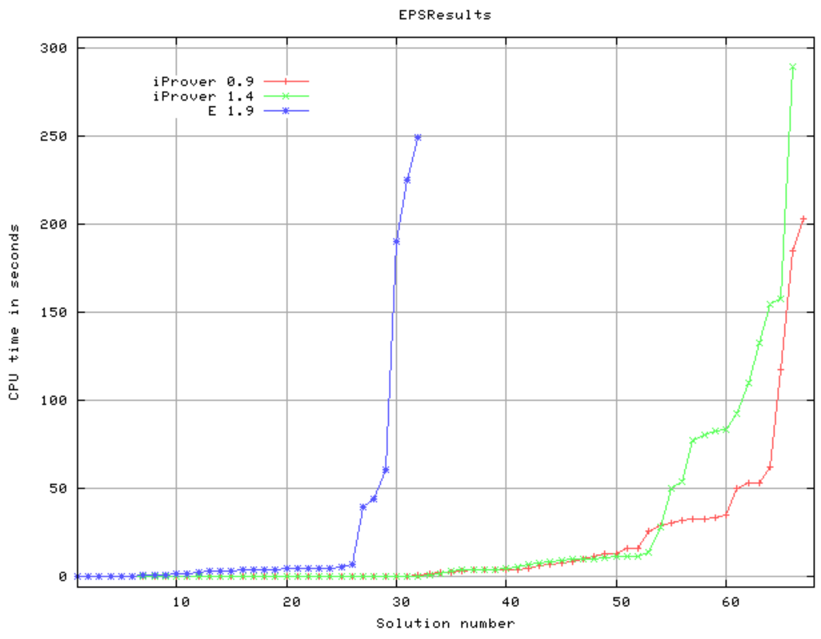


Figure 5.1: EPR division results taken from http://www.cs.miami.edu/~tptp/CASC/J7/WWWFiles/ResultsPlots.html



(a) EPT sub-division results



(b) EPS sub-division results

Figure 5.2: EPR sub-divisions results both taken from http://www.cs.miami.edu/~tptp/CASC/J7/WWWFiles/ResultsPlots.html

criteria	iProver-0.9	iProver-1.4	E-1.9
Solved(200)	183	180	88
Average-CPU-time	22.96	22.80	31.02
Solutions	62	174	88
New-Solved(10)	10	9	5

Table 5.1: Summary for EPR division results

EPR reasoning Techniques

Reasoning in EPR is very interesting, since it is a fragment of FOL that could be transformed to propositional logic with some processing such as grounding techniques. So both reasoning techniques done for FOL and propositional logic are valid for EPR fragment, however for the propositional case further handling will be required.

According to [17], we could classify methods used for EPR fragment reasoning into two categories:

- Grounding based methods, which are mainly techniques applied to limit the number of generated clauses in the grounding step, afterwards normal methods of reasoning for propositional calculus could be applied as the EPR problem will be reduced to a propositional one. Some of those grounding based methods are mentioned below:
  - Splitting
  - Pure Predicates
  - Linking restrictions
  - Sort inference
  - Incremental Search
- Non-grounding based methods, which are methods using normal inference systems and calculi used for generic FOL.
  - Resolution Calculus, which was first proposed in [22]
  - Model evolution Calculus, which was proposed by [4]
  - Instantiation Calculus, which was proposed in [10]

Resolution is the oldest one of all the three calculi. In the beginning the results mentioned in [11] showed that it is difficult to be able to decide EPR fragment using a resolution procedure. Later on, a resolution procedure was found to be able to decide EPR [8]. However, resolution is inefficient at least practically for EPR and this could viewed from the results of the EPR division in the yearly CASC competition.

Model evolution calculus has the same decidability for FOL as resolution with saturation as proposed here in [2]. So it is refutationally complete and sound. But it has an advantage over resolution, since it is in general a decision procedure for EPR fragment, however this is not the case with most resolution procedures. And practically the automated theorem provers that implement the Model evolution calculus have better results in the CASC competition for EPR division.

Instantiation Calculus has the same decidability for FOL as resolution with saturation and model evolution calculus. And it is decision procedure for EPR fragment like Model evolution calculus. And the automated theorem provers that depends on Instantiation calculus have a very good results in the CASC competition for the EPR division [29, 30], at least the first on this division in the last CASC competition. CASC-J7. is an Instantiation based theorem prover.

Table 5.2 found below summarizes the comparison between the three Non-grounding methods for reasoning in EPR.

Point of Comparison	Resolution	Model Evolution	Instantiation
Relatively New	No	Yes	Yes
Semi-decidable for FOL	Yes	Yes	Yes
Decidable for EPR	No, in general	Yes	Yes
Practically proved efficiency in EPR division	No	Yes	Yes
Example for Prover	E	Darwin	iProver

Table 5.2: Summary for Non-grounding methods for reasoning in EPR

Range Restricting Transformations

According to [3], The range restricted transformations implemented in this project, that was originally devised in [3], were also added to other theorem provers MSPASS and KRHyper. And the results showed improvement and effectiveness in trying them over the satisfiable set problems in TPTP, specifically Version 3.1.1.

Chapter 6

Conclusion

Conclusion

Chapter 7

Future Work

This project has a fertile environment were many aspects could be added and extended in a simple way. And those points will be discussed in the sections of this chapter in details. Moreover, those points could be viewed in two different categories, the first of them is implementation view and it will be discussed further in section 7.1, while the other is a testing and evaluational view and this will be presented in section 7.2.

7.1 Implementational Future Work

This section is devoted to discuss the related enhancements and implementations that could be added in E to achieve the goal of model construction. Most of those points will also help in evaluating the implemented technique in a way or another.
Some Points include modifications for the implemented part such as sub-section 7.1.1, while others will need a whole new implementation as in sub-section 7.1.4.

7.1.1 Extension for Transformations

As discussed before the original transformations mentioned were simplified because it was only intended for EPR sub-class in FOL. So a great extension that could be done is to extend the transformations to all sub-classes of FOL instead of only EPR by adding the simplified steps in the implemented crr and rr procedures on one hand, and by adding the shifting and blocking transformations on the other hand.

7.1.2 Adding Splitting Techniques

Another addition for that project that could be added is implementing splitting techniques. Since it was one of the limitations that did not make the implemented Transformations work on their own and needed further handling by augmenting it with the Bachmair and Ganzinger Model construction Technique.

So this could be done by adding a suitable splitting technique with backtracking, and in this case the Bachmair and Ganzinger Model Construction Technique won't be enabled, however another part for further handling would be needed to extract the explicit model from the saturated splitted set of clauses.

7.1.3 Implement Other Model Construction Techniques

Augmenting E with other Model Construction Techniques would add a value for it. As well as, it will make us have a good evaluation on the implemented Bachmair and Ganzinger Model Construction Technique since we will have a meaningful comparison on the performance and the effect of each of the different techniques.

7.1.4 More on Bachmair and Ganzinger Model Construction Technique

Adding the General case for Bachmair and Ganzinger Model Construction Technique that deals with the non ground case for the saturated set of clauses, as explained here in [15], may have a good output since the transformations will not be used in this case and it will act directly on the saturated set without having them acting. And this will be a good research point to compare the effect of the transformations as a Model Construction Technique.

7.2 Testing and Evaluational Future Work

Testing is very important to be able to evaluate any project. So the coming points are of a great importance to have a fair judgement on the implemented techniques and to discover the limitations of applying them in saturation-based theorem provers.

7.2.1 Testing on a Server

Testing medium sized and large sized problems on a large server would be of a great importance. Since only a personal computer of 4 GB RAM were used in the testing so only small sized problems were able to run on it without crashing. And the results of these problems is important to have a full overview on the performance and the impact of the project specially on those problems in the EPR set that were not terminating in the original configurations. Only after that we could have a fair evaluation on the project.

7.2.2 More Testing on the Transformations

More Testing on the Transformations is needed with consideration of the prover itself to know why the transformations alone did not perform what it was supposed to do. Then after knowing the reason that could be enhanced accordingly.

7.2.3 More Testing on the Bachmair and Ganzinger Model Construction Technique

Also More Testing on the implemented Bachmair and Ganzinger Model Construction Technique is of major importance at least to know the limitations of applying it in saturation-based theorem provers.

Appendix

Appendix A

Lists

FOL	First order logic
ATP	Automated theorem proving
EPR	Effectively propositional calculus
CRR	Classical Range Restricting Transformation
RR	Range Restricting Transformation
BS	Basic Shifting Transformation
PF	Partial Flattening Transformation
BL	Blocking Transformation

List of Figures

List of Tables

Listings

Appendix B

Syntax of FOL

Syntax of FOL

The syntax of FOL consists of:

- Predicates, which is a mapping for properties in a language. Moreover, the symbols used for representing predicates are finite and specific for each problem.
- Terms, which consists of functions and variables as shown below:
  - Functions, which itself can be divided according to the arity of the function symbol as follows:
    - if (arity == 0), then it is considered a constant
    - if (arity > 0), then it is a proper function symbol
  - Variables, and they are infinite list of symbols
- Special symbols
  - $\perp$ which represents false
  - $\top$ which represents true

Atoms which are the basic building blocks of a formula, follow the following format:

$$p(t1, ..., tk)$$

where p is a predicate symbols, any ti is a term, and k is the arity of the predicate symbol p. A Literal is an atom or a negated atom. A formula consists of only one atom is called an Atomic formula.

Compound formulas are formed by:

- Connectives, and they are divided into:

  - $\rightarrow$: implication
  - $\neg$ or $\sim$: negation
  - $\vee$ or $|$: disjunction
  - $\wedge$ or & : conjunction
  - $\equiv$: equivalence

- Quantifiers, and they are the following two:
  - $\forall$ which is the universal quantifier
  - $\exists$ which is the existential quantifier

A Clause is a disjunction of Literals. A ground clause is a clause having no variables. A positive clause is a clause who has no negated atoms. While a negative clause is a clause who contains only negated atoms. A mixed clause is a clause who consists of both atoms and negated atoms. A unit clause is a clause containing one Literal.

Bibliography

[1] Proving that androids, javas and pythons sorting algorithm is broken (and showing how to fix it). http://www.envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/. Accessed: 2015-7-27.

[2] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. *Handbook of automated reasoning*, 1:19–99, 2001.

[3] Peter Baumgartner and Renate A Schmidt. Blocking and other enhancements for bottom-up model generation methods. In *Automated Reasoning*, pages 125–139. Springer, 2006.

[4] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In Franz Baader, editor, *Automated Deduction CADE-19*, volume 2741 of *Lecture Notes in Computer Science*, pages 350–364. Springer Berlin Heidelberg, 2003.

[5] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic logic and mechanical theorem proving*. Academic press, 2014.

[6] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, pages 345–363, 1936.

[7] Koen Claessen and Niklas Sörensson. New techniques that improve mace-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications*, pages 11–27, 2003.

[8] Christian Fermüller. *Resolution methods for the decision problem*, volume 679. Springer Science & Business Media, 1993.

[9] Melvin Fitting. *First-order logic and automated theorem proving*. 1996.

[10] Harald Ganzinger and Konstantin Korovin. New directions in instantiation-based theorem proving. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, pages 55–64. IEEE, June 2003.

[11] William H. Joyner, Jr. Resolution strategies as decision procedures. *J. ACM*, 23(3):98–417, July 1976.

[12] Konstantin Korovin. System description: iprover-an instantiation-based theorem prover for first-order logic. In *IJCAR*, pages 292–298, 2008.

[13] Konstantin Korovin. Inst-gen–a modular approach to instantiation-based automated reasoning. In *Programming Logics*, pages 239–270. Springer, 2013.

[14] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *Computer Aided Verification*, pages 1–35. Springer, 2013.

[15] Christopher Lynch. Constructing bachmair-ganzinger models. In *Programming Logics*, pages 285–301. Springer, 2013.

[16] Juan Antonio Navarro and Andrei Voronkov. Proof systems for effectively propositional logic. In *Automated Reasoning*, pages 426–440. Springer, 2008.

[17] Juan Antonio Navarro-Pérez. *Encoding and Solving Problems in Effectively Propositional Logic.* PhD thesis, Citeseer, 2007.

[18] F.J. Pelletier, G. Sutcliffe, and C.B. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.

[19] Ruzica Piskac, Leonardo de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic with equality. Technical report, Technical Report MSR-TR-2008-181, Microsoft Research, 2008.

[20] Frederic Portoraro. Automated reasoning. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Winter 2014 edition, 2014.

[21] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI communications*, 15(2, 3):91–110, 2002.

[22] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

[23] Stephan Schulz. E-a brainiac theorem prover. *Ai Communications*, 15(2):111–126, 2002.

[24] Stephan Schulz. System description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 8312 of *Lecture Notes in Computer Science*, pages 735–743. Springer Berlin Heidelberg, 2013.

[25] Stewart Shapiro. Classical logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Winter 2013 edition, 2013.

[26] G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.

[27] G. Sutcliffe, C.B. Suttner, and F.J. Pelletier. The ijcar atp system competition. *Journal of Automated Reasoning*, 28(3):307–320, 2002.

[28] Geoff Sutcliffe. The tptp problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[29] Geoff Sutcliffe. The cade-24 automated theorem proving system competition–casc-24. *AI Communications*, 27(4):405–416, 2014.

[30] Geoff Sutcliffe. The 7th ijcar automated theorem proving system competition–casc-j7. *AI Communications*, 28, 2015. To appear.

[31] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(5):345–363, 1936.