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: 31 August, 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
31 August, 2015

Acknowledgments

First of all, I have to thank my supervisor Professor Dr. Stephan Schulz for his continuous support and understanding through out the last six months and being an example for a researcher.

Many thanks for Prof. Peter Baumgartner and Dr. Renate Schmidt for their help in understanding their paper, and answer my very long questions.

I thank my Eqyptians professsors and doctors for their help when being abroad specially Assoc. Prof. Haythem Ismail, and Dr. Hany El-Sharkawy.

I really can not say how much love I have for my whole family for their support and encouraging to achieve this stage of my education. Without them in my life, it would have been very hard to have any successes in my life.

And last but not least, My friends the ones who travelled with me and the others who were in Egypt were a gift from GOD to me. Thank you for your help, and encouraging.

Abstract

Automated theorem proving has applications in manby fields such as mathematics, and verification. 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.

Here in this project we added a model generation technique for a subclass of first order logic named effectively propositional in the 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 an explicit model for the specification.

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

Conjecture is a statement that its validity is not determined yet and need to be proved. There are two types of conjectures: Valid Conjectures, and Failed ones. Valid conjectures are the conjectures that are true in every interpretation of the problem specification. On the other hand, Failed conjectures are the ones that are inconsistent in some interpretation.

Given a specification consisting of some axioms and possibly some conjectures. Automated theorem provers can provide a proof and sometimes a derivation in the case of valid conjectures. Automated theorem provers are able to prove those specifications by many methods. Proving by contradiction is a leading one of those methods and it used by many Calculi. Moreover, it is used by the saturation based theorem prover E. The basic idea in proving by contradiction is having the axioms in addition to the negation of the conjecture(s). Then applying to them the different rules of the implemented calculus till a contradiction is found. So when you prove that the negation of x is false, then x is true in the first place.

So when the Conjectures are failed ones, the contradiction does not exist. However, a state of saturation is reached where no more clauses could be generated or simplified using any rules of the calculi. The clauses at the saturation phase are in some sense a model for the specification, since all the clauses satisfies the original set. But it is notable to mention that this model is not an explicit one.

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 algorithm, 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 over 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 is really beneficial to be able 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, or a fragment, named 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. Then an overview of related work will be in chapter 4. Afterwards in chapter 5 we will explain the procedures followed to test the accuracy and efficiency of the implemented techniques. Moreover, A discussion of the results will be found in chapter 6. Then a conclusion will be given in chapter 7. And last but not least a discussion for related future work will be in chapter 8.

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 First order logic (FOL) in section 2.1
- A background on Effectively propositional (EPR) in section 2.2
- A background on Problem set in section 2.3
- A background on E in section 2.4

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
- Equality

2.1.1 FOL overview and Example on it

Overview on FOL

Quantifiers, variables, and functions are what signifies FOL over propositional logic. Having functions and 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 here in appendix 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) $\forall X(Human(X) \rightarrow Mortal(X))$
(2) $Human(socrates)$
(3) $Mortal(socrates)$

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

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 [6]. But those trials were not successful. Later on, in the same year both [34, 7] 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 or may not halt/terminate at all, 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 [25]. So First order logic (FOL) is a semi-decidable logic.

2.1.3 Universe of FOL

Herbrand was a french mathematician who died at the age of 23 in 1931. Herbrand had numerous contributions in the field of logic. One of his major contributions is devising a procedure for generating the Universe of set of first order formulas; where the universe is the set of constants and grounded function symbols that represent terms in predicates. The procedure is defined recursively. So in most of the cases the Herbrand Universe of any set of first order formulas is infinite.

2.1.4 Equality

The Equality relation adds a lot of expressiveness and it is so intuitive to have it in logic. Adding equality to FOL is very beneficial such that it eases encoding problems to FOL, however it adds a burden for how it should be handled. So FOL with equality adds a distinguished predicate symbol $=$ or $\approx$ that represents equality to the set of predicate symbols. Moreover, it adds to the problem specification some axioms that helps in dealing with equality such as: reflexivity and symmetry.

In general, Equations are of the form: $t1 \approx t2$, Where each of the t1 and t2 is a term.

2.2 Effectively propositional (EPR)

This section is discussing important points related to EPR, so the following points will be covered:

- subsection 2.2.1 will explain what EPR is.
- Example on EPR will be given in subsection 2.2.2.
- Universe and Decidability of EPR will discussed in subsection 2.2.3.
- And last but not least in subsection 2.2.4 we will talk about Equality in EPR.

2.2.1 What is EPR

Effectively propositional (EPR) or sometimes known as Bernays-Schönfinkel class is a fragment of First order logic (FOL) in which all of its formulas follow the following format:

Listing 2.2: Format of EPR formula

$$\exists* \ \forall* \ F$$

where F is the formula.
Moreover, F has no proper functions symbols (all functions present are nullary ones "constants").

EPR fragment has a huge number of applications, e.g. Planning and Verifcation.

2.2.2 Example on EPR

Example for a formula in EPR:

Listing 2.3: Example for an EPR formula

$$\exists X \exists Y \forall Z \left(P(a, X, Y, Z)\right).$$

Keeping 2.2 in mind, this will make every skolemized formula that originally was in EPR format have no proper function symbols. After transforming the above equation, it will be:

Listing 2.4: Example for a skolemized EPR formula

$$\forall Z \left(P(a, b, c, Z)\right).$$

2.2.3 Decidability and Universe of EPR

Since the domain of EPR formulas contains only constants, and the number of constants in a formula is finite. So Herbrand universe of EPR formulas is finite. From that we could prove that EPR fragment is a decidable fragment in FOL, because of the finite number of ground interpretations it can have.

2.2.4 Equality in EPR

As we have mentioned before that EPR formulas will never contain proper function symbols. So terms will be one of two options:

- constant
- variable

And since Equality may be added to FOL as discussed in subsection 2.2.3, then we should consider the case if the formulas are from the EPR fragment. Equations are in the following form $t1 \approx t2$, where t1 and t2 are terms. And since terms in EPR are only variables and constants. So the ground case of the Equation will contain only of constants. This result is important and we will refer to it later in chapter 3.

2.3 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 [31], 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, and GRP. Those categories came from the scientific meaning of the different problems, 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 Automated theorem proving (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.

The theorem prover iProver is clear example on the integration between the different automated theorem provers . According to [15], iProver does not do the step of clausification by itself. It uses other theorem provers to have this step done. It uses Vampire for having the problem in CNF format, and it can use E as well to do so.

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

Listing 2.5: 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) )).
%------------------------------------------------------------------------------
```

2.4 The theorem prover E

2.4.1 What is E

E is a fully automatic theorem prover for FOL with equality implemented in ANSI-C that was released in 1998. Moreover, It is a saturation-based theorem prover. The Calculus implemented in E is variant of the superposition calculus proposed in [2].

2.4.2 What can E do

Since superposition calculus is a refutation one. So 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.

The returned saturated set is considered a model in the sense that it satisfies all the formulae in the give specification. However, an explicit model for the specification is not provided. Worth mentioning that it is not guaranteed in E to terminate/halt for EPR problems. This could be seen from the performance of E in EPR problems as it does not terminate in more than 50% of them.

2.4.3 The Code flow of E

This part is devoted from giving a brief on the main procedure in E. Figure 2.1 is a simplified version of the code flow in E.

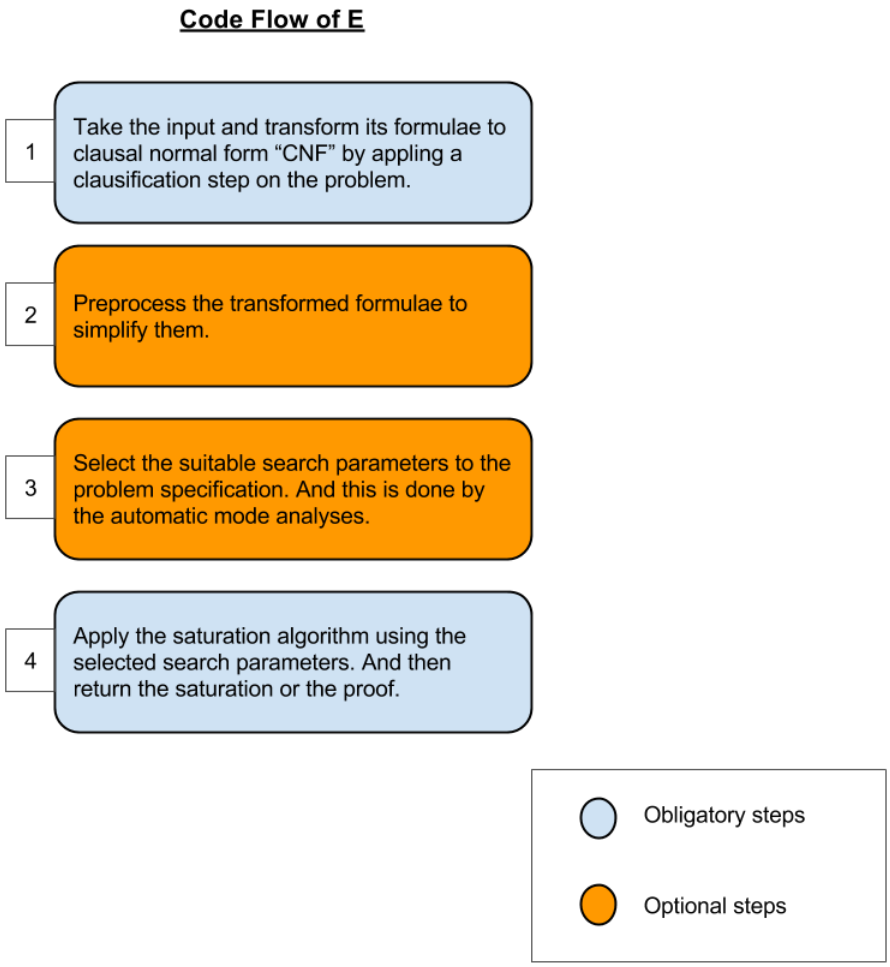


Figure 2.1: E Simplified code flow

2.4.4 Latest results

The latest results showed performance approaching 70% over all the CNF and FOF problems of the TPTP problem set and this is according to [27].

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.

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 their 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 [4]. A brief on the original transformations will be given in subsection 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 subsection 3.1.2.

3.1.1 Original transformations

The original procedures discussed in [4] 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 here in [23]. Moreover, those transformations are guaranteed to terminate for any given problem set, which should be a gain for us since most 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 succedent must exist in the antecedent as well.

An example for a range restricted clause is found below:

Listing 3.1: Range Restricted Clause Example

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

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

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 be 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 they are needed.
- **Shifting transformations**
  are the second two transformations. They are optional to be used. They complement one another, and do not replace each other. They were introduced mainly to prevent the non-termination of the transformations and to prevent generating redundant and unpleasant clauses, generated 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.

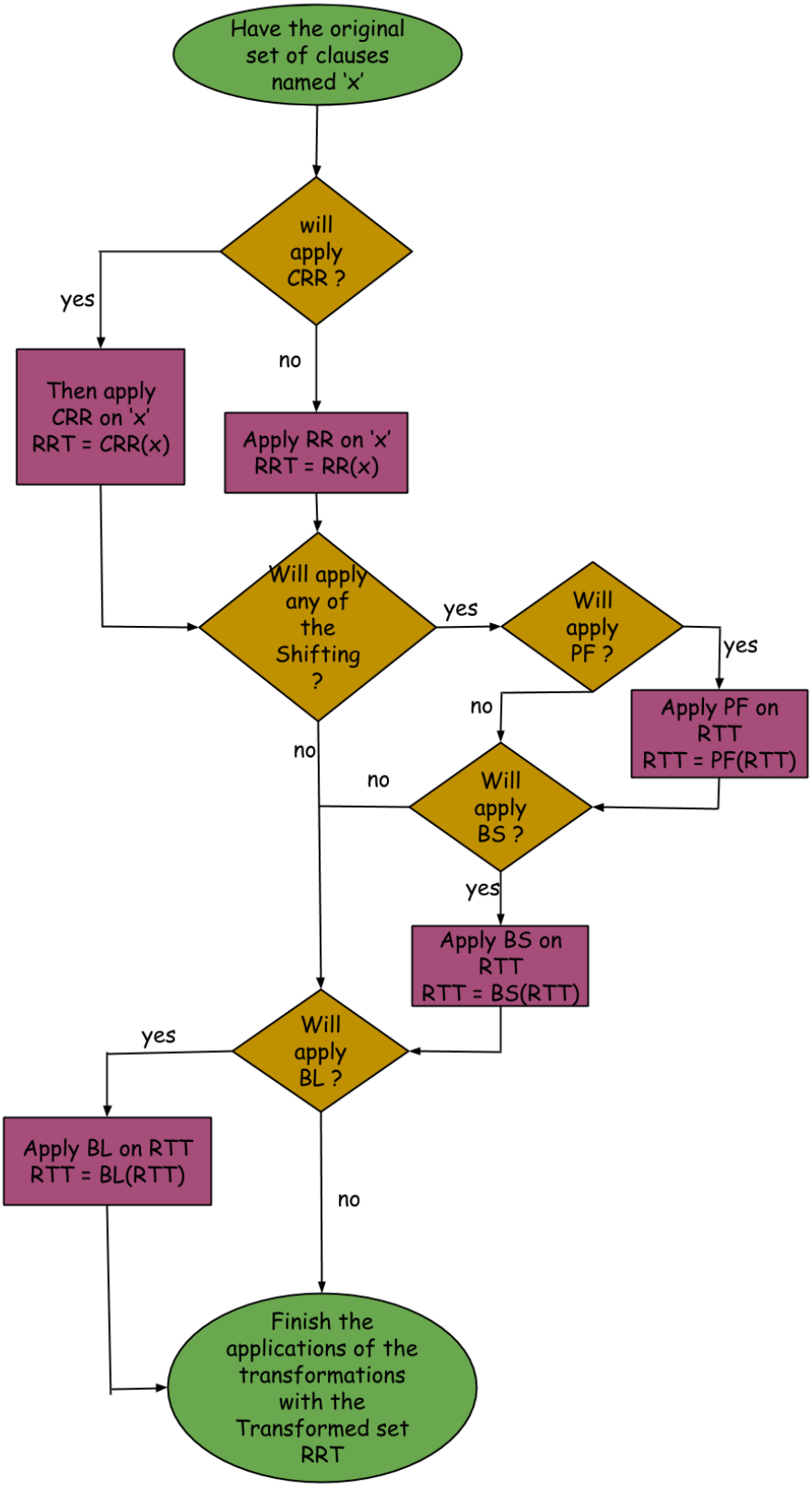


Figure 3.1: Flow of the original transformations

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. We will cover the following points in this subsection:

- 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 is for RR. And for a detailed explanation for the steps, you can check [4].

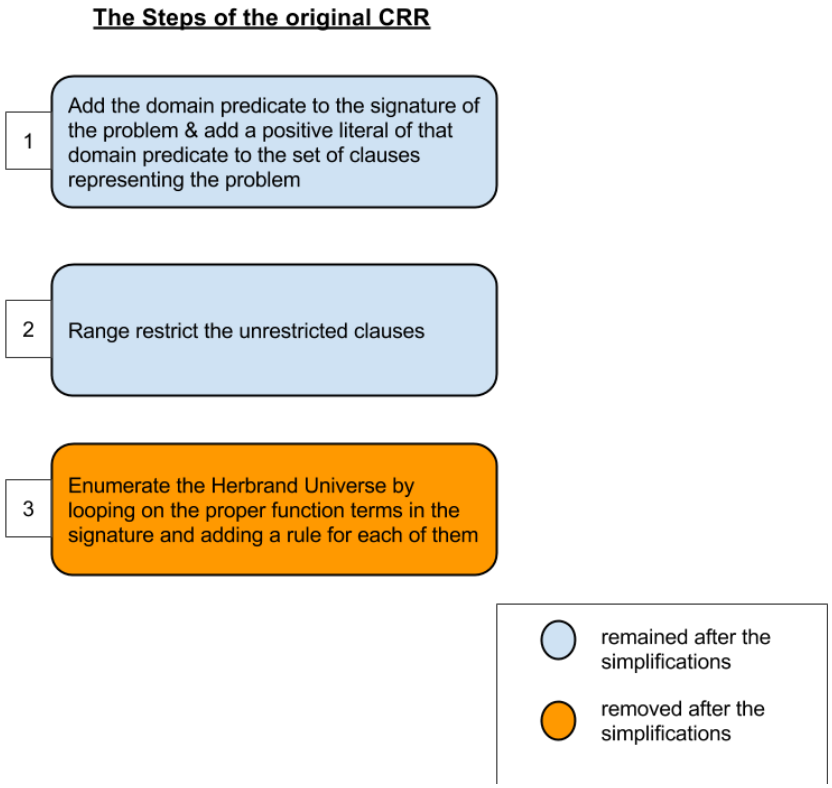


Figure 3.2: Original CRR with the kept and removed steps after the simplifications

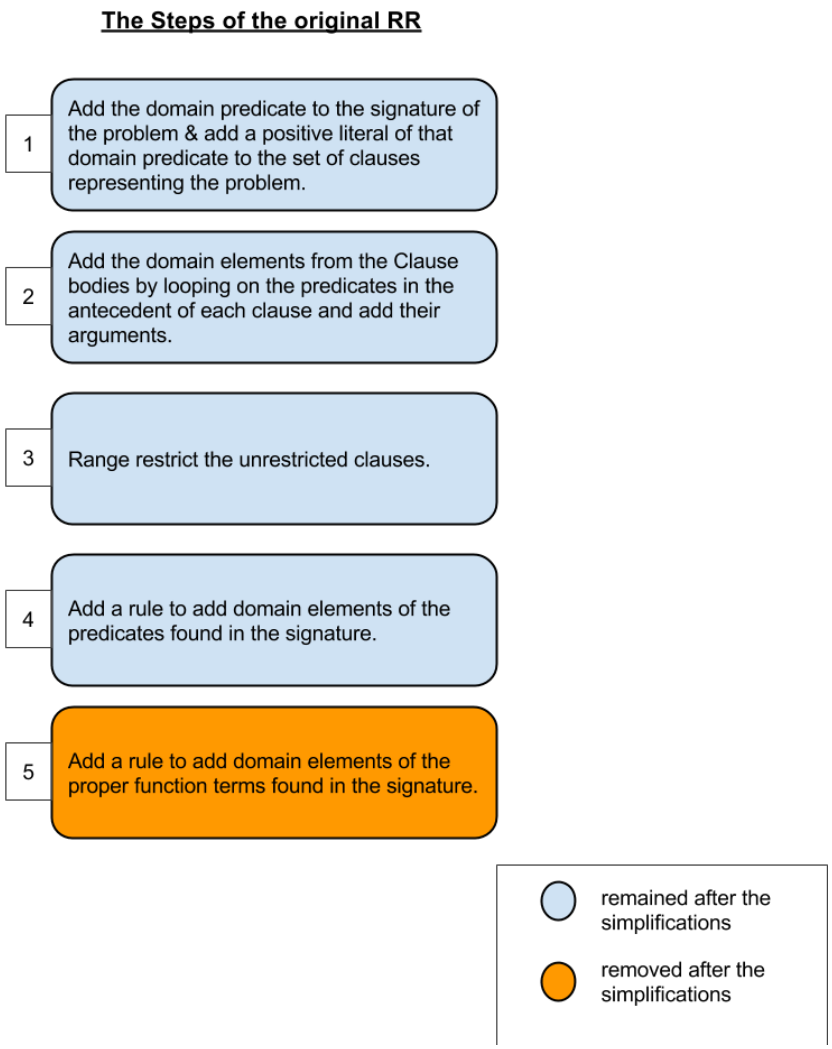


Figure 3.3: Original RR with the kept and removed steps after the simplifications

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 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 that problem from CRR, and RR.

It is notable to say that the following examples follow the TPTP CNF syntax. Moreover, the $\sim$ sign is used for negation, while the | sign represents an or sign.

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 resolution-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. This saturated set came from applying the normal prover to the transformed set of clauses.

In our case the simplified version of the range restricting transformations is the one implemented. 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 [18]. 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 [3].

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 of 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. For an explanation of term orderings, you can refer to [2]. And for specific discussion on the implemented orderings in the theorem prover E refer to [27].

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 in descending order in terms of the ordering
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:

Algorithm 1 Ground Positive Case for Bachmair and Ganzinger Model Construction

```
1: procedure EXTRACT_MODEL(clauses_set, ordering)
                          ▷ clauses_set is the saturated set of clauses
                          ▷ ordering is the used ordering in the saturation
2:     let model ← {}
                          ▷ model is the set of positive literals in the constructed model
                            at the beginning it is an empty set
3:     sort_literals(clauses_set, ordering)
                          ▷ sort_literals sorts the literals in each clause descendingly
4:     sort_clauses(clauses_set, ordering)
                          ▷ sort_clauses sorts the clauses ascendingly in the clause set
5:     for each clause c ∈ clauses_set do
6:         if is_true_by_model(c) ≠ true then
                          ▷ is_true_by_model checks whether the current clause is
                            true by the partial model we have or not
7:             set model ← model + get_maximal_literal(c)
                          ▷ get_maximal_literal if not true yet the it gets the maximal
                            literal, which is the first one since it is sorted, and adds it
                            to model
8:         end if
9:     end for
10:    return model
11: end procedure
```

In the implementation, We made a great use of the shared terms implemented in E. Since each term is only represented once in a term bank. So we added an attribute for the term structure representing whether it is evaluated to true or false. Comparison functions that takes the ordering into consideration were implemented. One is used for sorting the literals in a given clause. The other is for sorting the clauses in the clause set.

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

Listing 3.5: Example for applying Bachmair and Ganzinger Model Construction Technique (SETUP)

```
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: Applying Model Construction Technique

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

```
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
 }
}
```

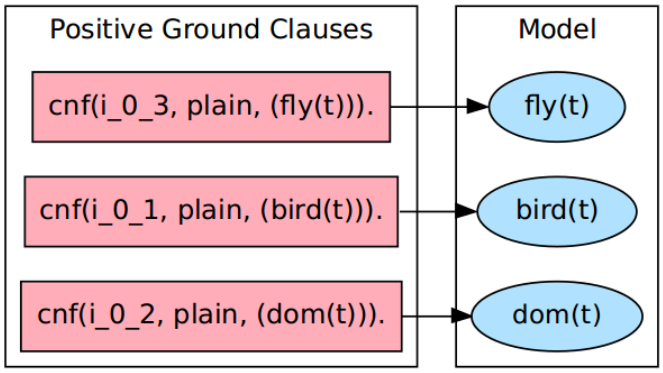


Figure 3.4: Rendered dot graph of Model

3.3 Code FLow

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.

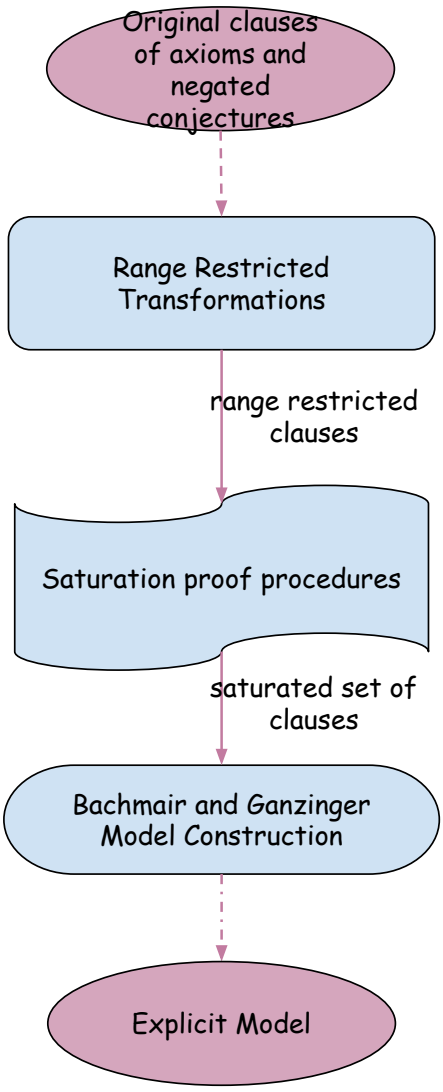


Figure 3.5: Simplified Flow chart for the code flow

Chapter 4

Related Work

Automated Theorem Proving has many applications in various fields, e.g., Mathematical Reasoning, Knowledge Representation, Planning and Scheduling. First we mention a defintion for ATP, according to [17], we could define ATP to be "is the use of a computer to prove non-numerical results, i.e. determine their truth (validiy)".

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.

In this chapter we will discuss the related work to the topic and bit of literature review over it. So it is divided into the following sections:

- Different Theorem provers
- Model Construction Techniques
- CASC competition
- Reasoning in EPR
- More on Range Restricting Transformations

4.1 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 or Resolution based as implemented in E.

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. It is notable to say that some automated theorem provers give both output according to the given problem.

Another Taxonomy for automated theorem provers relies on whether they are fully automated or interactive. Interactive automated theorem provers needs hints from human beings, however in the case of fully automated theorem proving no assistance is needed from humans. An example for a fully automated one is E, Vampire and Dawrin. On the other hand, IMPS [9] is an interactive example for a proof system.

4.2 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 instantiation on the active 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, are 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. [8]

4.3 CASC competition

Nowadays, Automated theorem provers have a yearly competition named CASC competition [21, 29]. 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 4.1:

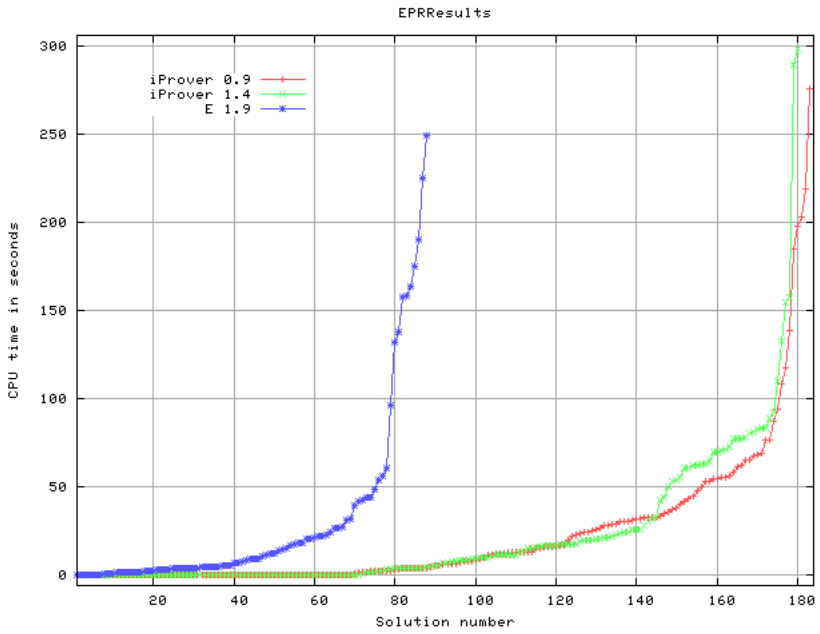
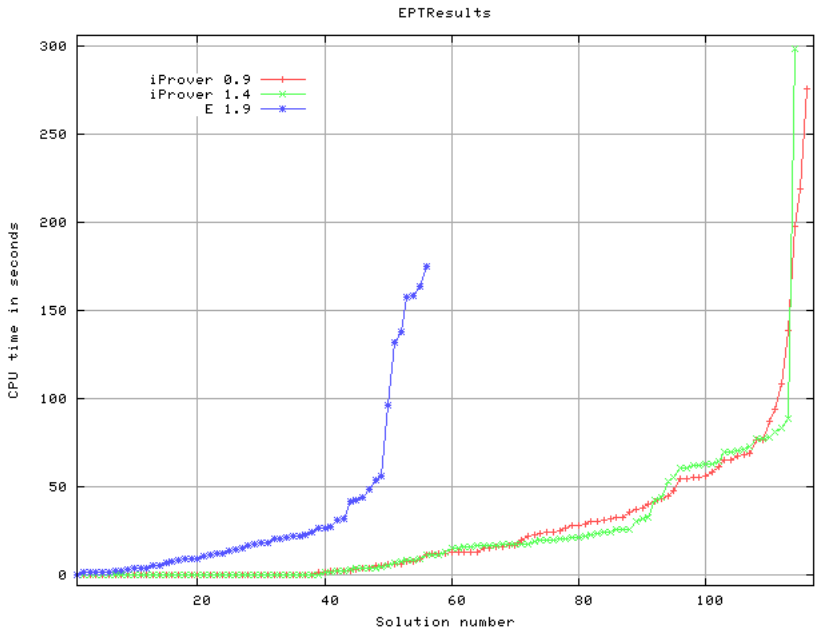
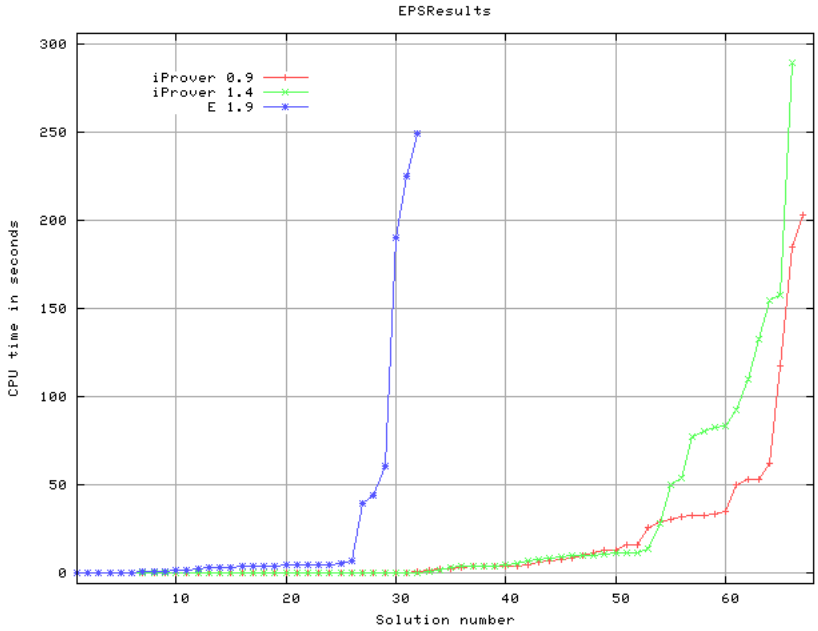


Figure 4.1: EPR division results taken from http://www.cs.miami.edu/~tptp/CASC/J7/WWWFiles/ResultsPlots.html

Moreover, latest results for the sub-divisions of EPR is given in figure 4.2; where sub-figure 4.2b shows the EPS results, while sub-figure 4.2a shows the EPT results. And table 4.1 is a summary for the results.



(a) EPT sub-division results



(b) EPS sub-division results

Figure 4.2: 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 4.1: Summary for EPR division results

4.4 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 [20], we could classify some of the methods used in reasoning for EPR fragment 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 [25]
  - Model evolution Calculus, which was proposed by [5]
  - Instantiation Calculus, which was proposed in [12]

Resolution is the oldest one of all the three calculi. In the beginning the results mentioned in [13] 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 [10]. 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 [3]. 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 [32, 33], at least the first on this division in the last CASC competition. CASC-J7. is an Instantiation based theorem prover.

Table 4.2 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 efficient in EPR division	No	Yes	Yes
Example for Prover	E	Darwin	iProver

Table 4.2: Summary for Non-grounding methods for reasoning in EPR

4.5 Range Restricting Transformations

According to [4], The range restricted transformations implemented in this project, that was originally devised in [4], 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 5

Testing, Validation and Evaluation

Here in this chapter, we explain what was done for validating the output and Evaluating the program. So we divide this chapter into two sections:

- Validating and Evaluating the Simplified Transformations.
- Validating and Evaluating the Implementation of Bachmair and Ganzinger Model Construction Technique.

5.1 Simplified Transformations

5.1.1 Evaluating Memory Efficiency

The first part of the implementation which is related to applying the transformations to 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.

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

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

5.1.2 Accuracy of the transformations

We tested the transformations over 204 out of 267 of the TPTP EPR SAT problems only over the Range Restricting Transformation (RR) procedure. The results of the transformations were exactly the same, except one, as a Prolog program called Yarralumla http://users.cecs.anu.edu.au/~baumgart/systems/yarralumla/, implemented by Peter Baumgartner, one of the authors of [4]. It worth mentioning that in our implementation we made an enhancement to the second step in the CRR transformation, which is the term abstraction step. According to the definition of second step, a clause like $P(a)|\ R(X)$ would result in the following two clauses $dom(a)|\ P(X)$ and $dom(X)|\ R(X)$. We only generated the first of them that gives the domain element a constant. The reason why we did that since the second clause is redundant with the definition of the forth step in RR. For the mentioned above clause, The following clauses would be generated from the forth step $dom(X)|\ P(X)$ and $dom(X)|\ R(X)$. So we only added the second clause in the forth step not in both the first and the forth steps. However, this modification did not affect the accuracy of the transformations. And only the absence of some redundant clauses were the difference between our implementation and Yarralumla's one.

We only tested over 204 of the 267 since the number of clauses in all of the remaining 63 problems exceeded 900 and that makes the program that transforms the two outputs to the same language crashes. But since in the rest of the 204 problems, 203 of them it was correct. Then we believe that our implementation is correct. The only problem that had a difference was HWC004-1.p. The reason for the difference between the two implementations was that in its SPC it is CNF_SAT_EPR. However, it is not an EPR problem since it contained function symbols. So this problem should not count in the EPR_SAT set. Details of the steps done in testing the accuracy of the transformations could be found in appendix C.

5.2 Bachmair and Ganzinger Model Construction Technique

5.2.1 Evaluating Memory Efficiency

The same tools mentioned for the transformations in 5.1.1 part were used. And the same results were achieved for the implemented model construction technique.

5.2.2 Accuracy of Implementation

The accuracy of the implementation were done manually for 11 problems. Those 11 problems will be mentioned in 6.2.

Chapter 6

Results

6.1 Testing for termination

The tests ran over all the (counter) satisfiable EPR problems in TPTP v6.1.0, and they were 267 problem. Only 101, which represents 37.8%, were terminated within 3 minutes time limit and 1GB memory limit. Table D.1 shows detailed results of running all the 267 problem with the time needed to terminate the problem. Another test were made but this time with 5 min. time limit and 2GB memory limit, and the results showed more 2 problems to terminate within that limit. Table D.2 shows the details of those 2 problems.

Figure 6.1 is a pie chart that shows the distribution of the terminated 101 problems with respect of rating of the problem.

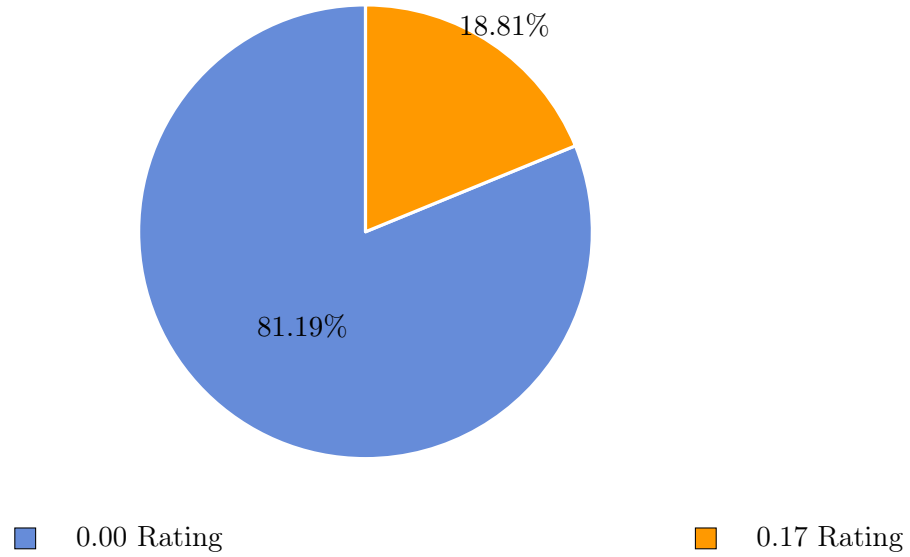


Figure 6.1: Distribution of terminated problems over different ratings

Table 6.1 shows how many problems were terminated in each category.

Category	Number of problems	Number of problems terminated
GRP	50	15
HWV	24	0
MGT	2	0
NLP	22	22
PUZ	13	9
SYN	114	32
HWC	1	1
KRS	24	19
MSC	1	0
PLA	4	0
SWB	3	3
SYO	9	0
Sum	276	101

Table 6.1: Problems solved per category

Some random problems were chosen to be tested thoroughly. SYN867-1 was one of them that gave interesting and weird results. The normal problem specification given to the prover terminated in 0.046s, however, when the transformed set was given to the prover,

it did not terminate till 1000 seconds passed. So this shows that transformations may add a burden over the prover itself, since the transformations add a lot of new clauses to original problem specification. In the case of SYN867-1 problem, the original specification composed of 58 clauses, however it increased tremendously after the transformations to be 722.

6.2 Testing for Model

11 problems were chosen to test their explicit model; one time with applying the transformations, another time without applying them. The models in the 2 cases were exactly the same, the only difference was that the models with the transformations included extra elements for the domain predict introduced in the transformations. Also it is notable that the all 11 problems were almost range restricted from the beginning.

Those problems with their rating are listed in the below table:

Problem	Rating	Problem	Rating
PUZ001-3.p	0.00	PUZ080+2.p	0.00
PUZ028-1.p	0.00	PUZ028-2.p	0.00
PUZ028-3.p	0.00	PUZ028-4.p	0.00
PUZ068+2.p	0.00	PUZ069+2.p	0.00
GRP125-4.004.p	0.17	GRP128-4.004.p	0.17
GRP123-8.004.p	0.17		

Table 6.2: Problems used to test the Model

Chapter 7

Conclusion

Here in this chapter we discuss what can we can conclude from that, However it gives back a lot of open questions at the same time.

With 37.8% termination from all satisfiable EPR problems, it showed that the range restricting transformations did not have a significant effect on the (counter) satisfiable EPR problems. It worth mentioning that most of the solved problems, that we checked, were almost range restricted problems from the beginning. So no clear benefit from the transformations were gained in range restricted problems.

One reason for that is that the theorem prover E has no backtracking splitting techniques as what implemented in MSPASS and KRHyper that the transformations showed promising performance when tried to them. However, it worth mentioning that splitting techniques is not obligatory to be used unless blocking is used, and in our case blocking is not used. So this seems to be not accurate enough when tried in E. Another reason is that transformations generate too many clauses which puts a heavy burden on the prover not if the normal specification were given to the prover right away.

By the use of Bachmair and Ganzinger model construction technique. We were finally able to extract the explicit minimal model from the saturated set. The current explicit model is the set of positive atoms that are true in the constructed model.

A gain from the project is that the transformations are implemented as a standalone program. So other theorem provers can try their output, which can help in having a good judgement on the effect of the transformations. Another gain is having now a TPTP to tme snytax converter that could be used in a very simple and easy way.

To sum up, the theorem prover E now has a mechanism to give back an explicit model when the given specification is a (counter) satisfiable one. But many future work could be done and this will be discussed in the next chapter 8.

At the end, we were able to add an explicit model construction in E with the help of range restricting transformations and Bachmair and Ganzinger model construction technique.

Chapter 8

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 8.1, while the other is a testing and evaluational view and this will be presented in section 8.2.

8.1 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 8.1.1, while others will need a whole new implementation as in sub-section 8.1.4.

8.1.1 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.

This could functionality could be added as a standalone part. In this case it would be helpful to have the transformations implemented so that other theorem provers could use it also to evaluate the effect of the transformations.

8.1.2 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.

8.1.3 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.

8.1.4 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 [18], 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 with the model construction technique.

8.2 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 resolution-based theorem provers.

8.2.1 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.

Also testing the transformations on the EPR problem set, the satisfiable and unsatisfiable problems, should be considered. In order to have a better overview of the termination problem on EPR set in resolution based theorem provers.

On the other hand, other automated theorem provers could use the implemented transformations and apply them. And see what are similarities and differences between the different automated theorem provers.

8.2.2 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
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

List of Examples

List of Algorithms

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.

Appendix C

Validation Details

Here in this chapter of the appendix, we give details for the validation checks that ware made for our implemented transformations.

In order to check the correctness of the transformations, we implemented a C program *tptp2tme.c* that takes a TPTP problem and then transform it to tme syntax, which is extended prolog language and the one used in the yarralumla program. We then implemented a Python script *compare_transformations.py* that takes the path of a folder as an input. It takes each one of the TPTP problems in the folder and then transform the problem into tme syntax. Then we applied Yarralumla to the tme version of problem and our transformations to the original TPTP problem.

Afterwards, we used some scripts implemented in the theorem prover SPASS and Yarralumla to change the output of both implementations of the transformations to TPTP syntax. Then we compared the two TPTP outputs with some C program *tptp_tme_compare.c* that we also implemented. This C program gives two outputs either True or False and the clause that were found in only one of the two transformations. In most of the problems *tptp_tme_compare.c* was sufficient to prove that they are the same. However, for some minor cases that needs further complicated handling it gives a False result. But in those cases, the results were checked manually and proved that the two transformations gave similar output.

Below table C.1 summarizes the results over the 203 problem. It gives how many problem the transformations ran correctly over it and the remaining was not checked because CPU limitations.

Category	Number of problems	Count of correctly transformed
GRP	50	50
HWV	24	8
MGT	2	2
NLP	22	22
PUZ	13	8
SYN	114	89
KRS	24	20
MSC	1	1
PLA	4	0
SWB	3	3
SYO	9	0
Sum	266	203

Table C.1: Transformations comparison Sat EPR TPTP problems

Appendix D

Detailed Results

Here in this chapter of the appendix, we give details for the results of the tests over all the 267 (counter) satisfiable EPR problems.

For the tests, we had a bash script running the transformations in the beginning then the normal prover then the model extraction part. Also it prints the time taken by each step. Moreover, time limit of 3 minutes and memory limit of 1GB were given to the 2nd step which is the normal prover itself. Only 101 of the problems were able to terminate within the time limit. The tests were done on a server with 8 GB RAM and 8 CPUs.

The below table D.1 gives the details of running the script over all the (counter) satisfiable EPR problems:

Problem	rating	transformations time	terminated	saturation time	model extraction time
GRP123-1.005.p	0.17	0.003s	no	–	–
GRP123-2.005.p	0.17	0.003s	no	–	–
GRP123-3.004.p	0.17	0.003s	yes	0.335s	0.016s
GRP123-4.004.p	0.17	0.003s	yes	2.060s	0.046s
GRP123-6.005.p	0.17	0.005s	no	–	–
GRP123-7.005.p	0.33	0.002s	no	–	–
GRP123-8.004.p	0.17	0.003s	yes	0.871s	0.029s
GRP123-9.004.p	0.17	0.003s	yes	0.506s	0.031s
GRP124-1.005.p	0.17	0.003s	no	–	–
GRP124-2.005.p	0.17	0.003s	no	–	–
GRP124-3.005.p	0.17	0.005s	no	–	–
GRP124-4.005.p	0.17	0.003s	no	–	–
GRP124-6.005.p	0.17	0.003s	no	–	–
GRP124-7.005.p	0.33	0.004s	no	–	–

GRP124-8.005.p	0.33	0.003s	no	–	–
GRP124-9.005.p	0.33	0.003s	no	–	–
GRP125-1.004.p	0.17	0.003s	yes	0.417s	0.016s
GRP125-2.004.p	0.17	0.006s	yes	0.069s	0.003s
GRP125-3.004.p	0.17	0.007s	yes	0.592s	0.007s
GRP125-4.004.p	0.17	0.003s	yes	2.338s	0.032s
GRP126-1.005.p	0.17	0.003s	no	–	–
GRP126-2.005.p	0.17	0.003s	no	–	–
GRP126-3.005.p	0.17	0.003s	no	–	–
GRP126-4.005.p	0.17	0.003s	no	–	–
GRP127-1.005.p	0.17	0.004s	no	–	–
GRP127-2.005.p	0.17	0.003s	yes	0.4411s	0.004s
GRP127-3.005.p	0.17	0.004s	no	–	–
GRP127-4.005.p	0.17	0.003s	no	–	–
GRP128-1.004.p	0.17	0.007s	yes	2.049s	0.033s
GRP128-2.004.p	0.17	0.007s	yes	0.316s	0.005s
GRP128-3.004.p	0.17	0.003s	yes	0.775s	0.007s
GRP128-4.004.p	0.17	0.005s	yes	58.915s	0.103s
GRP129-1.005.p	0.17	0.003s	no	–	–
GRP129-2.005.p	0.17	0.004s	no	–	–
GRP129-3.005.p	0.33	0.003s	no	–	–
GRP129-4.005.p	0.17	0.002s	no	–	–
GRP130-1.005.p	0.17	0.007s	no	–	–
GRP130-2.005.p	0.17	0.003s	no	–	–
GRP130-3.004.p	0.17	0.007s	yes	2.526s	0.007s
GRP130-4.004.p	0.17	0.005s	yes	1m7.036s	0.112s
GRP131-1.005.p	0.17	0.005s	no	–	–
GRP131-2.005.p	0.17	0.003s	no	–	–
GRP132-1.005.p	0.17	0.007s	no	–	–
GRP132-2.005.p	0.17	0.007s	no	–	–
GRP133-1.004.p	0.17	0.008s	no	–	–
GRP133-2.004.p	0.17	0.003s	no	–	–
GRP134-1.005.p	0.17	0.003s	no	–	–
GRP134-2.005.p	0.17	0.007s	no	–	–
GRP135-1.005.p	0.17	0.003s	no	–	–
GRP135-2.005.p	0.17	0.003s	no	–	–
HWC004-1.p	0.00	0.006s	yes	0.017s	0.002s
HWV042-1.p	0.50	0.010s	no	–	–
HWV042-2.p	0.67	0.012s	no	–	–
HWV048-1.p	0.50	0.012s	no	–	–
HWV048-2.p	0.83	0.020s	no	–	–
HWV049-1.p	0.67	0.020s	no	–	–
HWV049-2.p	0.67	0.016s	no	–	–
HWV053-1.p	0.67	2.449s	no	–	–

HWV054-1.p	1.00	0.622s	no	–	–
HWV062-1.p	0.50	1.412s	no	–	–
HWV063-1.p	0.88	2.795s	no	–	–
HWV066-1.p	0.50	0.346s	no	–	–
HWV067-1.p	1.00	3.688s	no	–	–
HWV070-1.p	1.00	1.976s	no	–	–
HWV071-1.p	0.67	0.038s	no	–	–
HWV074-1.p	0.83	0.910s	no	–	–
HWV075-1.p	1.00	2.064s	no	–	–
HWV076-1.p	0.50	0.050s	no	–	–
HWV077-1.p	1.00	0.123s	no	–	–
HWV079-1.p	0.50	0.020s	no	–	–
HWV080-1.p	1.00	0.364s	no	–	–
HWV081-1.p	1.00	0.128s	no	–	–
HWV082-1.p	0.83	1.309s	no	–	–
HWV085-1.p	0.83	0.689s	no	–	–
HWV086-1.p	1.00	11.613s	no	–	–
KRS021+1.p	0.00	0.073s	yes	0.017s	0.003s
KRS022+1.p	0.00	0.006s	yes	0.020s	0.005s
KRS023+1.p	0.00	0.003s	yes	0.016s	0.002s
KRS024+1.p	0.00	0.008s	yes	0.013s	0.004s
KRS025+1.p	0.00	0.004s	yes	0.021s	0.003s
KRS026+1.p	0.00	0.005s	yes	0.021s	0.002s
KRS027+1.p	0.00	0.003s	yes	0.015s	0.004s
KRS040+1.p	0.00	0.009s	yes	33.127s	0.007s
KRS041+1.p	0.00	0.012s	yes	0.034s	0.003s
KRS053+1.p	0.00	0.004s	yes	0.025s	0.002s
KRS054+1.p	0.00	0.008s	yes	0.022s	0.003s
KRS055+1.p	0.00	0.008s	yes	0.013s	0.005s
KRS056+1.p	0.00	0.004s	yes	0.018s	0.004s
KRS057+1.p	0.00	0.009s	yes	0.016s	0.006s
KRS058+1.p	0.00	0.007s	yes	0.016s	0.002s
KRS059+1.p	0.00	0.004s	yes	0.015s	0.004s
KRS060+1.p	0.00	0.003s	yes	0.016s	0.003s
KRS061+1.p	0.00	0.003s	yes	0.015s	0.002s
KRS062+1.p	0.00	0.008s	yes	0.016s	0.002s
KRS279-1.p	1.00	0.545s	no	–	–
KRS282-1.p	1.00	3.913s	no	–	–
KRS285-1.p	1.00	7.515s	no	–	–
KRS288-1.p	1.00	17.552s	no	–	–
KRS290-1.p	1.00	12.633s	no	–	–
MGT066-1.p	0.17	0.079s	no	–	–
MGT066+1.p	0.00	0.005s	no	–	–
MSC008-1.010.p	1.00	0.004s	no	–	–

NLP005-1.p	0.00	0.005s	yes	0.035s	0.004s
NLP006-1.p	0.00	0.004s	yes	0.021s	0.005s
NLP008-1.p	0.00	0.003s	yes	0.035s	0.004s
NLP012-1.p	0.00	0.003s	yes	0.028s	0.008s
NLP013-1.p	0.00	0.004s	yes	0.033s	0.004s
NLP023-1.p	0.00	0.006s	yes	0.029s	0.005s
NLP024-1.p	0.00	0.003s	yes	0.033s	0.005s
NLP042-1.p	0.00	0.003s	yes	0.024s	0.003s
NLP114-1.p	0.00	0.004s	yes	0.022s	0.007s
NLP115-1.p	0.00	0.003s	yes	0.025s	0.004s
NLP116-1.p	0.00	0.006s	yes	0.018s	0.006s
NLP118-1.p	0.00	0.002s	yes	0.025s	0.004s
NLP119-1.p	0.00	0.002s	yes	0.018s	0.009s
NLP120-1.p	0.00	0.002s	yes	0.026s	0.003s
NLP121-1.p	0.00	0.005s	yes	0.019s	0.006s
NLP123-1.p	0.00	0.002s	yes	0.028s	0.003s
NLP124-1.p	0.00	0.004s	yes	0.023s	0.004s
NLP125-1.p	0.00	0.009s	yes	0.018s	0.005s
NLP126-1.p	0.00	0.003s	yes	0.027s	0.005s
NLP127-1.p	0.00	0.004s	yes	0.024s	0.004s
NLP128-1.p	0.00	0.003s	yes	0.021s	0.003s
NLP129-1.p	0.00	0.007s	yes	0.019s	0.003s
PLA038-1.p	0.83	0.163s	no	–	–
PLA040-1.p	0.83	0.285s	no	–	–
PLA041-1.p	1.00	1.040s	no	–	–
PLA043-1.p	0.50	0.209s	no	–	–
PUZ001-3.p	0.00	0.003s	yes	0.017s	0.005s
PUZ018-2.p	0.17	0.002s	yes	3.702s	0.278s
PUZ028-1.p	0.00	0.007s	yes	0.018s	0.004s
PUZ028-2.p	0.00	0.004s	yes	0.044s	0.009s
PUZ028-3.p	0.00	0.018s	yes	0.093s	0.007s
PUZ028-4.p	0.00	0.013s	yes	0.067s	0.009s
PUZ052-1.p	1.00	0.005s	no	–	–
PUZ053-1.p	1.00	0.010s	no	–	–
PUZ068+2.p	0.00	0.630s	yes	7.895s	0.051s
PUZ069+2.p	0.00	0.610s	yes	50.541s	0.046s
PUZ079+2.p	0.00	0.609s	no	–	–
PUZ080+2.p	0.00	0.640s	yes	27.216s	0.045s
PUZ138+2.p	0.00	0.614s	no	–	–
SWB011+4.p	0.00	0.009s	yes	0.043s	0.016s
SWB031+4.p	0.00	0.010s	yes	0.037s	0.016s
SWB035+1.p	0.00	0.006s	yes	0.039s	0.007s
SYN056-1.p	0.00	0.004s	yes	0.017s	0.006s
SYN059-1.p	0.00	0.006s	yes	0.026s	0.002s

SYN307-1.p	0.17	0.002s	yes	0.020s	0.004s
SYN317-1.p	0.00	0.007s	yes	0.015s	0.003s
SYN322-1.p	0.00	0.002s	yes	0.019s	0.002s
SYN418-1.p	0.17	0.010s	no	–	–
SYN419-1.p	0.17	0.013s	no	–	–
SYN420-1.p	0.33	0.019s	no	–	–
SYN421-1.p	0.17	0.014s	no	–	–
SYN422-1.p	0.17	0.019s	no	–	–
SYN423-1.p	0.33	0.014s	no	–	–
SYN424-1.p	0.33	0.018s	no	–	–
SYN425-1.p	0.17	0.015s	no	–	–
SYN426-1.p	0.33	0.019s	no	–	–
SYN427-1.p	0.33	0.022s	no	–	–
SYN428-1.p	0.33	0.020s	no	–	–
SYN429-1.p	0.33	0.027s	no	–	–
SYN430-1.p	0.00	0.009s	yes	33.533s	0.082s
SYN431-1.p	0.17	0.010s	yes	0.621s	0.015s
SYN432-1.p	0.00	0.003s	yes	21.682s	0.059s
SYN433-1.p	0.00	0.005s	no	–	–
SYN434-1.p	0.33	0.006s	no	–	–
SYN435-1.p	0.33	0.005s	no	–	–
SYN437-1.p	0.33	0.010s	no	–	–
SYN438-1.p	0.17	0.010s	no	–	–
SYN441-1.p	0.33	0.012s	no	–	–
SYN446-1.p	0.17	0.010s	no	–	–
SYN449-1.p	0.33	0.004s	no	–	–
SYN453-1.p	0.33	0.011s	no	–	–
SYN456-1.p	0.33	0.005s	no	–	–
SYN463-1.p	0.33	0.006s	no	–	–
SYN464-1.p	0.33	0.010s	no	–	–
SYN490-1.p	0.00	0.003s	yes	0.263s	0.018s
SYN491-1.p	0.00	0.005s	yes	0.043s	0.005s
SYN492-1.p	0.00	0.007s	yes	0.025s	0.003s
SYN493-1.p	0.00	0.005s	yes	0.024s	0.003s
SYN494-1.p	0.00	0.002s	yes	0.064s	0.006s
SYN495-1.p	0.00	0.004s	yes	1.272s	0.038s
SYN496-1.p	0.00	0.003s	yes	0.026s	0.003s
SYN497-1.p	0.00	0.004s	yes	0.024s	0.004s
SYN513-1.p	0.33	0.013s	no	–	–
SYN514-1.p	0.33	0.010s	no	–	–
SYN515-1.p	0.00	0.006s	no	–	–
SYN516-1.p	0.00	0.006s	yes	0.809s	0.029s
SYN517-1.p	0.00	0.004s	yes	5.968s	0.120s
SYN518-1.p	0.33	0.013s	no	–	–

SYN519-1.p	0.33	0.018s	no	–	–
SYN520-1.p	0.33	0.018s	no	–	–
SYN521-1.p	0.00	0.006s	yes	0.239s	0.013s
SYN522-1.p	0.00	0.007s	no	–	–
SYN523-1.p	0.00	0.007s	yes	0.353s	0.018s
SYN524-1.p	0.00	0.011s	yes	0.602s	0.024s
SYN525-1.p	0.00	0.005s	no	–	–
SYN526-1.p	0.00	0.006s	yes	3.691s	0.076s
SYN527-1.p	0.00	0.003s	yes	3.749s	0.075s
SYN528-1.p	0.00	0.004s	yes	2m10.319s	0.336s
SYN529-1.p	0.00	0.007s	no	–	–
SYN530-1.p	0.00	0.011s	yes	2m19.609s	0.329s
SYN531-1.p	0.00	0.003s	yes	16.203s	0.189s
SYN532-1.p	0.00	0.003s	yes	1.260s	0.062s
SYN533-1.p	0.00	0.009s	no	–	–
SYN534-1.p	0.00	0.004s	yes	0.733s	0.029s
SYN535-1.p	0.00	0.007s	yes	1.347s	0.062s
SYN536-1.p	0.00	0.006s	no	–	–
SYN537-1.p	0.00	0.008s	no	–	–
SYN538-1.p	0.00	0.006s	no	–	–
SYN539-1.p	0.00	0.012s	no	–	–
SYN540-1.p	0.17	0.005s	yes	26.690s	0.127s
SYN541-1.p	0.00	0.005s	yes	8.123s	0.101s
SYN542-1.p	0.33	0.008s	no	–	–
SYN543-1.p	0.17	0.005s	no	–	–
SYN544-1.p	0.17	0.016s	no	–	–
SYN545-1.p	0.33	0.013s	no	–	–
SYN546-1.p	0.33	0.015s	no	–	–
SYN547-1.p	0.33	0.011s	no	–	–
SYN720-1.p	0.00	0.010s	yes	0.268s	0.036s
SYN811-1.p	0.33	0.260s	no	–	–
SYN812-1.p	0.17	2.332s	no	–	–
SYN814-1.p	0.33	0.335s	no	–	–
SYN815-1.p	0.33	0.976s	no	–	–
SYN816-1.p	0.33	0.934s	no	–	–
SYN817-1.p	0.17	1.084s	no	–	–
SYN818-1.p	0.33	3.163s	no	–	–
SYN821-1.p	0.50	1.144s	no	–	–
SYN822-1.p	0.33	1.160s	no	–	–
SYN823-1.p	0.17	0.079s	no	–	–
SYN824-1.p	0.33	0.657s	no	–	–
SYN825-1.p	0.33	2.141s	no	–	–
SYN826-1.p	0.33	2.291s	no	–	–
SYN827-1.p	0.17	0.114s	no	–	–

SYN828-1.p	0.17	0.955s	no	–	–
SYN829-1.p	0.33	3.001s	no	–	–
SYN830-1.p	0.33	3.315s	no	–	–
SYN831-1.p	0.33	3.066s	no	–	–
SYN832-1.p	0.33	2.805s	no	–	–
SYN835-1.p	0.17	0.114s	no	–	–
SYN838-1.p	0.33	1.127s	no	–	–
SYN839-1.p	0.33	3.504s	no	–	–
SYN840-1.p	0.33	3.297s	no	–	–
SYN841-1.p	0.33	3.585s	no	–	–
SYN842-1.p	0.33	3.995s	no	–	–
SYN851-1.p	0.33	1.143s	no	–	–
SYN852-1.p	0.33	3.922s	no	–	–
SYN853-1.p	0.33	3.921s	no	–	–
SYN854-1.p	0.33	3.785s	no	–	–
SYN855-1.p	0.33	3.872s	no	–	–
SYN863-1.p	0.33	1.165s	no	–	–
SYN864-1.p	0.33	1.206s	no	–	–
SYN867-1.p	0.17	0.021s	no	–	–
SYN868-1.p	0.17	0.019s	no	–	–
SYN870-1.p	0.17	0.013s	no	–	–
SYN872-1.p	0.33	0.113s	no	–	–
SYN888-1.p	0.33	0.151s	no	–	–
SYN902-1.p	0.33	0.164s	no	–	–
SYO583-1.p	1.00	21.599s	no	–	–
SYO584-1.p	1.00	3.091s	no	–	–
SYO585-1.p	1.00	2.859s	no	–	–
SYO586-1.p	1.00	0.457s	no	–	–
SYO590-1.p	1.00	0.881s	no	–	–
SYO593-1.p	0.67	0.180s	no	–	–
SYO595-1.p	0.67	0.137s	no	–	–
SYO599-1.p	1.00	42.541s	no	–	–
SYO603-1.p	1.00	1.055s	no	–	–

Table D.1: Results of Sat EPR TPTP problems

Another test were made but the time limit was 300 seconds instead of 180 seconds and 2GB memory limit. Only two more problems were solved within that limit. The below table D.2 shows them:

Problem	rating	transformations t.	saturation t.	model extraction t.
GRP130-2.005.p	0.17	0.004s	3m16.420s	0.006s
PUZ079+2.p	0.00	0.606s	3m30.429s	0.039s

Table D.2: More Solved Sat EPR TPTP problems

Bibliography

[1] Proving that androidâĂŹs, javaâĂŹs and pythonâĂŹs 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. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

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

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

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

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

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

[8] 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.

[9] WilliamM. Farmer, JoshuaD. Guttman, and F.Javier Thayer. Imps: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11(2):213–248, 1993.

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

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

[12] 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.

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

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

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

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

[17] Donald W Loveland. Automated theorem proving: a quarter century review. *Automated Theorem Proving: After*, 25:1–45, 1984.

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

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

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

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

[22] 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.

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

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

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

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

[27] 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.

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

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

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

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

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

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

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