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.

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 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 concerned with 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.

2.1 Background on first order logic topics

FOL, which is also known as first order predicate logic, is an expressive logic that allows us to formulate most of our spoken language sentences in a defined way such that it could be further handled with rules such as simplification, inference rules, etc.
We could represent formulas in FOL in so many forms. So 2.1.1 will be devoted to that part of background.
Moreover, in general the universe of FOL is infinite because of the existence of the quantifiers and function terms. So in 2.1.2 some topics related to that property will be mentioned including Herbrand Universe.
Also this project is concerned to a specific class of FOL its calculus named EPR, and this will be discussed in 2.1.4.

2.1.1 Different forms of FOL

FOL can be represented in different forms. Even some algorithms need to work with specific one. Moreover, there are some procedures that could transform a formula from one form to another. Here in this part, we will discuss the most important forms and the relevant ones to the project.
1- general form
2- clausal normal form
3- prenex normal form
4- skolem normal form

General Form

General form

Clausal normal form

Clausal normal form (CNF)

Prenex normal form

Prenex normal form (PNF)

Skolem normal form

Skolem normal form (SNF)

2.1.2 The universe of FOL

Discuss the infinity of the universe in general.
And then mention the Herbrand Universe.

2.1.3 Skolemization

Skolemization is the step that transforms PNF formulas into SNF. In which all the existentially quantified variables are removed and replaced by some function terms and its arguments are all the universally quantified variables appeared before the one in concern. Example for a formula in PNF:

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

After it transformed into SNF:

$$\forall X \forall Y \left(P(a, b, X, Y, f(X, Y)) \right).$$

2.1.4 EPR

EPR or sometimes known as Bernays-Schoenfinkel class is a class of first order logic in which all of its formulas follow the following format:
$\exists *$ $\forall *$ F, where F is the formula. Moreover, F has no proper functions symbols (all functions present are nullary ones "constants").
Example for a formula in EPR:

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

Keeping 2.1.3 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:

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

2.2 TPTP Problem set

TPTP problems are first order problems that are considered benchmarks to measure the performance of the theorem provers. TPTP problem set consists of different categories of problems such as : PUZ, NLP, etc....

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 applies 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 [4]

Chapter 3

Methodology and Implementation

Methodology and Implementation

3.1 Transformations

The methods applied in the project to extract the models follows the transformations discussed in [2]. A brief on the 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. Those simplifications will be discussed in 3.1.2.

3.1.1 Original transformations

The original procedures discussed in [2] generally work for all sub-classes of FOL. Those procedures should be applied to a given set of axioms in a specific form "implication form" which is explained here – , and here – to know how to transform to that form.
Transformations are series of procedures, mainly about changing the clauses to certain form named range restricted form. In general the transfromations deal with clauses in the implication from. Range restricted clauses: are the clauses in which all the variables that appear in the succeedent must exist in the anticedent as well. Transformations also add a domain predicate that will help in finding the interpretations.
There are 3 types of transformations:
First 2 transformations: "Range restricting transformations" 'crr' and a variant of it 'rr' mainly responsible for changing the clauses to the range restricted form. The difference is that 'rr' adds elements to the domain when it needs them, while 'crr' enumerate the Herbrand Universe.
Second 2 transfromations: "Shifting transformations" 'bs' and 'pf' mainly for preventing non-termination and generating unpleasant clauses from steps in 'rr'
Last transfromation: "Blocking transformation" 'bl' mainly for detecting periodictiy, so it defines a new predicate sub/2 that represents a subterm relation
Order of applying those transformations is shifting -¿ range restricting -¿ blocking

3.1.2 Simplified transformations

This part is concerned about discussing the simplifications that were added to the original transformations highlighted in 3.1.1.
Keeping in mind the definition of EPR as mentioned here in 2.1.4.
So the following were made to each of the procedures:
1- Every step that only deals with proper function symbols is removed since they are not existing in EPR.
2- Any step or procedure that was introduced because of the existence of problems because of proper function symbols were removed as well. Ex.: pf, sh, and bl.

Therefore the resultant simplified procedures are the following:
For crr:
(0) Initialization. Initially, let crr(M) := M.
(1) Add a constant. Let dom be a fresh unary predicate symbol not in P , and let c be some constant. Extend crr(M) by the clause dom(c) . (The constant c can be fresh or belong to f .)
(2) Range-restriction. For each clause H B in crr(M), let x1 , . . . , xk be the set of variables occurring in H but not in B . Replace H B by the clause H B dom(x1) dom(xk).

For rr:
(0) Initialization. Initially, let rr(M) := M.
(1) Add a constant. Same as Step (1) in the definition of crr.
(2) Domain elements from clause bodies. For each clause H B in M and each atom P(t1 , . . . ,tn) from B , let P(s1 , . . . , sn) be the term abstraction of P(t1 , . . . ,tn) and let be the corresponding abstraction substitution. Extend rr(M) by the set dom(xi) P(s1 , . . . , sn) — 1 i n and xi ti .
(3) Range-restriction. Same as Step (2) in the definition of crr.
(4) Domain elements from P . For each n-ary P in p , extend rr(M) by the set dom(xi) P(x1 , . . . , xn) — i i n.

3.2 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 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.

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

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 [3], 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
EPR	Effectively propositional calculus
PNF	Prenex normal form
SNF	Skolem normal form
CNF	Clausal normal form

List of Figures

Appendix B

Forms of first order logic formulas

Different forms of first order logic formulas.

Appendix C

Algorithms

Different Algorithms used.

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] P. Baumgartner and R.A. Schmidt. Blocking and Other Enhancements for Bottom-Up Model Generation Methods. 2006.

[3] Christopher Lynch. Constructing Bachmair-Ganzinger Models. 7797, 2013.

[4] Stephan Schulz. System Description: E 1.8. 2013.