Dynamic Semantic Resolution

Yunshi Sun
u5783028

Supervisor:
Prof. John Slaney

A project report submitted in partial fulfilment of
the requirement for the course Computing Project (COMP8715) at
The Research School of Computer Science
Australian National University

Oct 28, 2016
Abstract

In automated theorem proving, resolution is a standard technique of inferencing. The idea of semantic resolution is to use semantic information, i.e. interpretation or model, to guide the proof search. In this project we use the very old idea of dynamic semantic resolution. That is letting theorem prover guided by changing model discovered at runtime. Since interpretation is updated with the information we get proof search progresses, it tends to fit the proof search closely. This report talks about the algorithm and concept behind the theorem prover. It also describes the way how we implement the program basing on existing components.
Declaration

This project is conducted from July 2016 to October 2016 in COMP8715 Computing Project for the degree of Master of Computing in Research School of Computer Science in ANU.

Except where otherwise indicated, I hereby declare that the work presented in this project is solely my original work and has not been previously submitted either in whole or in part for a degree in any university.

Yunshi Sun
u5783028
Oct 28, 2016
Australian National University
Canberra, ACT, Australia
Acknowledgements

I would like to express my deepest gratitude to my project supervisor, Prof. John Slaney, for his patient instructions and professional suggestions for this entire semester. Thank you for encouraging me and guiding me through the entire project.

I would also like to thank the project convener Prof. Weifa Liang. Thank you for your effort for organizing everything well and your advice on our presentation and writing report.

Special thanks to the Australian National University and the College of Engineering and Computer Science for giving me this opportunity.
# Contents

1 Introduction ......................................................... 1
   1.1 Background .................................................................................................................. 1
   1.2 Basic Concept ............................................................................................................. 2
   1.3 Motivation .................................................................................................................... 3
   1.4 Project Objectives ........................................................................................................ 5

2 Dynamic Semantic Resolution ........................................... 6
   2.1 Good model vs Bad model ............................................................................................ 6
   2.2 What is Dynamic Semantic Resolution ......................................................................... 6
   2.3 Algorithm of Dynamic Semantic Resolution .................................................................. 6

3 Implementation .............................................................. 9
   3.1 Prover9 and Finder ...................................................................................................... 9
   3.2 Structure ...................................................................................................................... 10
   3.3 Current Work .............................................................................................................. 10

4 Summary ............................................................................. 31
   4.1 Conclusion .................................................................................................................. 31
   4.2 Future Work ................................................................................................................. 31

Bibliography ........................................................................ 32

Appendix
   A. Independent Study Contract ......................................................................................... 32
List of Figures

1. Entire search space using binary resolution..............................3
2. Entire search space using semantic resolution..........................4
3. Effect with bad model..........................................................6
4. Graph representation of semantic resolution............................
5. Structure of the theorem prover.............................................6
Chapter 1

Introduction

1.1 Background

This project falls in the field of automated theorem proving. Automated theorem proving is an area of study to get computers to prove logical and mathematical statements. The computer programs are usually called the automated theorem provers. They have been widely used in a variety of domains. Especially in the areas of logic, mathematics proving and software design and verification.

The most exciting recent success in mathematics has been the settling of the Robbins problem by the automated theorem prover called EQP. In 1933 Herbert Robbins conjectured that a particular group of axioms form a basis for Boolean algebra, but no one (until the solution by EQP) could prove this. The proof that confirms that Robbins' axioms are a basis for Boolean algebra was found October 10, 1996, after about 8 days of search by EQP, on an RS/6000 processor ("An Overview of Automated Theorem Proving", 2016). The result was also reported by the New York Times.

Given a problem expressed in a formal logic form, the automated theorem prover is able to generate proof automatically by applying logical deduction or using rule of inference. In theory, it is capable of solving any problem that expressed in a formal logic form. But in practice, the theorem provers are limited by time and memory constraints and sometimes they are not especially smart.
1.2 Basic concept

Before we go further, I want to clarify some terms and concept associated with automated theorem proving. You might see them been used very often in this report. It will be easy to look up to put them all in this section.

- We say that theorem provers are able to solve problem expressed in a formal logic form. **Propositional logic** and **first order logic** are two very basic and classic branches of formal logic. That’s also the area my project focusses on.

- In a logic system, a **literal** is a statement that is either true or false. In first order logic a literal is equivalent to a predicate, i.e. $p, q, not r$. In propositional logic it is an atomic proposition, i.e. $\exists x f(x), \forall y g(y)$.

- A **clause** is made up of literals in a disjunctive form, i.e. $p | q | r, \exists x f(x) | \forall y g(y)$.

- In automated theorem proving, when we want to find a proof, the theorem prover usually forms some new clauses from existing clauses by using inferencing rules. **Resolution** is one of the inference rules first proposed by John Alan Robinson in 1965. It works by proving the negation of the goal implies a contradiction.

- In theory, resolution is **sound** and **complete**. This means it can prove any theorem that is true and will not prove any theorem that is false.

- The clause produced by a resolution rule is sometimes called a **resolvent**.

- **Binary resolution** is the simplest form of resolution. It takes two clauses with negated literals, clashes away the negated literals and combine the rest literals to form a new clause.

  \[
  \begin{array}{c}
  A \lor B \lor D \quad C \lor \neg A \lor E \\
  \hline
  B \lor D \lor C \lor E
  \end{array}
  \]

  i.e.
1.3 Motivation

Most modern theorem provers use a purely syntactic method to generating proofs. A syntactic method means that we do not care about the meaning of each symbol when we are finding proofs. The theorem prover doesn’t really understand what it is proving and simply treat input clauses as meaningless symbols. Binary resolution is one of the inference rules uses syntactic method.

Considering the problem set \{ p(x) \mid q(x), \neg p(x), \neg q(a) \mid \neg q(b) \}. When we try to solve it using binary resolution, we pick a pair of clauses from the problem set, clash away the negated literals and add the new clause to the problem set. We keep doing it until we reach an empty clause (which means contradiction).

![Figure 1. Entire search space using binary resolution ("Semantic Resolution Example", n.d.).](image)

The picture above shows the whole search space of the problem using binary resolution. In the picture, each black dot represents a binary resolution inference. The black edges connect the parents to the inference and blue edges connect resolvent to the inference. There are 12 clauses and 19 inferences in the search space. We need 3 steps to find a proof.
There is also another method using semantic information to guide the search and we call it semantic method. Comparing to syntactic method, semantic method is “smarter” since it has an idea of what the clauses mean. It is able to choose whether certain clause can be generated in the proof by its meaning.

Figure 2. Entire search space using semantic methods (“Semantic Resolution Example”, n.d.).

For the same problem, the picture above shows the search space using semantic methods with one “good” model (We’ll talk about how good means later). As we can see, 5 of the 12 clauses in the search space of binary resolution is unreachable. It reduces almost half of the search space. Even it doesn’t reduce the number of steps to the goal, it shows us the right direction which leads the proof straight to the goal.

From this example, we have seen the power of semantic methods to reduce the search space almost to triviality. As we said in the previous section, theorem provers are limited by time and memory constraints. If we can find a good model to guide our search, it will save us a lot of time and memory. That motivate us to having the semantic guidance in our theorem prover.
1.4 Project Objectives

In this project, we’ll take the idea of letting resolution be guided by semantic information, i.e. interpretation or model, implement a version of program using existing components to do the reasoning.

The project objectives are specified as below:

- Understand the structure and algorithm of the pre-existing components.
- An implementation of dynamic semantic resolution basing on pre-existing components.
- There are no special requirements on the programming language, but it should be efficient considering running time and memory.
- Test and research the effect on reduction of search space when selecting different models.
- Test the method against different sets of problem from the TPTP (Thousands of Problems for Theorem Provers) library.
Chapter 2

Dynamic Semantic Resolution

2.1 Good model vs Bad model.

In previous section, we have seen the power of semantic methods to reduce search space of a problem. In that example, we compare the search space using binary resolution with the one using semantic resolution. We also mentioned that the model is good but didn’t explain why. In this section, we’ll compare results with different models and explain how we are going to define good model in our project.

Recall the problem we see in section 1.3. We are given the problem set problem set \{ p(x) \mid q(x), \neg p(x), \neg q(a) \mid \neg q(b) \} and try to prove it by reaching an empty clause (which means contradiction). Since there are many models can be selected to guide our proof search. Do all of them have the effect to reduce search space and lead us to the goal? The picture below shows the effect of another model for the same problem.

Figure 3. Effect with bad model ("Semantic Resolution Example", n.d.).
Comparing to figure 1 and figure 2 in section 1.3, the effect of this model is very small. All clauses remain reachable in the same way as they are in the result of binary resolution. Only one inference has been blocked. The effect of this model is negligible. Considering the fact, the cost of finding a model and checking truth value of each clause in the model is very expensive, we’ll possibly use much more time to find the goal we want than using binary resolution.

Since the problem size is very small, we can examine all possible models for this problems. It turns out the model we use in figure 2 is the best one which has blocked most of the search space and leads us to the goal directly. But is it possible to know which one is good and which one is bad before we examine them all? By far, there is still no answer for this problem. We have to balance the cost and effect of it to make best use of the semantic information. We hope what we have done in this project would be some help to get the answer.

In this project, we define the good model, which is the one we will use to guide our proof search, as the model satisfied most of the clauses. Sometimes there might be no such model to satisfy all clauses we have, so we will discard some of them when generating model. But we think it will still lead us to the goal with the right direction, as long as it satisfies most of the clauses.
2.2 What is Dynamic semantic resolution?

We finally come to our project topic which is Dynamic Semantic Resolution (DSR). In previous sections, I have explained what is resolution and how semantic methods will be good for our theorem prover. The only thing left is “dynamic”.

As we mentioned, resolution was first proposed by John Alan Robinson in 1965 (Robinson, 1965). Later he published an improved version of resolution algorithm which is called hyper-resolution. Unlike binary resolution, hyper-resolution is able to deal with more than two clauses at the same time which shortens the proof length and steps. In 1967, Slagle showed that hyper-resolution is a special case of semantic resolution (Slagle, 1967). That is using semantic information to guide the inference steps. When using semantic resolution, the theorem prover is no longer blindly manipulating symbols as it does with binary resolution; instead it has some concept of their meaning (Simon, 2003).

In this project, we are going to build our theorem prover basing on Slagle’s theory of semantic resolution. But to make sure we are always guided by the best model, we’ll update the model when we have certain amount of new information from the theorem prover. So that we believe the guidance provided will be more closely adapted to the search development. The idea was proposed by John Slaney in 1993 (Slaney, 1993). He named the theorem prover as SCOTT. In this project, we will follow his idea and implement DSR efficiently basing on existing components.

The reason we call the project as DSR is because the theorem prover is based on theory of semantic resolution and the guiding model is dynamically updated during run time.
2.3 Algorithm of Semantic Resolution

We have introduced resolution and some variants of it, including binary resolution and hyper resolution, in previous sections. Here we will talk about the algorithm we use to realize semantic resolution.

Semantic resolution was first introduced by Slagle in 1967. According to his paper, semantic resolution might more appropriately be called semantic hyper-resolution. We generate proof by using hyper resolution under semantic constraints. As shown in the picture below, at each inference step, we pick one clause called the nucleus and an arbitrary number of other clauses called the satellites. We clash one literal in each satellite with a complementary literal in the nucleus. We also need to make sure no literal in the nucleus has been clashed more than once.

Figure 4. Graph representation of semantic resolution.
All other literals who do not take part in the clash will form the new clause called hyper-resolvent. Some techniques have been used like factoring and unifying during the inference. We not going to explain details of all these techniques since they are focus of the project. The semantic constrain is that all satellites and the hyper-resolvent must be false in the guiding model. That’s how use semantic information to guide our search. If the hyper-resolvent is true, we’ll not include it in the search space, and hence it might block some redundant clauses which has no help for the proof. This explained why some clauses are unreachable in Figure 2 and that’s what we really want.

In our project, we are not going to do the hyper resolution directly. We will emulate hyper resolution inference by using binary resolution under semantic constraints. The input of the inference is the same, one nucleus and a number of satellite. Each time we pick one satellite and clash it with the nucleus. This result is neither a satellite or a nucleus, and we need to re-define the inference for them. We call these intermedia clauses as nuclei, but only the literals inherited from the nucleus parent can be clashed with other clauses. Here we also require the nuclei follow the semantic constrains. When there is no satellite can be found to clash with the nuclei, we finish one hyper resolution inference.
Chapter 3

Implementation

3.1 Prover9 and Finder

The implementation of DSR is based on two pre-existing programs. One of them is the theorem prover called Prover9. The other one is the model generator called Finder.

Prover9 is an automated theorem prover for first-order and equational logic developed by William McCune. It is worth to mention that William McCune is also the creator of the EQP theorem prover which solve the Robbins conjecture. The primary rule of inference used by Prover9 is resolution. It repeatedly makes resolution inferences until deriving a contradiction. Sadly, prover9 is no longer updated since 2009. The origin version of Prover9 is not 100% fit our requirement. So the project including making some changes to Prover9 to transform it to the way we want. Considering it has more than 20,000 lines of C code, reverse-engineering it is not an easy task.

Finder is a model generator developed by John Slaney. It takes as input a first order theory, expressed as a set of clauses, and gives as output the models of that theory with domains of given finite cardinality (Finder Page, n.d.). In our project we use it to generate and update guiding model for the theorem prover.

At first we planned to use another model generator called Mace4 which is closely bundled with Prover9. Both of them have the same syntax and are included in the same library. The reason we choose Finder instead of Mace4 is that Finder is able to set a parameter which allows the number of constraints the model can violate. If you set the parameter to zero, it will find you the exactly model satisfying everything, otherwise it will use different strategy for backtracking to find the model with less number of unsatisfied constraints (It will still find the perfect model if exists). Mace4 cannot do that. Surely we can do another reverse-engineering on Mace4 to give it this ability, but time does not allow us to fix both of them in this project. However, we still use Mace4 in our project, but for different purpose. I’ll talk about it in later sections.
3.2 Structure

The overall structure of our theorem prover contains three components, two of them are Prover9 and Finder which we just introduced. The last part is what we called DSR and it is the main task of this project. It behaves like the coordinator between Prover9 and Finder. It calls each program in turn, analyzing the output from one and use the result to generate input for the other. The below diagram shows the data flow of the theorem prover.

At first, our program called DSR called the prover9 with problem clauses set. Prover9 will generating new clauses using the algorithm we discussed. After a number of inference steps (we can change the parameter) we pass the new clauses back to DSR to do processing. DSR will pass the processed clauses to the Finder for updating our guiding model. The updated model is passed back to DSR for processing, and then it will be the new model to guide our proof search in Prover9. The theorem prover and model generator inform each other dynamically during the search.

One of the tasks of DSR is to make sure Prover9 and Finder can understand each other. That is parsing the result from the Prover9-syntax to Finder-syntax and vice versa. Besides it also needs to extract information from the Prover9 output since Finder requires the input containing definition of all items in the vocabulary and sorts. Currently this is done with the help of Mace4 since it shares same syntax with Prover9.
3.3 Current work

This section was meant to show the results of the implementation. But due to a lot of reasons, the theorem prover is not completed by the time this report is written. So I’ll list what we have achieved by far and talk about what we need to complete in the next chapter.

- The DSR is written by Python. That’s simply because Python is my personal preference and I haven’t written any C programs before (not even a hello_world.c)

- The reverse-engineering on Prover9 is the one we finished first. The changes are made on the source code of Prover9. So clearly speaking, we are using a modified version of Prover9 in our theorem prover. Most of the changes are done my supervisor John (Thank you!). With the modification we are able to emulate semantic resolution by binary resolution using the algorithm previously discussed.

- Functions to parse input and output between Finder-syntax and Prover9-syntax have been finished by myself. This is implemented is the DSR part. A temporary file will be created to store the intermediary results when DSR running.

- Currently we are struggling at the function to extract information from the Prover9 output. We need to get the vocabulary and sorts from the output and use it as the input for Finder. According to the plan, we should get this done with the help of Mace4. But the modifications we made on Prover9 have caused some bugs to Mace4 as use the same public library.

- Other than the function of extracting vocabulary and sorts, all other parts and functions of DSR haven been finished. That includes functions calling each program we have, getting the results of them and other processing functions.

- We also add a function to try to solve the input problem using Prover9 within a time limit. If we can’t get a proof within the time limit, we’ll start to use semantic resolution. This is because Prover9 has a algorithm to select the clauses to do binary resolution so that sometimes it doesn’t need to travel the entire search space and find the proof in a very short. This function can be turned off and we can start generating proof using dynamic semantic resolution directly.
Chapter 4

Summary

4.1 Conclusion

In this report, we have explained the algorithm to emulate semantic resolution by a number of steps of binary resolution. During that process, semantic information is guiding the proof by checking truth value of the clause in the model. This makes theorem prover smart and no longer blind. But this causes incompleteness for hyper-resolution and the like.

The implementation of DSR is well-designed. It bases on the high-performance theorem prover Prover9 and the model generator Finder, passes information and coordinate work between these two. Even though not enough testing has been done to see how good DSR performs, DSR has shown its potential to reduce the search space and fasten the speed to get the proof. We believe with the completion of DSR, we can get the desired result we want.
4.2 Future work

In this section, we describe some works need to be done in the future and some idea might improve the performance of the prover.

- One of the weirdest thing we met during this project was that we couldn’t get Prover9 compiled on lab machines of CSIT. Seems the problem is because the C compiler in Ubuntu can’t find correct reference to the math library. We worked on it for weeks but couldn’t solve it. We also looked for help online and from the IT Helpdesk but got no solution. At last, I have to borrow a MacBook to continue this project. If some other student takes this project and prefer to do it on the lab machine, this has to be fixed first.

- The urgent work is to finish the extracting function so that we can testing the performance of it comparing with Prover9 itself and other high-performance theorem prover.

- The effectiveness of semantic resolution relative to guiding model is still an open question. The way we define and select good model might not be the best criteria. Deeper research and experiments need to be done to answer this question.

- One good idea might be integrating DSR directly into the Prover9. It is better for Prover9 to talk directly to Finder instead of passing information by the middle man. This would also increase the performance of the theorem prover a bit as the procedure would be simplified.

- We also have the brave conjecture that use multiple model to guide the proof search, but this idea is at very early stage. Besides this idea is closely related to the effectiveness of model selection. It is better to leave this idea to the last.
Bibliography


INDEPENDENT STUDY CONTRACT

Note: Enrolment is subject to approval by the projects co-ordinator

SECTION A (Students and Supervisors)

UniID: 5783028
SURNNAME: SUN
FIRST NAMES: YUN SHI
PROJECT SUPERVISOR (may be external): JOHN SLANEY
COURSE SUPERVISOR (a RSCS academic):
COURSE CODE, TITLE AND UNIT: COMP0715 Computing Project: 12 units

SEMESTER [ ] S1 [ ] S2 YEAR: 2016

PROJECT TITLE: Dynamic Semantic Resolution

LEARNING OBJECTIVES: Understand mainstream automated reasoning methods.
Demonstrate ability to carry out research under supervision and present experimental results.

PROJECT DESCRIPTION:
The resolution algorithm is a standard technique for automated reasoning. Semantic resolution is a version in which an interpretation is used to guide the proof search. In dynamic semantic resolution, the interpretation is updated during the search, so that it tends to fit the specific problem. Dynamic semantic resolution has never been fully implemented and subjected to systematic experiment. In this project, we shall implement a version of the algorithm using existing components to do the reasoning and test the method against the standard TPTP benchmark sets for non-equational first order problems.
ASSESSMENT (as per course’s project rules web page, with the differences noted below):

<table>
<thead>
<tr>
<th>Assessed project components:</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report: name style: report</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g. research report, software description...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artefact: name kind: software</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g. software, user interface, robot...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation:</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MEETING DATES (IF KNOWN):

STUDENT DECLARATION: I agree to fulfill the above defined contract:

Yours sincerely

Signature

Date

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student’s academic record and believe this student can complete the project.

Signature

Date

REQUIRED DEPARTMENT RESOURCES:

SECTION C (Course coordinator approval)

Signature

Date

SECTION D (Projects coordinator approval)

Signature

Date

Research School of Computer Science

Form updated Jun-12