A Theorem Prover utilizing Natural Deduction

Bernardo Pinto de Alkmim

Proposta de Projeto Final de Graduação

Centro Técnico Científico
Departamento de Informatics
Curso de Computer Science

Orientador: Prof. Edward Hermann Haeusler

Rio de Janeiro
June de 2016

Bernardo Pinto de Alkmim

A Theorem Prover utilizing Natural Deduction

Proposta de Projeto Final, apresentada ao programa Bachelors in Computer Science da PUC-Rio como requisito parcial para a obtenção do Bachelors of Science in Computer Science.

Prof. Edward Hermann Haeusler
Orientador
Departamento de Informatics — PUC–Rio

Rio de Janeiro, 13 de June de 2016

XPTO

XPTO

XPTO, *XPTO.*

Agradecimentos

XPTO

Resumo

Alkmim, Bernardo; Haeusler, Edward. **A Theorem Prover utilizing Natural Deduction**. Rio de Janeiro, 2016. 26p. Proposta de Projeto Final — Departamento de Informatics, Pontificial Catholic University of Rio de Janeiro.

Em várias áreas da informática há demanda por métodos formais para validação de sistemas, objetivo para o qual existem diversas técnicas diferentes a ser aplicadas. A Dedução Natural é um dos procedimentos de provas de sentenças lógicas que busca aproximar-se do processo pelo qual um ser humano seguiria para realizar tal prova. Neste documento, será apresentada uma proposta de implementação de um Provador Automático de Teoremas na linguagem de programação Lua utilizando esse método de resolução, aplicando conhecimentos aprendidos nas áreas de Lógica, Teoria da Computação, Algoritmos e Engenharia de Software ao decorrer do curso de Bacharelado em Ciência da Computação. Esse projeto final tem por objetivo principal responder à pergunta: como a Dedução Natural pode auxiliar a Prova de Teoremas via o uso de computadores (em especial, via automatização das provas)?

Palavras–chave

Lua. Prova Automática de Teoremas. Lógica. Métodos Formais. Dedução Natural.

Abstract

Alkmim, Bernardo; Haeusler, Edward. **A Theorem Prover utilizing Natural Deduction**. Rio de Janeiro, 2016. 26p. Final Project Proposal — Department of Informatics, Pontificial Catholic University of Rio de Janeiro.

In several areas of Informatics there is a need for system validation via formal methods, in which several different techniques can be applied. Natural Deduction is one of the logic sentence proving methods which aims to approximate itself to the expected process by which a human being would think in order to construct such proof. In this document, will be presented a proposal of implementation of an Automatic Theorem Prover (ATP) in the Lua programming language utilizing this resolution method, applying knowledge acquired in the areas of Logic, Theory of Computation, Algorithms and Software Engineering while in the course of Bachelors of Science in Computer Science. This project aims mainly to answer the following question: how can Natural Deduction be utilized to implement a graph-based Automated Theorem Prover?

Keywords

Lua. Automated Theorem Proving. ATP. Logic. Formal Methods. Natural Deduction.

Sumário

Lista de figuras

1 Introduction

In this chapter I will introduce the context of the problem to be solved, as well as an explanation as to why it is a problem and its relevance.

1.1 Context

Since Computer Science is a relatively new field (especially when compared to other STEM fields), it is in a constant and accelerated process of maturing. Patterned methods and techniques arise frequently in areas related to Software Engineering due to an increasing need to validate the systems being produced, particularly distributed systems and other which have harder to predict behavior. In order to have a truly formal validation (something that is no widely utilized nowadays in Informatics, specially with agile methodologies and the need of constant and fast delivery of working software) of programs like these, just project patterns, unit testing, or similar techniques are not enough.

In order to validate systems, we make use of formal methods. Formal methods, as the name suggests, utilize mathematical knowledge to guarantee, at a theoretical level, behavior and execution patterns of systems. In practice, of course, there can always occur an external physical interference (or something alike) in any system, which is something not necessarily validated, but can be predicted via statistical analysis for quality assessment (which is a bit off-topic to the subject here discussed).

One example of an area that utilizes formal methods nowadays is Integrated Circuit Design. After a mistake having occurred, which ended up being known as *Pentium FDIV bug*[Hal95], that involved a failure in the floating point unit of old Intel Pentium processors, the usage of formal methods started to be applied to validate the functionality of arithmetic operations.

1.2 How to apply formal methods

Formal methods are applied to system validation via the use of Logic. The validation of a system ends up translated to one or more theorems modeled under a certain logical structure. Logical sentences are part of the daily lives of computer scientists, mathematicians, engineers, philosophers, specially those who deal with plenty of formal methods and have the need of demonstrating different kinds of theorems.

As it usually happens in Logic, demonstrations of theorems can be mostly (sometimes even completely) deterministic in nature. With this, demonstrating such theorems in logical sentences, being them in Classical, Intuitionistic, Proposicional, First Order etc. Logic, can become a long, tiring, and exponentially big process (something limited by the finite size of paper sheets, unfortunately), and basically mechanic, since it is divided into the application of a reasonably short set of simple rules (varying, of course, with the logical framework being utilized).

2 State of the Art

In this chapter I will explain what is being made in the area of theorem proving with computational aid, presenting different methodologies utilized. There will also be introduced the main focus of this project: Natural Deduction.

2.1 Proof assistants

Since solving theorems can become very tedious or error-prone, or even difficult to model, enter the so called proof assistants: with their aid, one can use computers to support the making and validation of the steps on a proof. In general, they work on a step-by-step basis, *i.e.* at every possible step of the demonstration to be given, the system gives the user a set of possible steps to follow in order to continue the demonstration (and, in some cases, the user has to type all the steps, which involves making the proof by hand before entering the input, and utilizing the proof assistant just to check and validate). In the end, one expects to have the demonstration finished in case everything goes well, or an error in the case of an incorrect step happening in the middle of the proof.

As there does not exist only one proof method in Logic (since there are many different logical frameworks), it is common for said assistants to come in many distinct *flavors*. For instance, an assistant can utilize *Tableaux* as a proving method, which utilizes *reductio ad absurdum*, or Truth Tables, which is a method by exhaustion (and is very limited, since, for instance, it does not even work for First Order Logic, for it would need infinite lines on a table). There are many others, but the main ones are: Sequent Calculus and Natural Deduction (which will be explained ahead, still in this chapter).

Despite accelerating the process, there is still unneeded mental effort of the researcher spent on proof that are, in general, secondary to their work. This ends, at times, shifting the focus from the main goal of said work.

Still, there is more: theorems are not limited to just three or four atomic logical sentences, as those who have had an introduction to logic can attest. Depending on the problem in question, specially for First Order Logic and others that introduce variables, the number of possible sentences can explode exponentially.

The ideal would be for a system that made this entire mechanical part by itself, without the need of user influence (aside from inserting the logical formula to be demonstrated, of course). With this, we would have a rather tiresome part being made by the machine, thus saving time and effort from the user and facilitating the process of software validation.

2.2 Automated Theorem Provers

Out of this need arise the Automated Theorem Provers (ATPs). They have exactly this purpose in mind: to demonstrate logical sentences (whilst having the secondary goal of showing the steps taken on the demonstration), as to make the work of the end user easier.

Just like the proof assistants, the ATPs need a logical framework over which to function. Some popular techniques utilized in ATPs are (as seen on [Wir06], [Chl11], [BP15], and [Sut09]):

– First Order Resolution

  With a focus on First Order Logic, this method is based on utilizing an inference rule called *resolution* iteratively over the formulae. Resolution is a simple rule that receives two formulae (always based on the $\vee$ operator) and has a third one (that also makes use of $\vee$) in the conclusion. Since there is only one logical operator, formulae tend to be really convoluted and have bad legibility.

– *Tableaux*

  Tableaux is a demonstration method based on *reductio ad absurdum*, trying to falsify the theorem received as input. If, for all possible interpretations, one reaches contradictions, one can conclude then that the theorem is valid, and the proof is the tree of the *Tableaux*.

– Model Checking

  With Model Checking, one aims to verify if a model is of a certain specification via exhaustive checking.

– Mathematical induction

  Proofs that utilize induction are based on two parts: base cases and an inductive hypothesis (which is the final objective of the demonstration, being reached via repeated application of inductive steps on the base cases).

– DPLL Algorithm

This algorithm aims to decide the satisfiability of logical propositions via graph on the graph that represents the syntax of the proposition to be verified.

In this work, will be utilized Natural Deduction (*ND*), to be explained ahead, still in this chapter.

Since this work is focused on an ATP, internal implementation details will be discusses in future chapters.

2.3 Natural Deduction

One of the points of ND is to get closer to the way an average human being would choose to solve a problem. Introduced as an alternative to Hilbert systems, it can be explained in a rather intuitive way for it deals with concepts that can seem obvious *a priori* for someone with a Logic background.

A logical sentence in ND takes the form of the *introduction* or *elimination* of a logical quantifier. The introduction of quantifiers consists of having one or more premises and generating a logical operator in the conclusion of the sentence. The elimination does the opposite, eliminating a logical operator on the premises generating a *smaller* conclusion, *per se*. Concatenating several of there steps, we can demonstrate theorems.

ND can be represented mainly through Gentzen Trees (which will be the representation method chosen for this work, since we do not have the problem of the limited space of a paper sheet) or Fitch Formatting (which is more linear and vertical, but can get convoluted). The Fitch Format can be seen in [Ind]. Here are a few examples of sentences in ND, according to the notation in [Hh08]:

- $\wedge$-Introduction

$$\frac{A \qquad B}{A \wedge B} \tag{2-1}$$

  In the $\wedge$-Introduction, we have two premises, A e B. Since A is true and B is true, we can conclude that $A \wedge B$ is true (*i. e.* we *introduced* the $\wedge$ quantifier).

- $\rightarrow$-Elimination (aka *Modus Ponens*)

$$\frac{A \qquad (A \rightarrow B)}{B} \tag{2-2}$$

  In the $\rightarrow$-Elimination, $A \rightarrow B$ is true, and we have A true as well. With this, it is clear that B is true (thus *eliminating* the $\rightarrow$ quantifier).

In ND for Propositional Logic there are introduction and elimination rules for the $\wedge$, $\rightarrow$, $\vee$, $\neg$ quantifiers, as well as a rule for the absurd ($\perp$). This rule for the absurd has, in fact, one version for Classical Logic and another for Intuitionistic Logic (in Intuitionistic Logic one cannot infer A true from $\neg\neg A$ true), showing even more how we need different types of ATPs, of at least ATPs with different behaviors according to different logical frameworkds.

For First Order Logic, besides the aforementioned rules, there are also introduction and elimination rules for $\forall$ and $\exists$.

There is another important concept that increases the legibility of ND: the *hypothesis discharge*. A discharge happens when a proof step needs another hypothesis that is not directly present in the premises (which means it needs a *global* hypothesis to this proof, such as an initial premise, that automatically becomes *discharge-able*, and can be utilized anywhere in the proof). This mechanism is better explained via an example:

– $\vee$-Elimination

$$\frac{(A \vee B) \qquad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C} \tag{2-3}$$

In the $\vee$-Elimination, we have $A \vee B$, which contains the quantifier to be eliminated and two other premises, both C. But only this is not enough for the elimination to occur. It only makes sense when C depends from either A or B. Now enter the hypotheses discharges. If, in one case, from A we can deduce C (C depends of A) and, on the other, we deduce it from B, we can then conclude what, either from A or from B we can get to C. Then, the elimination can occur.

2.4 Why Natural Deduction?

Natural Deduction aims to be a deductive system that has a more natural approach to how logic is dealt with, closer to how a human being faces logical reasoning, as previously said.

Instead of having one *magical* rule applied several times, natural deduction aims to have a complete demonstration as something readable and of easy (*-ier*) comprehension. Let us understand *easy comprehension* here for someone in the area of Logic. In other words, ND does not involve concepts that are way too abstract or specific, whilst looking at the same time for readability. This is something that other methods end up not living up to in one way or another: they either involve too specific concepts, or end up having a barely legible demonstration.

One such example is Sequent Calculus: it involves really complicated concepts to produce an extremely efficient, but not so legible proof. It is, in fact, the most utilized method to make ATPs, but the proofs generated by this method are often treated as black boxes, since reading them requires pen and paper on the side. Natural Deduction proofs are locally self contained (with the exception of hypothesis discharges, which are always indexed to facilitate cross referencing).

3
Systematic Mapping

In this chapter will be presented the Systematic Mapping results in order to make a Literature Review and to better define the characteristics of the ATP to be implemented.

3.1
Search filters and final results

In order to make a Literature Review, were defined a few parameters: Search Sources, Search Terms, and Main Filters.

3.1.1
Search Sources

The sources utilized were:

– Google Scholar

– IEEE

– ACM

3.1.2
Search Terms

The terms utilized were:

– ATP

– automated theorem proving

– natural deduction

– logic

3.1.3 Main Filters

The main criteria were:

- Papers published in the last 10 years
- Papers with over 4 citations
- Papers which main focus was ATP development.

Those which did not fulfill these criteria were excluded.

3.1.4 The selected papers

In the end, 14 papers lasted, 8 from Google Scholar, 1 from IEEE, and 5 from ACM.

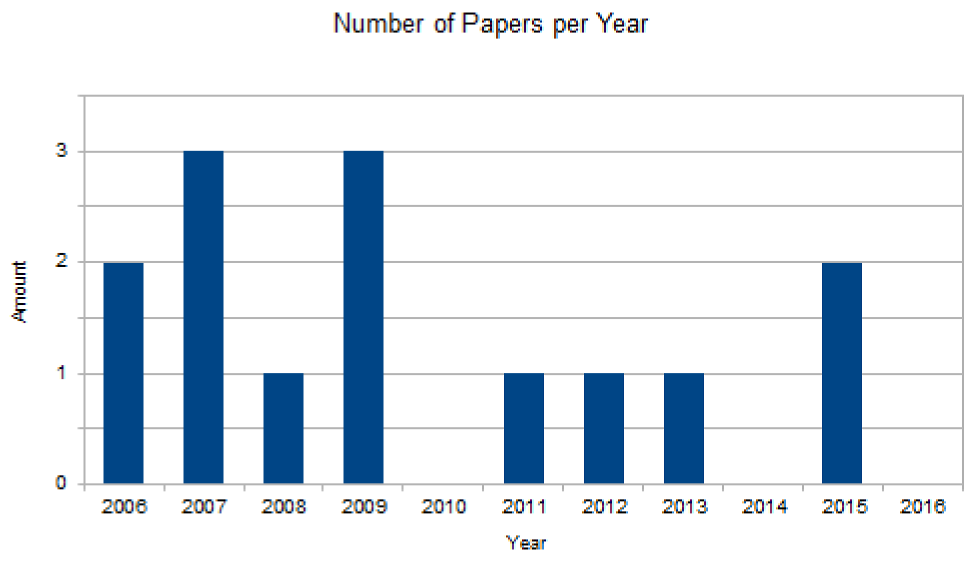

Figura 3.1: Found papers listed by year

3.2 Main findings

Upon evaluation of the found papers, a lot was found, both in graph-based ATPs (specially [Bla09], [Sul12], [BN15], and [Chl11]), and in ATPs based on Natural Deduction (specially [Bol07] and [Sha09]), but none had a graph-based approach to an ATP utilizing Natural Deduction. In fact, 9 were about non-ND based ATPs, and 5 were about ND ATPs.

It is somewhat expected, since it is intuitive to try to implement a tree-based ND ATP, and a graph-based would not be so much better. Here enter the concept of repeated formulae, which comes from the hypothesis discharge characteristic of ND: some sentences can appear many times in a same proof, occupying unnecessary memory. With a graph-based approach, there repeated formulae would be different pointers to the same logical formula, optimizing the usage of memory. In the upcoming chapters will be introduced a plan on how to see how relevant this problem is, thus evaluating the relevance of this niche.

4 Proposal and Objectives

In this section, we define the main goal of the project, sub-goals, the programming language utilized, the method of internal representation of logical sentences, and on what existing ATP it will be based on.

4.1 Research question and sub-questions

Research question: *How to make a graph-based Automated Theorem Prover utilizing Natural Deduction?*

In order to reach this main goal, we will need to reach a few sub-goals:

1. How to implement the ATP focusing on Propositional Intuitionistic Logic?

   With Propositional Logic, we can then guarantee a completely functional ATP, although limited.

2. How to expand it to Classical Propositional Logic, allowing the user to choose between Classical and Intuitionistic?

3. How to extend it for Intuitionistic First Order Logic?

   Having an ATP focused in Propositional Logic, to extend it to a First Order Logic one involves adding a few deduction rules, without drastic changes.

4. In the end, how to adapt it for First Order Classical Logic?

4.2 Implementation details

As seen on the previous chapter, it will be implemented with a graph-based approach in mind, trying to reduce the size of proofs by removing redundancies on the proof tree. The programming language utilized will be Lua. It will also have a simple interface to the end user by the usage of the Löve framework[Mic]. The ATP will be based on an already existent ATP from TecMF[Inf] (Laboratório de Tecnologia em Métodos Formais), a laboratory from the Department of Informatics from PUC-Rio, under the direction of professor Edward Hermann Haeusler. Said

ATP is graph-based as well, but utilizes Sequent Calculus instead of Natural Deduction.

4.3 Why Lua?

A few reasons make Lua the best candidate for this project:

1. Lua intends to be a small language, easy to comprehend and extend.
2. Has international visibility and a big community.
3. Lua was created at PUC-Rio.
4. It already has a known graphical interface framework.
5. The existing ATP from TecMF was implemented in Lua as well.

5
Quantitative Research

In this chapter I will present a proposal of a quantitative research to be made with the development of this ATP in mind. The plan is to assert that proofs in ND utilize less space than those in other methods, especially when one has the repetition of logical sentences in mind.

5.1
Hypotheses

Here are the hypotheses of our quantitative research:

- H_0 (or null hypothesis)

  Natural deduction does not offer a reduction of necessary space in logical demonstrations, even when repetitions are accounted for.

- H_1 (directional hypothesis)

  The use of natural deduction (especially when removing repetitions) shortens the size of proofs significantly in relation to other proof methods.

5.2
Defining some terms

In order to continue with the quantitative research some terms will be needed to be defined:

- Size of a proof

  The size of a proof is an integer that represents the sum of all the logical sentences on a proof. A simple $\wedge$-introduction, for instance, has a size of 3.

- Repetition of a formula

  If the same formula appears more than once in the whole proof, it can be counted as repeated. Note: for hypothesis discharges, one can only be repeated if it has the same label (which means it was generated by the same deduction step).

5.3 Parts of the research

This quantitative research will be divided into three parts:

1. Creating proofs to be viewed in ND and other deduction methods.
2. Comparing their sizes.
3. Removing redundant nodes from the ND proofs.
4. Comparing their new size to those of the other deduction methods (including ND itself).

After this, we will have a solid basis on which to stand regarding our intent to make a graph-based ATP with ND.

6 Qualitative Research

In this chapter I will present a proposal of a qualitative research to be made with the development of this ATP in mind. The plan is to assert that the ATP will have characteristics that are missing in other ATPs, according to their end users: mathematicians, logisticians, computer scientists, engineers, philosophers etc., trying, in the end, to answer the question: what features should an automated theorem prover have?

6.1 Overview

The qualitative research will be divided into two main sections: an interview with the end users and an observation of their usage of their current available ATPs (the one they use daily).

6.2 Interview

The questions for the interview are as follows:

1. When in your work do you need to prove theorems? What is the frequency of occurrence of theorems in need of proving?
2. How often do you use ATPs for that?
3. How "big"would you say the proofs usually are? (Big in terms of number of atomic logical sentences.)
4. On what logical frameworks do you usually use ATPs for? Propositional, modal, first-order logic etc.?
5. What is the relevance of those theorems you need to prove in relation to your main work?
6. Do you work in research or for a company?
7. When you were studying as an undergraduate (if applicable), did you ever feel the need of an ATP?

8. What have you felt was missing from the ATPs you have used?
9. Did an ATP you use ever missed a crucial feature you needed? Which one was it? Did you settle for another?
10. Would interactive visual representations of your proof be of any use? Do you have any in mind?
11. The ATPs you have used were based on which proof methods?
12. Did you find the proofs of ATPs you have used easy to follow?
13. What has been proven to be more important to you in an ATP: speed or memory usage?

6.3 Observation

The interviewees will, then, be asked to utilize the ATP they mainly use in the way they utilize it daily, in order to further check where do they struggle and how they overcome said struggles, so that this may end up not making the same mistakes as others.

7
Final considerations

The field of Automated Theorem Proving is growing with the advent of bigger and more distributed systems, and ends up appearing in areas of Informatics where it was once ignored. Natural Deduction is a great method to utilize in an ATP because it tries to shorten the gap between people in the area of Logic and *outsiders* from other areas in STEM fields, thus making ATPs more accessible. We intend to follow this line of thought for this work.

Bibliografia

[Hal95] Tom R. Halfhill. *An error in a lookup table created the infamous bug in Intel's latest processor, http://www.byte.com:80/art/9503/sec13/art1.htm.* Acessado em 2016/3/31., 1995.

[Hh08] Edward Haeusler e Paulo Blauth Menezes. *Teoria das Categorias para Ciência da Computação*. Bookman, 2 ed., 2008.

[Wir06] C. Wirth. *Progress in Computer-Assisted Inductive Theorem Proving by Human-Orientedness and Descente Infinie?* 2006.

[Bla09] J. et al Blanchette. *Hammering towards QED*. 2009.

[Sul12] N. Sultana. *LEO-II and Satallax on the Sledgehammer test bench.* 2012.

[BN15] S. Bhme e T. Nipkow. *Sledgehammer: Judgement Day*. 2015.

[Sut09] G. Sutcliffe. *The 4th IJCAR Automated Theorem Proving System Competition CASC-J4*. 2009.

[Ind] Andrzej Indrzejczak. *Natural Deduction, www.pims.math.ca/knotplot/.* Acessado em 2016/03/31.

[Mic] Anders Ruud e Michael Enger e Tommy Nguyen. *Löve, https://love2d.org/*. Acessado em 2016/3/31.

[Chl11] A. Chlipala. *Mostly-Automated Verification of Low-Level Programs in Computational Separation Logic*. 2011.

[BP15] C. Benzmller e B. Paleo. *Higher-Order Modal Logics: Automation and Applications*. 2015.

[Bol07] A. et al Bolotov. *Automated Natural Deduction for Propositional Linear-time Temporal Logic*. 2007.

[Sha09] N. Shankar. *Automated deduction for verification*. 2009.

[Inf] Formal Methods Group at Informatics Departament/PUC-Rio. *http://www.tecmf.inf.puc-rio.br/*. Acessado em 2016/3/31.