Automated Theorem Proving

by
Jagadish Bapanapally

1 Abstract

Logic can be defined as the formal study of reasoning; if we replace "formal" by "mechanical" we can place almost the entire set of methodologies used in the field of automated theorem proving (ATP) within the scope of logic. The Automated Theorem proving has several applications which has made researchers to build various Automated Theorem Proving systems which are applied in different ways for variuos purposes. In this paper I would like to do Literature review on Theorem Proving and various Automated Theorem Proving systems and their applications.

2 Introduction

Logic is a science studying the principles of reasoning and valid inference. Automated deduction is concerned with the mechanization of formal reasoning,following the laws of logic. The roots of the field go back to the end of the last century when Frege developed his Begriffsschrift, the first comprehensive effort to develop a formal language suitable as a foundation for mathematics. Alas, Russell discovered a paradox which showed that Freges system was inconsistent, that is, the truth of every proposition can be derived in it. Russell then devised his own system based on a type theory and he and Whitehead demonstrated in the monumental Principia Mathematica how it can serve as a foundation of mathematics. Later, Hilbert developed a simpler alternative, the predicate calculus. Gentzens formulation of the predicate calculus in a system of natural deduction provides a major milestone for the field. In natural deduction, the meaning of each logical connective is explained via inference rules.

Gentzens work also contains an early consistency proof for a formal logical system. As a technical device he introduced the sequential calculus and showed that it derives the same theorems as natural deduction. The famous Hauptsatz establishes that all proofs in the sequential calculus can be found according to a simple strategy. It is immediately evident that there are many propositions which have no proof according to this strategy, thereby guaranteeing consistency of the system.

Most search strategies employed by automated deduction systems are either directly based on or can be derived from the sequential calculus. We can broadly classify procedures as either working backwards from the proposed theorem toward the axioms, or forward from the axioms toward the theorem. Among the backward searching procedures we find tableaux, connection methods, matrix methods and some forms of resolution. Among the forward searching procedures we find classical resolution and the inverse method. The prominence of resolution among these methods is no accident, since Robinsons paper represented a major leap forward in the state of the art. It is natural to expect that a combination of forward and backward search could improve the efficiency of theorem proving system. Such a combination, however, has been elusive up to now, due to the largely incompatible basic choices in design and implementation of the two kinds of search procedures.

This paper is centered around Theorem Proving, the Automated Theorem Proving Systems and implementation of a succession of theorem provers for Automated Theorem Proving Systems. We start with a theorem proving system, followed by ACL2 theorem prover and how this can be applied to prove Rectifiable arcs theorem. The goal of the paper is to give a thorough understanding of the central techniques in automated theorem proving.

3 Theorem Proving

There are various methods that are being opted by the scientists to prove a Mathematical Theorem. The problem has been tackled in different ways by various scientists and one of them being Automated Theorem proving. One approach starts with a random interconnection of many model nerve cells. A second approach is based on a psychological rather than a neurological approach to the thought process. A third approach is based on the logic rather than either psychology or neurology. So this tells us the early stages of invention to Automated Theorem provers.[21]

Different Researchers cover different levels of concepts in Theorem proving. A logic is defined to be a vocabulary and formation rules and which provides a definition of proof by inferring these rules. A computer based Theorem Prover must construct a proof which contains a sequence of formulas or logics which are defined in this system. There are several researchers which are providing these prrofs in different ways as explained. But if we are trying to prove this in a system we must establish that rules or arguments that we are using to prove a theorem must also be proved by using same system.[19] explains the various areas that is concentrated by researchers in automated theorem proving. It explains the basic principles of automated theorem proving what it needs to do for theorem proving. It explains in a philosophical way in which a theorem prover must be working.[19]

There are proposes where a high level logical system which can be implemented on computers to prove theorems. The semantics of their system and basic definitions of symbolic logic has been presented which uses various heuristic methods which will be able to prove theorems using the symbolic logic. The machine lists an information processing system that is able to discover, using heuristic methods, proofs for theorems in symbolic logic. It was initially confined only to description, and have not attempted to generalize in abstract form about complex information processing. Because of the nature of the description, involving considerable rigor and detail, it may be useful to set out the main features of LT, especially as these appear to deflect basic characteristics of complex systems.

First of all, Logic Theory can be specified at all only because its structure is basically hierarchical, and makes repeated use of both iteration and recursion. So true is this, that one of Logic Theories main features, the use of a problem-sub problem hierarchy, is hardly visible in the program at all.

This system offers no guarantee of finding a proof; the other hand, it brings to its task a number different heuristic methods for achieving its goals. All of these methods are important in making LT sufficiently powerful to find proofs in most cases, and to-find them with a reasonable amount of computation, but not all of them are essential. Without chaining, for instances, LT could still function.[18]

Some workers in automatic theorem proving, including the authors, believe that it will be many years (if ever) before machines alone can prove difficult theorems in mathematics. Thus some, who hope to see machines used as practical assistants to pure mathematicians, have redirected their attention to man-machine theorem provers and theorem proof checking. Thus a Man-Machine theorem proving system has been developed at University of Texas. This explains about a theorem prover which doesn't prove theorems automatically but it takes help of a person in intermediate steps whenever it is needed.It also gives examples of the proofs that were proved by this system.

This system is organized in the same general way as those of Guard, Allen and Luckham, and Huet, but with many major differences.For example, Allen and Luckham, and Huet use variations of RESOLUTION as their principal rules of inference, whereas this system uses a modified form of IMPLY, which is a natural-deduction-type method. Also this system displays formulas on the scope in a natural, easy to read, manner and has available a variety of interactive commands the user can employ to help with the proof. Among these is a feature called DETAIL which allows the human to interact only when needed and only as much as is required for the proof.[2]

Fortran-Compiled List-Processing Language is a language that was developed on Fortran which is more convenient to program rather than using a symbolic expressions which was done till then. A compiled computer language for the manipulation of symbolic expressions organized in storage as lists has been developed as a tool to make more convenient the task of programming the simulation of a geometry theorem-proving machine on the IBM 704 high-speed electronic digital computer. Statements in the language are written in usual Fortran notation, but with a large set of special list-processing functions appended to the standard Fortran library. The algebraic structure of certain statements in this language corresponds closely to the structure of an NSS list, making possible the generation and manipulation of complex list expressions with a single statement. There are many programming advantages accruing from the use of Fortran, and in particular, the ease with which massive and complex programs may be revised, combined with the flexibility offered by a list organization of storage make the language particularly useful where, as in the case of our theorem-proving program, intermediate data of unpredictable form, complexity, and length may be generated.[12]

Modern machine learning[4]has been applied to the automation of heuristic selection in a first order logic theorem prover. One objective was to find if there are any features of a proof problem that are both easy to measure and provide useful information for determining heuristic choice. Another was to determine and demonstrate a practical approach to making theorem provers truly automatic.

In the experimental work, heuristic selection based on features of the conjecture to be proved and the associated axioms is shown to do better than any single heuristic. Additionally a comparison has been made between static features, measured prior to the proof search process, and dynamic features that measure changes arising in the early stages of proof search.

4 Automated Theorem Proving Systems

Waldmeister High-Performance Theorem Prover[22]
This is a first order Automatic Theorem Proving System. This is one of the best Automatic theorem provers.It is a three-level logical system model which consists of the basic operations on the lowest level, where great stress is put on efficient data structures and algorithms. For the middle level, where the inference steps are aggregated into an inference machine, flexible adjustment has proven essential during experimental evaluation. The top level holds control strategy and reduction ordering. Although at this level only standard strategies are employed, really large proof tasks have been managed in reasonable time.

Equational prover(EQP[17]
Equational prover(EQP)is an automated theorem proving system which was used to solve the problem posed by Herbert Robbins. This presents a solution that all Robbins algebras are boolean and it is proved by EQP. This was considered a major breakthrough for theorem provers in application to mathematics.

E a brainiac theorem prover[20]
This is a first order logic theorem prover. E is a fully automatic theorem prover for clausal logic with equality. It is a saturating prover based on a purely equational view, with a strong emphasis on rewriting. E- Theorem Prover is considered as brainiac for two reasons. First, E's proof search is more intelligent as there is much work done developing and evaluating good search control heuristics. Secondly, in rewriting engine, E substitues very complex operations in place for a large number of simpler operations.

Otter theorem prover[16]
Otter (Organized Techniques for Theorem-proving and Effective Reasoning) is an automated theorem proving program for first-order logic with equality developed by the Mathematics and Computer Science Division of Argonne National Laboratory.This is developed by William McCune. This program is coded in the C programming language and run mostly on UNIX systems, but there are versions for both Macintosh and Windows.

The HOL System[13]
The HOL system is designed to support interactive theorem proving in higher order logic (hence the acronym 'HOL'). To this end, the formal logic

is interfaced to a general purpose programming language (ML, for meta-language) in which terms and theorems of the logic can be denoted, proof strategies expressed and applied, and logical theories developed. The version of higher order logic used in HOL is predicate calculus with terms from the typed lambda calculus (i.e. simple type theory). This was originally developed as a foundation for mathematics. The primary application area of HOL was initially intended to be the specification and verification of hardware designs.

The COQ System[5]
Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. As a proof development system, Coq provides interactive proof methods, decision and semi-decision algorithms, and a tactic language for letting the user define its own proof methods. Connection with external computer algebra system or theorem provers is available. It is also used as a platform for the formalization of mathematics or the development of programs, Coq provides support for high-level notations, implicit contents and various other useful kinds of macros.

The Boyer-Moore theorem prover[3]
The boyer-moore theorem prover was the basis for the development of ACL2 theorem prover.It is also known as "Nqthm"(New Quantified theorem prover). It has been used to perform a variety of verification tasks for two decades. It supports the constrained function symbols and the use of a derived rule of inference permitting the initiation of function symbols.

ACL2 theorem prover[15]
It is one of the general-purpose automated reasoning systems.ACL2(A Computational Logic for Applicative Common Lisp) is a reimplemented extended version of Boyer and Moores Nqthm and Kaufmanns Pc-Nqthm, intended for large scale verification projects. The next section is a detailed description of ACL2 Theorem Prover.

5 ACL2 Theorem Prover

ACL2 is an efficient functional programming language. ACL2 functions and system models can be compiled by any compliant Common LISP compiler.

Models can be made to execute efficiently. The ACL2[14] system is almost written entirely in ACL2. ACL2 is also used to model hardware designs, microprocessors, algorithms, and other artifacts that are not only analyzed symbolically but are also tested by execution.

ACL2 is a direct descendant of the Boyer-Moore system, Nqthm, and its interactive enhancement, Pc-Nqthm. Like Nqthm, ACL2 supports a Lisp-like, first-order, quantifier-free mathematical logic based on recursively defined total functions. A significant difference between the Boyer-Moore logic and that of ACL2 lies in the treatment of numbers. The Boyer-Moore logic recognizes only the integers, whereas ACL2 has built-in support for all rational numbers in the complex plane. However, the ACL2 predicate characterizing numbers enumerates all the possible numeric types, explicitly ruling out the irrationals. This may seem to be a reasonable limitation if one views ACL2 as a vehicle for proving theorems about Common Lisp functions.

Experience with the earlier systems supports the claim that such a logic is sufficiently expressive to permit one to address deep mathematical problems and realistic verification projects. The fact that the Nqthm logic is executable is also an important asset when using it to model hardware and software systems: the models can be executed as a means of corroborating their accuracy. ACL2 has been used in modeling and verification projects within Computational Logic, for several years.

This simple fact cannot be expressed in ACL2, since ACL2 does not have a first-class notion of infinite set. Hence non-standard analysis[11] is introduced which provides a suitable framework for reasoning about the irrationals in ACL2. The formalism of non-standard analysis in ACL2 follows the axiomatic approach pioneered by Nelson in Internal Set Theory (IST). Since ACL2 is a first-order logic with only limited support for quantification, it seems far too weak to reason about the reals. Consider, for example, the least upper bound axiom, which states that every bounded set of real numbers has a least upper bound.

ACL2(r)[6] modifies ACL2 by introducing some functions to its Ground Zero Theory, definitional principles, and derived inference rules all of which correspond to their counterparts in IST. For example, ACL2(r) introduces the built-in function standard which corresponds to the predicate standard of IST. ACL2(r) also classifies function symbols as either classical or non-classical: Functions which depend on standard are non-classical, while all other functions (including all functions that are also in ACL2) are classical. As with IST, ACL2(r) restricts the use of induction on formulas that are non-classical. ACL2(r) includes a new definition principle via the event defun-std. This principle corresponds to using the shadow of a non-classical function in IST to construct a new classical function. Finally, ACL2(r) adds the derived inference rule that corresponds to ISTs Transfer Principle with the event defthm-std. This event allows users to submit to ACL2(r)classical formulas that they want to be proved using transfer.

Also ACL2(r), a modified version of ACL2 is introduced with support for non-standard analysis[11]. Before introducing non-standard analysis into ACL2, it is necessary to extend the ACL2 numeric system to include the irrationals. This can be accomplished by adding the new type recognizers realp and complexp. In addition, the ACL2 arithmetic axioms need to be modified to account for these new types. In many cases, this can be accomplished by substituting realp for rationalp and complexp for complex-rationalp. Hence it is significant that ACL2(r) can prove any theorems at all from analysis, since its language, with weak support for quantifiers and without infinite sets, may appear too barren for analysis.

Many improvements can be shown to the theory of differentiation that is formalized in ACL2(r).As an initial step, shows how the normal rules for the differentiation[8] of composite functions can be introduced in ACL2(r). More important, is how the application of these rules can be largely automated, so that ACL2(r) can automatically define the derivative of a function that is built from functions whose derivatives are already known. Second, shows a formalization in ACL2(r) of the derivatives of familiar functions from calculus, such as the exponential, logarithmic, power, and trigonometric functions. These results serve as the starting point for the automatic differentiation tool described above. Third, describes how users can add new functions and their derivatives, to improve the capabilities of the automatic differentiator.

Misra introduced the powerlist[10] data structure and powerlist algebra, which is particularly well-suited to express and reason about recursive parallel algorithms. Powerlist is a linear data structure, a list of elements. The powerlists data structure, created by Misra in the early 90s, is well suited to express recursive, data-parallel algorithms. Misra has shown how powerlists can be used to give simple descriptions to very complex algorithms, such as the Fast Fourier Transform (FFT)[9]. Such simplicity in presentation facilitates reasoning about the resulting algorithms, and in fact Misra has presented a stunningly simple proof of the correctness of the FFT.

Powerlists are formalized using the ACL2 theorem prover. The formalization of powerlist algebra demonstrates the usefulness of reasoning about constrained functions with encapsulate. However, encapsulate supports reasoning about functions, it is a strictly first-order inference rule. Moreover, its application in ACL2 requires extensive use of hints to direct the theorem prover in the mapping of actual functions to constrained functions. Automation in this process would be enormously useful in Theorem Proving.

The verification of many algorithms for calculating transcendental functions is based on polynomial approximations to these functions, often Taylor series approximations. However, computing and verifying approximations to the arctangent[7] function are very challenging problems, in large part because the Taylor series converges very slowly to arctangent a 57th-degree polynomial is needed to get three decimal places for arctan(0:95). Medina proposed a series of polynomials that approximate arctangent with far faster convergence a 7th-degree polynomial is all that is needed to get three decimal places for arctan(0:95). In order to make arctangent more tractable, Medina first reduces the domain of arctangent to [0,1] and the result of Medinas which defined a polynomial approximation to arctangent helps in converging quickly. The formalization in ACL2(r) of a polynomial can be described as an approximation to arctangent. The proof made heavy use of results from prior work formalizing real analysis, such as the FTC, composition rules for derivatives, etc. In addition, a handful of results were missing and were proved as part of this effort, such as the First Derivative Test.

6 Rectifiable curves Theorem

A Rectifiable curve is a curve which has finite length.The algebraic definition of curves in the space is a set of points (x, y) of the plane verifying an equation of the form :

$$F(x, y) = 0.$$

There is also a geometric definition of curves. A curve is the intersection of two surfaces. This definition is more ambiguous since the result could be an empty set.

In general we can say that ℓ is a curve if it is the image in the plane or in space of an interval $[a, b]$ of real numbers of a continuous function γ.

Length of a polynomial curve P is the sum of the lengths of its constituent line segments; we denote it by $L(P)$. Given a curve ℓ, we shall define its length to be a limit of sequence. Let ℓ be a simple curve , and let (P_n) be a sequence of polygonal approximations of ℓ such that the maximum length of their line segments converges to 0. The length of ℓ is is by definition the limit of the sequence of the lengths of (P_n)[23]:

$$L(P_\mathrm{n}) = \limsup_{n\to\infty} L(P_\mathrm{n}).$$

We consider piece wise differentiable arc γ with the equation $z = z(t)$. Then the length of a curve can also be defined as the least upper bounds of all sums:

$| z(t_1) - z(t_0) | + | z(t_2) - z(t_1) | +......+ | z(t_{\mathrm{n}}) - z(t_{\mathrm{n\text{-}1}}) |$

where $a = t_0 < t_1 < ..... < t_{\mathrm{n}} = b$. if this least upper bound is finite we say that the arc is $rectifiable$.[1].

This Theorem has not yet proved using Automated Theorem Proving systems. This theorem can be proved using differential rules which were introduced in ACL2(r). My Thesis work is to provide a proof for this theorem using ACL2.

7 Conclusion

As like many Software tools Theorem Proving systems were introduced originally for mathematics but now they have a wide range of applications which provide motivation for the work to make Theorem Prover more accessible. A key modern application of Theorem Provers in modern days is in the verification of hardware and software designs. There are several developments in the field of Theorem Proving to introduce such Automated Theorem proving systems and will continue as these are very important and have wide variety of applications.

References

[1] L. V. Ahlfors. *Complex Analysis- an introduction to the theory of analytic functions of one complex variable.* McGraw-Hill Book Company, 1978.

[2] W. W. Bledsoe and P. Brueil. A man machine theorem-proving system. 1974.

[3] R. S. Boyer and M. Kaufmann. The boyer-moore theorem prover and its interactive enhancement. 1993.

[4] J. P. Bridge. Machine learning and automated theorem proving. 2010.

[5] A. Chlipala. *Certified Programming with Dependent Types.* 2013.

[6] R. Gamboa and J. Cowles. Theory extension in acl2(r). 2007.

[7] R. Gamboa and J. Cowles. Formal verification of medinas sequence of polynomials for approximating arctangent. 2014.

[8] R. Gamboa and P. Reid. Automatic differentiation in acl2. 2011.

[9] R. A. Gamboa. The correctness of the fast fourier transform: A structured proof in acl2. 1999.

[10] R. A. Gamboa. A formalization of powerlist algebra in acl2. 2009.

[11] R. A. Gamboa and M. Kaufmann. Non-standard analysis in acl2. 2001.

[12] H. Gelernter, J. R. Hansen, and C. L. Gerberich. Fortran-compiled list-processing language. 1960.

[13] M. Gordon. The hol system description. 2005.

[14] M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning An Approach.* kluwer Academic, 2000.

[15] M. Kaufmann and J. S. Moore. An industrial strength theorem prover for a logic based on common lisp. 1997.

[16] A. L. Mann. A case study in automated theorem proving: Otter and eqp. 2003.

[17] W. McCune. Solution of the robbins problem. 1997.

[18] A. Newell and H. A. Simon. The logic theory machine a compex information processing system. 1961.

[19] F. J. Pelletier. The philosophy of automated theorem proving. 1989.

[20] S. Schulz. E a brainiac theorem prover. 2002.

[21] C. E. Shannon. Machines are getting brainy. 1961.

[22] R. V. Thomas Hillenbrand, Arnim Buch and B. L. Chner. Waldmeister high-performance equational deduction. 1997.

[23] C. Tricot. *Curves and Fractal Dimension.* Springer, 1995.