Automated Generation of Interesting Theorems

Yury Puzis, Yi Gao, Geoff Sutcliffe
Department of Computer Science, University of Miami
P.O. Box 248154, Coral Gables, FL 33124, USA
Email: {puzis,yigao}@mail.cs.miami.edu, geoff@cs.miami.edu

Abstract

In the logical theory of a set of axioms there are many boring logical consequences, and scattered among them there are a few interesting ones. The few interesting ones include those that are singled out as *theorems* by experts in the domain. This paper describes the techniques, implementation, and results of an automated system that generates logical consequences of a set of axioms, and uses filters and ranking to identify interesting theorems among the logical consequences.

1. Introduction

Automated Theorem Proving (ATP) deals with the development of computer programs that show that some statement (the conjecture) is a logical consequence of a set of statements (the axioms). ATP systems have been used successfully in a range of domains, including software design and verification, e.g., (Denney, Fischer, & Schumann 2004), natural language processing, e.g., (Bos & Markert 2005), and the semantic WWW, e.g., (Tsarkov *et al*. 2004). In applications, a primary mode of operation is to supply an ATP system with axioms and a conjecture, and to ask the system to produce a proof (or at least an assurance that there is a proof) that the conjecture is a theorem of the axioms. Full automation of this task has been highly successful when the problem is expressed in classical 1st order logic, so that a proof by refutation of the clause normal form of the problem can be obtained. There are some well known high performance ATP systems that search for proofs by refutation of the clause normal form, e.g., E (Schulz 2002) and Vampire (Riazanov & Voronkov 2002). The techniques presented in this paper are in terms of classical 1st order logic, and henceforth all discussion is in that context.

The successes of ATP are testament to the heuristics that guide the search from the axioms towards the conjecture (in refutation based systems, towards a contradiction). Given a set of (non-contradictory) axioms, the number of logical consequences reachable at a given search depth (i.e., the size of the theory up to that depth) is $O(NumberOfAxioms^{2^{SearchDepth}})$. In such a theory, as well as the target conjecture, there are inevitably very many "boring" logical consequences, and scattered among them there are a few "interesting" ones. The interesting ones include those that are singled out as *theorems* by experts in the domain. This observation leads to the core idea for this work: to generate the logical consequences of a set of axioms, and identify the interesting theorems therein.

This paper describes the techniques, implementation, and performance of an automated system that uses an ATP system to generate logical consequences of a set of axioms, and then uses filters and ranking to identify interesting theorems among the logical consequences. The system is called AGInT (Automated Generation of Interesting Theorems).

1.1. Automatic Conjecture and Theorem Creation

The notion and a drive for automated discovery of knowledge goes back at least to the 13th century, with Ramon Llull's *ars combinatoria* for mechanically generating Christian doctrines. More recently, Newell and Simon predicted that a computer would discover and prove an important mathematics theorem (Simon & Newell 1958), and several programs have been developed with a focus on mathematical theorem discovery: Doug Lenat's AM program (Lenat 1976), Simon Colton's HR program (Colton 2002), and Roy McCasland's MATHsAiD program (McCasland 2005). Some of the concepts embodied in some of AGInT's filter measures have been used in these and other reasoning tools, e.g., (Denzinger & Schulz 1996).

There are at least four ways in which conjectures and interesting theorems can be created. The first approach, the *inductive* approach, is often used by humans. For example, noting that the prime numbers 5, 7, 11, 13 are all odd, the conjecture that all prime numbers are odd may be induced. An attempt is then made to prove (or disprove) the conjecture, and if disproved, modifications may be made, e.g., that all prime numbers greater than 2 are odd. Such reparation of faulty conjectures is described in (Lakatos 1976), and a project to automate reparation is described in (Pease *et al*. 2001). The inductive approach has the advantage of being stimulated by observations in the domain. but has the disadvantage that induction is unsound. As a result many non-logical conjectures are produced, and effort is expended finding disproofs. In some cases a disproof may be hard to find, and the conjecture remains open.

The second approach, the *generative* approach, generates

conjectures and tests them for theoremhood. The simplest form of generation is syntactic, in which conjectures are created by mechanical manipulation of symbols, e.g., (Plaisted 1994). The MCS program (Zhang 1999) generates conjectures syntactically and filters them against models of the domain. A stronger semantically based approach is taken by the HR program, which generates conjectures based on examples of concepts in the domain. Like induction, generation is unsound. However, if the rules by which the generation is performed are sufficiently conservative then this approach may generate a higher fraction of theorems than the inductive approach.

The third approach, the *manipulative* approach, generates conjectures from existing theorems. An existing theorem is modified by operations such as generalization, specialization, combination, etc. This approach is used in abstraction mapping, which converts a problem to a simpler problem, and uses a solution to the simpler problem to help find a solution to the original problem (Plaisted 1980). Manipulation of ATP problems has also been used to produce new problems for testing the robustness of ATP systems' performances (Voronkov 2000). An advantage of the manipulative approach is that if the manipulations are satisfiability preserving, then theorems, rather than conjectures, are produced from existing theorems. However, the conjectures produced by the manipulative approach are typically artificial in nature, and thus boring.

The fourth approach is the *deductive* approach. In this approach logical consequences are generated by application of sound inference rules to the axioms and previously generated logical consequences. This can be done by an appropriately configured saturation-based ATP system. The generation may be guided so that only interesting theorems are produced, or may be post-processed to select interesting theorems from all generated logical consequences. The advantage of this approach is only logical consequences are ever generated. The challenge of this approach is to avoid the many boring logical consequences that can be generated. The work described in this paper uses the deductive approach. The deductive approach is also taken in the MATHsAiD program, which is specialized in a conservative fashion towards mathematical domains, especially set theory.

2. The AGInT System

The overall architecture of AGInT is shown in Figure 1. At the macro-level, AGInT iterates over a process that: (i) uses an ATP system to generate logical consequences from a set of axioms enriched with the most interesting theorems from the previous iteration (none in the first iteration), (ii) filters the logical consequences twice to remove boring logical consequences, (iii) ranks the remaining candidate theorems, (iv) post-processes the ranked theorems in conjunction with all the interesting theorems stored in the previous iteration, (v) stores the survivors as the new set of interesting theorems, and (vi) loops. Each iteration of the loop is limited by a CPU time limit imposed on the ATP system, and the maximal number of iterations is a system parameter.

The ATP system used to generate the logical consequences must implement the set-of-support strategy (Wos,

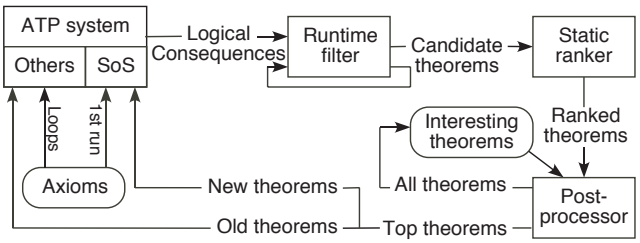

Figure 1: AGInT Architecture

Robinson, & Carson 1965), which forces the derived formulae to have one of the set-of-support formulae (placed in the set-of-support at the start of execution) as an ancestor. In AGInT's first iteration the axioms are placed in the set-of-support, so that the logical consequences cover the entire search space. In subsequent iterations a fixed maximum number (a system parameter) of the most interesting theorems are cycled back as input to the ATP system. Of the interesting theorems cycled back, those deduced in the immediately preceding iteration are placed in the set-of-support, while those carried forward from earlier iterations are supplied with the axioms to interact with the set-of-support. Thus the logical consequences generated in the second iteration onwards have an interesting ancestor that was generated in the immediately preceding iteration. The interesting theorems carried forward act as lemmas in the subsequent iterations. This technique focuses the search of the ATP system, improves the interestingness of the logical consequences generated, and allows the search to proceed deeper into the search space. The effect is illustrated in Figure 2, where the dots represent the interesting theorems found in that iteration's part of the search space, and which are added to the set-of-support in the subsequent iteration. If none of the cycled interesting theorems are from the immediately preceding iteration (i.e., all were generated in earlier iterations), then the system has converged, and is halted.

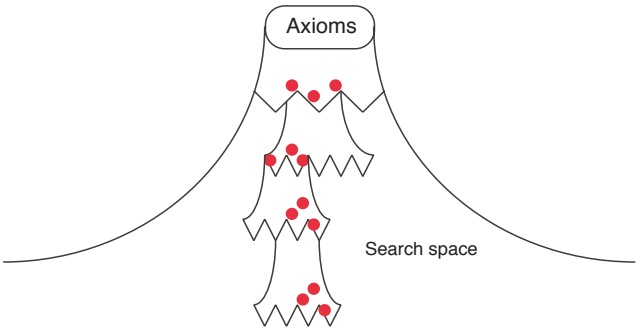

Figure 2: The Effect of the Set-of-Support Strategy

The final output from AGInT is an ordered list of the interesting theorems retained in the store. A human user can then select from the list those theorems (s)he considers worth using. It is not expected that all the theorems ranked as most interesting by AGInT will be precisely those selected by the

user. However, in the experimental results obtained thus far, and presented in Section 4, the rankings generated by AGInT are not inconsistent with a reasonable human selection. Additionally, in the testing AGInT generated and identified interesting theorems that had not been explicitly identified in the source domain.

The implementation of AGInT is somewhat adapted to the generation and filtering of formulae in clause normal form, i.e., disjunctions of literals. The ATP system is thus used to generate clauses, and some of the techniques described below are specialized for clauses. Most of the techniques are, however, generally applicable in first order logic.

2.1.The Runtime Filter

The task of the runtime filter is to aggressively filter out and discard boring logical consequences, to reduce the stream generated by the ATP system to a manageable level. The overall architecture of the runtime filter is shown in Figure 3, and details of each specific filter are given below.

Each incoming logical consequence must first pass the pre-processor, and must then pass the majority (i.e., at least four) of the seven filters: obviousness, weight, complexity, surprisingness, intensity, adaptivity, and focus. Those that pass the filters are used to update a sliding window associated with each filter, and are then stored for a second pass (described below). The logical consequences that are output from the runtime filter are hypothesized to be interesting theorems.

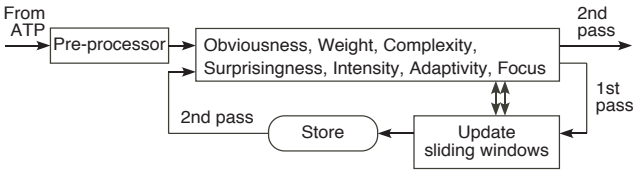

Figure 3: Runtime Filter Architecture

Each filter maintains a sliding window defined by the best distinct scores from the filter's evaluation of the logical consequences seen so far. The bounds of a window are thus the worst and best scores in the window. The number of distinct scores used to form a window is a system parameter. As multiple formulae typically have the same score for a given filter, a window of N scores typically accommodates many more than N formulae.

The upper and lower bounds of each window are initialized to the worst possible score for that filter. If an incoming logical consequence is scored equal to or better than the lower bound, it passes the filter. If the new score is different from all the scores in the window, and there are less than the maximum number of scores in the window, then the new score is added to the window. If the new score is different from all the scores in the window, and there are already the maximum number of scores in the window, then the new score displaces the current lower bound score. The new worst score in the window becomes the lower bound of the window. In the latter case, if the new score is better than the upper bound the effect is to slide the window up, while if the new score is within the window the effect is to reduce the size of the window. As the number of different scores for each filter is limited in practice, the windows do not shrink excessively.

Initializing the upper and lower bounds to the worst possible score allows all logical consequences through until the window starts sliding up. As a result some boring logical consequences early in the stream may pass the runtime filter. Therefore the logical consequences that pass the runtime filter in the first pass are stored, and when the stream terminates (due to the CPU time limit on the ATP system generating the stream) they are filtered again, with the windows fixed from the first pass. This removes any that do not meet the final lower bounds.

The individual filters are described next.

Pre-processor
The preprocessor detects and discards obvious tautologies, e.g., clauses that contain an atom and it's negation, and clauses containing a `true` atom. Additionally, any `false` literals are removed from clauses.

Obviousness
Obviousness estimates the difficulty of proving a formula. Formulae that are more difficult to prove are more interesting. The obviousness score of a formula is the number of inferences in its derivation. A higher score is better.

Weight
Weight estimates the effort required to read a formula. Formulae that contain very many symbols (variables, function and predicate symbols) are less interesting. The weight score of a formula is the number of symbols it contains. A lower score is better.

Complexity
Complexity estimates the effort required to understand a formula. Formulae that contain very many different function and predicate symbols, representing many different concepts and properties, are less interesting. The complexity score of a formula is the number of distinct function and predicate symbols it contains. A lower score is better.

Surprisingness
Surprisingness measures new relationships between concepts and properties. Formulae that contain function and predicate symbols that are seldom seen together in a formula are more interesting. The symbol-pair surprisingness score of a pair of function/predicate symbols is the number of axioms that contain both symbols divided by the number of axioms that contain either symbol. The surprisingness score of a formula is the sum of the symbol-pair surprisingness scores, over all pairs of distinct function/predicate symbols in the formula. A lower score is better.

Intensity
Intensity measures how much a formula summarizes information from the leaf ancestors in its derivation tree. Formulae that summarize a lot of information are more interesting. The plurality score of a formula (or set of formulae) is number of function and predicate symbols in the formula divided

by the number of distinct function and predicate symbols in the formula. Plurality measures the extent to which symbols are repeated in the formula. The intensity score of a formula is the plurality of its leaf ancestors divided by the plurality of the formula itself. A higher score is better. Intensity works well with complexity – interesting things are often compact, which means intense but not complex.

Adaptivity
Adaptivity measures how tightly the universally quantified variables of a formula are constrained. It is valid only for formulae in clause normal form, where variables are known to be universally quantified without considering their polarity (with respect to, e.g., negation or implication). Each occurrence of a universally quantified variable in a clause further constrains its properties. Thus a variable that occurs only a few times is less constrained than one that occurs more often. The adaptivity score of a clause is the number of distinct variables in the clause divided by the number of variable occurrences in the clause. A lower score is better.

Focus
Focus measures the extent to which a formula is making a positive or negative statement about the domain of application. Formulae that are very negative or very positive are more interesting. The focus measure relies on the observation that predicates almost always make positive statements of domain properties. It is valid only for formulae in clause normal form. For clauses with four or more literals the focus score is measured using an entropy formula: Let FPL and FNL be the fractions of positive and negative literals in a clause. The focus score of a clause is $1 + FPL * log_2(FPL) + FNL * log_2(FNL)$. Clauses with up to three literals are considered to have perfect focus because their polarity distribution is limited. A higher score is better.

2.2. The Static Ranker

The task of the static ranker is to compute a final interestingness score for the interesting theorems that are passed out from the runtime filter. This is done in two phases: first a usefulness score is computed for each theorem, and second, all the scores are individually normalized and then averaged.

Usefulness
Usefulness measures how much an interesting theorem has contributed to proofs of further interesting theorems, i.e, its usefulness as a lemma. Theorems that are ancestors of more interesting theorems are themselves more interesting. The usefulness score of a theorem is the ratio of its number of interesting descendents (i.e., descendents that have passed the runtime filter) over its total number of descendents. A higher score is better.

Normalization and Averaging
The scores of the theorems, from each of the runtime filter and static evaluations, are normalized into the range 0.0 to 1.0. The theorems with the worst score are given a final score of 0.0, the formulae with the best score are given a final score of 1.0, and all other scores are linearly interpolated in between. If the worst and best score of a particular filter are equal, then that filter does not differentiate between the theorems, and those scores are removed from the theorems. The remaining scores of each interesting theorem are averaged to produce a final interestingness score.

2.3. The Post-processor

The task of the post-processor is to remove less interesting theorems that are redundant with respect to more interesting theorems, thus implementing a novelty filter on the theorems. Redundancy is tested in terms of subsumption for clauses, and deducibility for all formulae. The tests performed in the post-processor, particularly deducibility, are quite computationally expensive (compared to the filters).

The first part of post-processing discards all except the 1000 most interesting of the combined set of the stored theorems and the theorems output from the static ranker. This discards interesting theorems that are extremely unlikely to survive to enrich the input to the ATP system in the next iteration, or to be effective within the remaining post-processing. (Experimentation with AGInT has shown that, e.g., most subsumptions done in the post-processing are done by more interesting theorems, and that keeping the 1000 most interesting theorems is sufficient in practice.) An intended side-effect is to reduce the number of expensive subsumption and deducibility tests that need to be performed.

The second part of post-processing considers the remaining interesting theorems in pairs, in descending order of interestingness, so that each theorem is compared with every other less interesting theorem. If the interesting theorems are clauses, then subsumption testing is done. If the first (more interesting) clause of a pair is subsumed by one of the axioms, it is discarded. Otherwise, if the first subsumes that second (less interesting) of the pair, the second is discarded. Otherwise, if the second subsumes the first, then the first is discarded. If no subsumption occurs then in some cases a test is made to find if the less interesting theorem can be easily proved from the axioms and the more interesting theorem. The test is made only if the leaf ancestor set from the proof of the less interesting theorem is a subset of that of the more interesting theorem. In this case it is assured that, in principle, the less interesting theorem can be proved. "Easily proved" means proved within a very small CPU time limit by some chosen ATP system. A less interesting theorem that is easily proved is not interesting in the shadow of the more interesting theorem, and is discarded.

3. Performance

The AGInT system has been evaluated by having it generate interesting theorems from axiom sets from the puzzles (`PUZ`) domain of the TPTP problem library (Sutcliffe & Suttner 1998), and from the axiomatization of set theory given in (McCasland 2005). These domains were chosen because the theories are easily understood, thus making it possible to make a human judgement of the interesting theorems. The ATP system used to generate the logical consequences was E (Schulz 2002) version 0.9, with the command line flags `-x'(1*Defaultweight(SimulateSOS))'`

-WNoSelection to implement the set-of-support strategy. The CPU limit on E for each iteration of AGInT's loop was 4s, on a 2GHz P4 computer with 512MB memory, and running the Linux 2.6 operating system. Maximally 10 iterations of the loop were run, and maximally 100 interesting theorems were cycled back as input to E in each iteration. The runtime filter used windows of 5 distinct scores. The post-processor used Otter 3.3 (McCune 2003), with a time limit of 4s, to test if a less interesting theorem can be easily proved from the axioms and a more interesting theorem.

For each axiom set, the English explanation of the axioms is given. For the PUZ tests, an English explanation of the conjecture of the TPTP problem is also given. The conjecture provides a sample theorem considered interesting by humans (the conjecture is not provided in the input to AGInT). The logical encodings of the axiom sets, which cannot be replicated here due to space restrictions, can be found in the indicated TPTP file for the PUZ tests. Following each axiom set description are the three most interesting theorems found by AGInT, translated directly into English. The final interestingness score of each is given in ()s. It is noteworthy that in each of the PUZ tests, AGInT generates and identifies interesting theorems that are not mentioned in the original problem, i.e., interesting theorems not explicitly identified in the source domain.

PUZ001-2
Axioms: Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler.
Conjecture: Aunt Agatha killed herself.
Interesting theorems: The killer does not hate the butler (1.00). Aunt Agatha killed herself (1.00). The killer is Aunt Agatha (0.96).

PUZ003-1
Axioms: There is a barbers' club that obeys the following three rules: (1) If any member A has shaved any other member B - whether himself or another - then all members have shaved A, though not necessarily at the same time; (2) Four of the members are named Guido, Lorenzo, Petrucio, and Cesare; (3) Guido has shaved Cesare.
Conjecture: Petrucio has shaved Lorenzo.
Interesting theorems: If two people are members, then the first has shaved the second (1.00). If someone is a member, then all members have shaved him (0.93). If someone is a member, then he has shaved Guido (0.85). Note: The conjecture is not saved as an interesting theorem, but is an instance of the more general theorems presented.

PUZ006-1
Axioms: Human observers in this exclusive club on Ganymede can't distinguish Martians from Venusians, males from females, except for the fact that Venusian women and Martian men always tell the truth and Venusian men and Martian women always lie. Ork says "Bog is from Venus". Bog says "Ork is from Mars". Ork says "Bog is male". Bog says "Ork is female".
Conjecture: Bog is female.
Interesting theorems: It is not a truth that "Ork is female" (1.00). It is not a truth that "Bog is from Venus" (1.00). Bog is not a truthteller (0.99). Note: The conjecture is generated as the seventh most interesting theorem, with an interestingness score of 0.98.

PUZ027-1
Axioms: There is an island with exactly three types of people - truthtellers who always tell the truth, liars who always lie, and normals who sometimes tell the truth and sometimes lie. Liars are said to be of the lowest rank, normals are middle rank, and truthtellers of the highest rank. Two people A and B on the island make the following statements. A: I am of lower rank than B. B: That's not true!
Conjecture: What are the ranks of A and B?
Interesting theorems: It is a truth that "everyone does not have a lower rank than themself" (1.00). A and B are normal (1.00). It is a truth that "B is normal" (0.98).

Set Theory
Axioms: $\forall X\ (X \notin \emptyset)$
$\forall A, B\ (\forall X\ (X \in A \Rightarrow X \in B) \Leftrightarrow A \subseteq B)$
$\forall A, B\ (\forall X\ (X \in A \Leftrightarrow X \in B) \Leftrightarrow A = B)$
$\forall A, B\ (\forall X\ ((X \in A \wedge X \in B) \Leftrightarrow X \in A \cap B))$
$\forall A, B\ (\forall X\ ((X \in A \vee X \in B) \Leftrightarrow X \in A \cup B))$
$\forall A, B\ (\forall X\ ((X \in A \wedge X \notin B) \Leftrightarrow X \in A - B))$
Interesting theorems: Nothing is a member of $\emptyset$ (1.00). $\emptyset$ is a subset of every set (0.99). The intersection of $\emptyset$ with any set is a subset of every set (0.91).

4. Conclusion

The discovery of interesting theorems is difficult from at least two perspectives. First, "interesting" is an ill-defined property that varies from person (or system) to person. Second, even with a chosen definition of what is interesting, the generation and identification of interesting theorems in a theory that grows super-exponentially with inference depth demands a carefully controlled process. AGInT has shown that the use of ATP technology, an integrated suite of filters and rankers, and a multi-iteration multi-phase process, is capable of producing pleasingly interesting theorems from a set of axioms. The final arbitration as to the interestingness of a theorem must be left to the user (human or otherwise), but computerized support that presents a limited selection of candidates can significantly reduce that effort.

Future work on AGInT will include incremental introduction of axioms into the system. Rather than presenting AGInT with the entire axiomatization of a theory at the first iteration, the axioms will be organized into layers, in the little theories sense (Farmer, Guttman, & Thayer 1992). A new outer loop will introduce the successive layers of axioms into AGInT in a controlled fashion. This approach, which is used in MATHsAiD, will allow AGInT to first explore a smaller theory, and then build on the results when

new axioms are introduced to extend the theory. Additional development of the filters is also planned, including a filter on the level of nested concepts, finer evaluation of focus for clauses with up to three literals, refinement of the adaptivity filter, measuring surprisingness by n-tuples of symbols, and measuring the usefulness of a formula in terms of the usefulness of its descendents,

Some of the process and parameters of AGInT have been established through empirical experimentation.[1] A formal study of the properties of the system is necessary. The goals of the study will include establishing a model of the whole computation, determining invariants of the component processes, and determining system parameters from the model. Further testing on both deep (e.g., set theory) and broad (e.g., the Cyc database) theories will provide an empirical evaluation of AGInT's performance.

References

Bos, J., and Markert, K. 2005. Recognising Textual Entailment with Logical Inference. In Brew, C.; Chien, L.-F.; and Kirchoff, K., eds., *Proceedings of the 2005 Conference on Empirical Methods in Natural Language Processing*, 836–840. Springer-Verlag.

Colton, S. 2002. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag.

Denney, E.; Fischer, B.; and Schumann, J. 2004. Using Automated Theorem Provers to Certify Auto-generated Aerospace Software. In Rusinowitch, M., and Basin, D., eds., *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, number 3097 in Lecture Notes in Artificial Intelligence, 198–212.

Denzinger, J., and Schulz, S. 1996. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation* 21:523–541.

Farmer, W.; Guttman, J.; and Thayer, F. 1992. Little Theories. In D., K., ed., *Proceedings of the 11th International Conference on Automated Deduction*, number 607 in Lecture Notes in Artificial Intelligence, 567–581. Springer-Verlag.

Lakatos, I. 1976. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press.

Lenat, D. 1976. *An Artificial Intelligence Approach to Discovery in Mathematics*. Ph.D. Dissertation, Stanford University, Stanford, USA.

McCasland, R. 2005. Automated Theorem Generation. In Carette, J., and Farmer, W., eds., *Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, Electronic Notes in Theoretical Computer Science. Elsevier.

McCune, W. 2003. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA.

Pease, A.; Colton, S.; Smaill, A.; and Lee, J. 2001. A Multi-Agent Approach to Modelling Interaction in Human Mathematical Reasoning. In *Proceedings of the 2001 International Conference on Intelligent Agent Technology*, Intelligent Agent Technology: Research and Development. World Scientific.

Plaisted, D. 1980. Abstraction Mappings in Mechanical Theorem Proving. In Bibel, W., and Kowalski, R., eds., *Proceedings of the 5th International Conference on Automated Deduction*, number 87 in Lecture Notes in Computer Science, 264–280. Springer-Verlag.

Plaisted, D. 1994. The Search Efficiency of Theorem Proving Strategies. In Bundy, A., ed., *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, 57–71. Springer-Verlag.

Riazanov, A., and Voronkov, A. 2002. The Design and Implementation of Vampire. *AI Communications* 15(2-3):91–110.

Schulz, S. 2002. E: A Brainiac Theorem Prover. *AI Communications* 15(2-3):111–126.

Simon, H., and Newell, A. 1958. Heuristic Problem Solving: The Next Advance in Operations Research. *Operations Research* 6(1):1–10.

Sutcliffe, G., and Suttner, C. 1998. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning* 21(2):177–203.

Tsarkov, D.; Riazanov, A.; Bechhofer, S.; and Horrocks, I. 2004. Using Vampire to Reason with OWL. In McIlraith, S.; Plexousakis, D.; and van Harmelen, F., eds., *Proceedings of the 3rd International Semantic Web Conference*, number 3298 in Lecture Notes in Computer Science, 471–485. Springer-Verlag.

Voronkov, A. 2000. CASC-16 1/2. Technical Report CSPP-4, Department of Computer Science, University of Manchester, Manchester, England.

Wos, L.; Robinson, G.; and Carson, D. 1965. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *Journal of the ACM* 12(4):536–541.

Zhang, J. 1999. System Description: MCS: Model-Based Conjecture Searching. In Ganzinger, H., ed., *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, 393–397. Springer-Verlag.

[1] As one reviewer commented, "This is witchcraft."