History and Prospects for First-Order Automated Deduction

David A. Plaisted

June 9, 2015

Introduction and General Comments

A Machine-Oriented Logic Based on the Resolution Principle
J.A.Robinson
J.ACM Volume 12 Number 1 January 1965

Abstract Theorem proving on the computer, using procedures based on the fundamental theorems of Herbrand concerning the first-order predicate calculus, is examined with a view towards improving the efficiency and widening the range of practical applicability of these procedures. A close analysis of the process of substitution (of terms for variables), and the process of truth-functional analysis of the results of such substitutions, reveals that both processes can be combined into a single new process (called *resolution*), iterating which is vastly more efficient than the older cyclic procedures consisting of substitution stages alternating with truth-functional analysis stages.

The theory of the resolution process is presented in the form of a system of first-order logic with just one inference principle (the resolution principle). The completeness of the system is proved; the simplest proof-procedure based on the system is then the direct implementation of the proof of completeness. However, this procedure is quite inefficient, and the paper concludes with a discussion of several principles (called search principles) which are applicable to the design of efficient proof-procedures employing resolution as the basic logical process.

Citation count: 758 (ACM Library), 4744 (Google Scholar?)

Significance

- The resolution rule
- Unification
- Refinements to resolution

Outline

- Fiftieth anniversary of the appearance of Robinsin's resolution paper
- History of first-order theorem proving both before and after 1965
- Personal reflections
- Possible future developments
  - Generalize model-based reasoning to first-order provers
  - Goal sensitivity and its importance
  - Analyze asymptotically the size of the search space of a first-order prover in terms of the size of a minimal unsatisfiable set of ground instances of a set of first-order clauses.

Importance of First-Order Logic

- Concentrate on first-order logic
- First-order logic is really central to the field
- There are probably significant advances yet to be made
- Methods that are good for first-order logic will also help to design higher order logic provers.

Personal Experiences

- Emphasize my personal experiences.
- It is a pity that Bill McCune, Harald Ganzinger, Greg Nelson, and Mark Stickel are not here to give their insights into the history of the field.

Search Space Issues

- Do different methods just explore different portions of the search space?
- Can the search space be reduced?
- Resolution as an improvement of DPLL for first-order logic
- DPLL over resolution for propositional logic
- Normal forms in rewriting
- Efficiency of basic operations

Pre-Resolution

- Friedrich Ludwig Gottlob Frege (b. 1848, d. 1925) was a German mathematician, logician, and philosopher who worked at the University of Jena. Frege essentially reconceived the discipline of logic by constructing a formal system which, in effect, constituted the first predicate calculus. In this formal system, Frege developed an analysis of quantified statements and formalized the notion of a proof in terms that are still accepted today. Frege then demonstrated that one could use his system to resolve theoretical mathematical statements in terms of simpler logical and mathematical notions.

- One of the axioms that Frege later added to his system, in the attempt to derive significant parts of mathematics from logic, proved to be inconsistent. Nevertheless, his definitions (e.g., of the predecessor relation and of the concept of natural number) and methods (e.g., for deriving the axioms of number theory) constituted a significant advance. To ground his views about the relationship of logic and mathematics, Frege conceived a comprehensive philosophy of language that many philosophers still find insightful. However, his lifelong project, of showing that mathematics was reducible to logic, was not successful.

[Stanford Encyclopedia of Philosophy]

- Frege was born in 1848 in Wismar, in the state of Mecklenburg-Schwerin (which is today part of the German federal state Mecklenburg-Vorpommern).

- His father, Alexander, a headmaster of a secondary school for girls, and his mother, Auguste, brought him up in the Lutheran faith.
- In 1873, Frege attained his doctorate under Ernst Christian Julius Schering, with a dissertation under the title of "Über eine geometrische Darstellung der imaginären Gebilde in der Ebene" ("On a Geometrical Representation of Imaginary Forms in a Plane"),
- in which he aimed to solve such fundamental problems in geometry as the mathematical interpretation of projective geometry's infinitely distant (imaginary) points.

Predicate Calculus

- Originally developed by Frege
- Considered predicates like "is happy" as signifying a function $H()$ of one variable that maps its argument x to a truth value $H(x)$, either true or false.
- Had universal quantification
- Existential quantification was expressed in terms of negation and universal quantification.
- The logic was originally second order.

Quantifiers

- William Ernest Johnson and Giuseppe Peano invented another notation, namely (x) for the universal quantification of x and (in 1897) ∃x for the existential quantification of x.
- For decades the canonical notation in philosophy and mathematical logic was (x)P to express "all individuals in the domain of discourse have the property P," and "$(\exists x)$P" for "there exists at least one individual in the domain of discourse having the property P."
- In 1935, Gentzen introduced the $\forall$ symbol, by analogy with Peano's $\exists$ symbol.
- $\forall$ did not become canonical until the 1960s.

Hilbert

Hilbert wanted to provide a secure foundation for mathematics including a formalization of mathematics and an algorithm for deciding the truth or falsity of any mathematical statement. This is known as Hilbert's program.

Gödel

- Gödel showed that Hilbert's program was impossible, in its obvious interpretation.
- He showed that any sufficienty powerful consistent formal system is incomplete.
- Or: For any sound and effective system F that can formalize Turing computations, there will be some Turing machine M that fails to halt on blank tape, but this fact cannot be shown in F.
- In fact, such a machine M can be constructed from F.
- These results apply to any consistent effective extension of Peano arithmetic, for example.
- Thus it is not possible to formalize all of mathematics in a computable way, and any attempt at such a formalism will omit some true mathematical statements.

Mechanical Theorem Proving

- The ATP community has inherited Hilbert's program to some extent, in attempting to prove and decide what can be proved and decided.
- There can be no recursive time bound on proving theorems, because of the undecidability of first-order logic.
- It is still possible to write theorem provers and attempt to improve their efficiency, even if not all true statements are provable.
- Herbrand's theorem gives a method to search for a proof of a formula in first-order logic by successively testing propositional formulas for validity.

- Herbrand's theorem is of major importance in software developed for theorem proving by computer.

Pre-Resolution Theorem Provers

- Gilmore's method was an early attempt to implement Herbrand's theorem.
- Another early approach was presented by Davis and Putnam.
- The linked conjunct method was still another early method that attempted to guide the instantiation of clauses to prove unsatisfiability.
- An early Wos paper mentions Gilmore's method but states that Davis and Putnam's method applied to sets of propositional clauses is much more efficient.
- The Wos paper also states that resolution can reduce the combinatorial explosion in Davis and Putnam's method by a factor in excess of 10^{50}.
- However, with faster propositional calculus implementations and by enumerating ground terms in a different way, this figure can possibly be reduced.

Resolution

- Unification and resolution in 1965 were the beginning of the modern era of theorem proving.
- Theorems could be proved that were significantly harder than those obtainable previously.
- Subsequent research was dramatically influenced by these ideas
- This influence continues even today even on non-resolution methods

Early Post-Resolution

During this period the provers were not much more powerful than the early provers combining hyper-resolution, set-of-support resolution, or UR resolution with paramodulation and demodulation.

The Argonne Group

- The Argonne group was the first group to devote serious effort to implementing Robinson's resolution rule.
- Some of the earliest theorems that they proved:

> In an associative system with left and right solutions, there is right identity element.
>
> In an associative system with an identity element, if the square of every element is the identity, the system is commutative.
>
> In a group, if the square of every element is the identity, the group is commutative.

For these proofs, the associativity of multiplication was represented by the following axioms.

$$\neg P(x, y, u) \vee \neg P(y, z, v) \vee \neg P(u, z, w) \vee P(x, v, w)$$

$$\neg P(x, y, u) \vee \neg P(y, z, v) \vee \neg P(x, v, w) \vee P(u, z, w)$$

- $P(x, x, e)$ was used to mean that the square of every element is the identity.
- Multiplication was represented in a relational manner. This reduces the need for explicit equational reasoning.
- The terms paramodulation and demodulation and the associated concepts were developed by the Argonne group.
- They also developed the set of support strategy.

- Hyper-resolution and P_1 deduction, on the other hand, were developed by Robinson.
- The Argonne prover was initially very slow.
- Finally McCune took the matter into his own hands and rewrote the entire prover in C, producing Otter, which was much faster and very easy to use.

Other Early Work

- Maslov's method
- Early refinements to resolution and other early strategies
  - ancestry filter form
  - model elimination
  - semantic resolution
  - locking resolution
  - merging
- Non-resolution methods
  - connection method of Bibel and Andrews' matings method
  - Now viewed as similar or identical to each other

At this time, the classic text of Chang and Lee appeared, which is still helpful. The Pelletier problems were often used to test theorem provers.

AI and Theorem Proving

- Initial enthusiasm for resolution in the artificial intelligence community
- Resolution was the basis of Cordell Green's QA3 system.
- Soon afterwards there was disenchantment with resolution and with uniform methods in general
- There was an emphasis on expert systems instead, which could perform at a human level in a narrow area.

- General systems were termed weak methods, and narrowly defined but more capable systems were termed strong methods by the AI community.
- Today ATP seems to be one tool in AI's toolkit, though not a solution to every problem.

Summary

We have considered various topics in the history of the field. Now for some personal reflections.

Personal Experiences

Pre-theorem proving work

- The book Computers and Thought
- Potential of computers for augmenting and simulating human intelligence.
- As an undergraduate I spent a summer working for MIT's Project MAC
- Early exposure to artificial intelligence research
- Later Vaughan Pratt explained the concept of NP-completeness
- Early work on this topic
- Also work in algorithms, partially motivated by Ed Reingold.

Equational reasoning work

- Dave Luckham suggested that I study methods for equational theorem proving.
- The seminal paper of Knuth and Bendix had only recently appeared.
- Published work appeared somewhat later

First-Order Theorem Proving

- When I started graduate school, it seemed that resolution and its refinements, such as ancestry filter form and locking resolution, plus model elimination, were the only games in town.
  - I didn't like resolution initially, but the concept is actually very natural.
  - If two clauses share a literal, then in some cases the shared literal can be eliminated and the remaining portions of the clauses can be joined together
  - The eliminated literal is a "bridge concept" to relate two other connected sets of concepts.
  - Two concepts related to the same concept are also related to each other, which is a natural idea.
- I was greatly influenced by Bledsoe's work
  - He showed that some reasonably simple set theory problems could not be solved easily by resolution.
  - He also did some early work on semantics

Problem sets

- In the early days, we tested our provers on common problems such as Schubert's Steamroller, the Zebra problem, and similar problems

  Zebra problem:

> There are five houses.
> The Englishman lives in the red house.
> The Spaniard owns the dog.
> Coffee is drunk in the green house.
> The Ukrainian drinks tea.
> The green house is immediately to the right of the ivory house.
> The Old Gold smoker owns snails.
> Kools are smoked in the yellow house.
> Milk is drunk in the middle house.

The Norwegian lives in the first house.
The man who smokes Chesterfields lives in the house next to the man with the fox.
Kools are smoked in the house next to the house where the horse is kept.
The Lucky Strike smoker drinks orange juice.
The Japanese smokes Parliaments.
The Norwegian lives next to the blue house.

Now, who drinks water? Who owns the zebra?

- Such problems seemed like personal friends.
- There was a sense of achievement when a proof was found.
- Today with the massive TPTP problem set the art of testing has greatly advanced, but perhaps something has also been lost.

Sequence of Strategies

My work in theorem proving progressed through a number of strategies, finding deficiencies in each one.

- In 1974 I had implemented a back chaining prover based on semantic trees with variables.
  - Vaughan Pratt gave me a problem to show that if in a group the square of every element is the identity, then the group is commutative.
  - My prover had a lot of trouble with this problem.
  - This example showed me that such a back chaining approach was not the way to go
- During my sabbatical at SRI I implemented a forward chaining resolution prover

- It had trouble with Pelletier's non-obviousness problem.
  - (1) $\neg q(c,d)$
  - (2) $\neg p(a,b)$
  - (3) $p(x,y) \vee q(x,y)$
  - (4) $q(x,y) \vee \neg q(y,x)$
  - (5) $\neg p(x,y) \vee \neg p(y,z) \vee p(x,z)$
  - (6) $\neg q(x,y) \vee \neg q(y,z) \vee q(x,z)$
- This convinced me that this also was not the right approach.

- At the University of Illinois, Steve Greenbaum implemented the Violet prover.
  - We put a lot of work into including abstraction, which did not turn out to be helpful.
  - The basic prover was fairly efficient compared with resolution provers of that time.
  - Used a discrimination net approach similar to term indexing
  - Linear time unification algorithm
  - Resolve the pair of clauses whose sum of sizes was as small as possible.
  - Still merits additional work.

Instance-Based Methods

- Next idea was to extend Prolog's back chaining strategy for Horn clauses to full first-order logic.
  - Led to the simplified problem reduction format and the modified problem reduction format
  - Still did not seem quite right
- Eventually I decided that what was needed was DPLL-style search in first-order logic.
- Led to work on instance based methods
- Leading to a sequence of provers

  - Clause linking
  - Semantic hyper-linking
  - Replacement rule theorem prover
  - Ordered semantic hyper-linking

- None implemented with highly efficient data structures except for OSHL
- Implemented by Hao Xu for his Ph.D. thesis with an inference rate often approaching 10,000 inferences per second.

Other instance-based methods

- Billon's disconnection calculus
- DCTP theorem prover
- Equinox
- Inst-Gen
- Linked conjunct method (very early)
- SATCHMO
- Hyper-tableaux?
- Even in Vampire?
- Perform particularly well on function-free clause sets
- These clause sets have some important applications.

Summary

So these are some of my personal experiences, closing with a discussion of instance-based methods. Now we return to the history of the field, and the developments that took place after the early post-resolution period.

Late Post-Resolution

During this period provers became significantly more powerful than the early resolution provers.

- Advances in refining resolution strategies
- Refining paramodulation and rewriting
- Basic paramodulation
- Instance-based methods
- Incorporating special axioms into first-order provers
- Decision procedures for specialized theories
- Efficient data structures
- Unification algorithms
- The CADE system competition has become an important event and a significant test of various provers.
- Lean theorem provers
- Major provers including Vampire, E, and Spass have become increasingly effective.
- The use of *strategy selection* has greatly helped major provers today, including Vampire and E.

Summary

This concludes our review of the history of the field. Now we make some general comments on resolution and a few other areas of automated deduction.

Comments on Resolution

Resolution initiated the modern era of theorem proving in 1965.

- The computation of most general unifiers avoids the necessity to enumerate all propositional instances of first-order formulas.
- Resolution not only uses most general unifiers, but with paramodulation and demodulation is also easily extendible to equality and rewriting.
- This is a good combination and may explain the persistence of resolution in theorem proving despite its inefficiency on non-Horn propositional problems.

Is it possible to go beyond resolution?

- A resolution prover is like a prolific but not very well organized mathematician filling notebooks with trivial deductions, with no overall sense of where he is going. Once in a while he stumbles on something interesting.
- What does a large set of clauses generated by resolution, mean? How is it making progress towards a proof? It is difficult to make any sense of tens of thousands of clauses in memory.
- Resolution is entirely syntactic; there is no semantics involved, though semantics can be introduced in semantic variants of resolution.
- Human mathematicians use semantics such as groups for group theory theorems. Perhaps our provers also should use more semantic information.

Even the most efficient propositional provers are benefited by conflict-driven clause learning (CDCL), which is essentially resolution. This shows that resolution is not going to disappear.

Propositional Calculus and SMT

- Increasing efficiency of propositional provers

- Even used to solve problems in other domains by translating them into propositional logic and then using a propositional satisfiability procedure.
- This is from the announcement for the Fifth International SAT/SMT Summer School, Stanford, CA, July 15-17, 2015:

> Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) solvers have become the engines powering numerous applications in computer science and beyond, including automated verification, artificial intelligence, program synthesis, security, product configuration, and many more. The summer school covers the foundational and practical aspects of SAT and SMT technologies and their applications.

- Propositional provers are now often used for model checking applications, in the bounded model checking approach.

Complexity of Propositional Satisfiability

- An NP complete problem
- Assuming that P is not equal to NP, satisfiability is exponential in the worst case.
- How is it then that propositional provers can be so efficient in practice?
  - Part of the reason is the so-called satisfiability threshold.
  - For problems with a large ratio of clauses to literals, DPLL is likely to finish quickly because the search tree will be small.
  - For clauses with a small ratio of clauses to literals, DPLL is likely to find a model quickly.
  - The hard problems tend to be those in the middle.
- The fastest propositional provers use not only DPLL but also CDCL, which helps them avoid repetitive parts of the search by learning the reason for various conflicts.

SMT

Another recent development is the increasing effectiveness of provers based on satisfiability modulo theories (SMT) and their applications.

- What's the next step beyond SMT to include more of first-order logic and decision procedures while maintaining the propositional efficiency of DPLL?
- Equinox achieved respectable performance in a possibly complete theorem proving method by combining an OSHL style prover and DPLL, and dealt with equality by congruence closure.
- Is it possible to extend this approach to more specialized decision procedures, and thereby obtain a way to extend SMT to a complete first-order strategy?

Equality and Term Rewriting Systems

- Musser's inductionless induction.
- I was amazed at the way one could prove inductive theorems by term rewriting system completion.
- Dallas Lankford had many pioneering papers in term-rewriting systems that unfortunately did not get published.
- Early termination techniques by Iturriaga for term rewriting systems were pioneering but largely superceded by the recursive path orderings, which were a tremendous advance in termination, though earlier orderings are still significant.
- The survey of rewriting by Huet and Oppen impressed me with the potential of term-rewriting techniques.
- Equational unification methods including AC unification have had a tremendous impact as well.
- Termination techniques using the dependency pair ordering are another significant development.

- Conditional term rewriting, higher order rewriting, rigid E-unification, and the Waldmeister prover.

Summary

We have discussed a few of the areas of automated deduction, namely propositional calculus, SMT, and term rewriting. Now we turn to some possible research directions in the field.

Discussion of Prover Features

Some desirable feaatures in a theorem prover:

- First-Order Logic
- Model Based Reasoning with Backtracking
- Goal Sensitive

First-Order

The first feature that is desired is that the logic should be first (or higher) order.

Model-Based Reasoning with Backtracking

The second feature is that it is desirable to have DPLL style model-based search and backtracking over partial interpretations in a first-order prover

- Model-based search with backtracking is what gives DPLL its efficiency.
- A survey paper by Bonacina, Furbach, and Sofronie-Stokkernmans considers various model-based theorem proving strategies, and shows how the term model-based reasoning can mean many different things.
- The following presentation is partly inspired by point set topology.

As motivation:

Any infinite set of points in [0,1] has a accumulation point, that is a point in which every neighborhood (open set) has infinitely many points in it.

Equivalently, any infinite sequence in [0,1] has an infinite subsequence that converges to a limit, that is, every neighborhood of the limit contains all but finitely many elements of the sequence.

This subsequence property is a consequence of *compactness*, namely, every open cover of [0,1] has a finite subcover.

How is this a consequence of compactness?

- Suppose an infinite set A of points in [0,1] did not have an accumulation point.
- Then about every $x \in [0,1]$ there would be a neighborhood $N(x)$ that contained only finitely many points of A.
- The set of such $N(x)$ is then a cover of [0,1].
- It does not have a finite subcover because any finite subcover will have only finitely many points from A.
- Thus [0,1] would not be compact (which it is).

With a suitable topology the set $\mathcal{I}$ of interpretations of an infinite set of atoms, also is compact and therefore has the subsequence property.

Now let's consider how to do theorem proving in first-order logic.

- Let $S = \{C_1, \ldots, C_n\}$ be a set of clauses and let A_i be the set of $I \in \mathcal{I}$ such that C_i contradicts I.
- Then S is unsatisfiable iff $\bigcup_i A_i = \mathcal{I}$.
- Thus for theorem proving, we want to know if $\bigcup_i A_i = \mathcal{I}$.

Here is one possible way to do theorem proving in this context:

```
Method A

Pick $I_1 \in \mathcal{I}$.
Pick $A_1$ such that $I_1 \in A_1$ (find $C_1$ that contradicts $I_1$).
If such an $A_1$ does not exist then stop, $S$ is satisfiable
If $A_1 = \mathcal{I}$ then stop, $A_1$ hence $S$ is unsatisfiable. *
Otherwise, pick $I_2 \in \mathcal{I}$, $I_2 \notin A_1$ ($I_2$ satisfies $C_1$). *
Pick $A_2$ such that $I_2 \in A_2$ (find $C_2$ that contradicts $I_2$).
If such an $A_2$ does not exist then stop, $S$ is satisfiable
If $A_1 \cup A_2 = \mathcal{I}$ then stop, $A_1 \cup A_2$ hence $S$ is unsatisfiable. *
Otherwise pick $I_3 \in \mathcal{I}$, $I_3 \notin A_1 \cup A_2$ ($I_3$ satisfies $C_1$ and $C_2$). *
...
```

This approach has the advantage of finiteness. However, the starred lines are not computable. In particular,

- the test for unsatisfiability is not decidable, and
- the search for I_j is not even partially decidable.

Instead, we do this:

```
Method B

Pick $I_1 \in \mathcal{I}$.
Pick $A_{k_1}$ such that $I_1 \in A_{k_1}$ ($C_{k_1}$ contradicts $I_1$)
If such an $A_{k_1}$ does not exist then stop, $S$ is satisfiable
Pick $N(I_1)$ such that $I_1 \in N(I_1) \subseteq A_{k_1}$ ($C_{k_1}$ contradicts all interpretations in $N(I_1)$).
If $N(I_1) = \mathcal{I}$ then stop, $N(I_1)$ hence $S$ is unsatisfiable. *
Otherwise, pick $I_2 \in \mathcal{I}$, $I_2 \notin N(I_1)$ ($I_2$ is not in the neighborhood $N(I_1)$). *
Pick $A_{k_2}$ such that $I_2 \in A_{k_2}$ ($C_{k_2}$ contradicts $I_2$).
If such an $A_{k_2}$ does not exist then stop, $S$ is satisfiable
```

Pick $N(I_2)$ such that $I_2 \in N(I_2) \subseteq A_{k_2}$ (C_{k_2} contradicts all interpretations in $N(I_2)$).
If $N(I_1) \cup N(I_2) = \mathcal{I}$ then stop, $N(I_1) \cup N(I_2)$ hence S is unsatisfiable. *
Otherwise, pick $I_3 \in \mathcal{I}$, $I_3 \notin N(I_1) \cup N(I_2)$ (I_3 is not in the neighborhoods $N(I_1)$ and $N(I_2)$). *
...

Here the sets $N(I_j)$ are neighborhoods of I_j that are chosen to make the starred lines computable; in particular, the test $\bigcup_j N(I_j) = \mathcal{I}$ decidable, and also the choice of $I_{j+1} \notin \bigcup_j N(I_j)$ is computable. The $N(I_j)$ are partial interpretations and represent the set of total interpretations having $N(I_j)$ as a subset.

Now Method B is computable but not necessarily terminating.

- To show termination we use the fact that $I_1, I_2, I_3, \ldots$ is an infinite sequence which therefore has an infinite subsequence converging to a limit I' because the set of interpretations is compact.
- Then every neighborhood of I' is also a neighborhood of all but finitely many I_j because I' is an accumulation point.
- Then we can use fairness of the choice of $N(I_j)$ to show that eventually some $N(I_j)$ will also be a neighborhood of I', hence of all but finitely many I_k, hence method B must stop if S is unsatisfiable.

This is the topological motivation for the method presented in the proceedings paper; there, a *conflict triple* for S is defined to be to be a triple (I, J, W) such that $J \in \mathcal{P}(I)$ (the set of neighborhoods of I), $W \in LC(S)$ (W is one of the C_j), and W contradicts I and all interpretations in J. The method given in the proceedings paper constructs a sequence (I_i, J_i, W_i) of such conflict triples.

- For DPLL, W_i is a clause that contradicts some prefix (subset) of I_i, and J_i would be some such prefix corresponding to the trail of DPLL. This means that J_i is always a finite partial interpretation for DPLL. J_i represents the set of all total interpretations having J_i as a subset.

- However, for first-order logic J_i could be a partial interpretation defined on infinitely many atoms, and might be defined for example on all ground instances of a finite set of possibly non-ground literals.
- The property that I_i is not an element of any previous $N(I_j)$ (does not have any previous J_j as a subset) is guaranteed in DPLL by the backtracking mechanism for DPLL.

By a *model-based method* we mean either something that fits into this formalism, or something that is in the same style even if it doesn't exactly fit the formalism.

- Model Evolution is one of the few strategies that is first-order and appears to be model-based in this sense.
- OSHL is model-based, but the clauses W_j are always ground clauses.
- Other instance-based methods including SATCHMO may also be model-based in this sense. SATCHMO also only generates ground clauses.

Goal Sensitivity

The third feature that is desired in a theorem prover is that inferences should be restricted to those that are related to the particular theorem and not just to general axioms.

- For clause form provers, if one wants to prove a theorem R from a set A of axioms, then one typically converts $A \wedge \neg R$ to clause form, obtaining a set S of clauses that is the union of clauses T from A and U from $\neg R$.
- In a goal sensitive method, clauses U from the negation of the theorem are typically considered to be relevant initially.
- Then the proof search is restricted so that clauses from T, and resolvents of input clauses, are only used if they are in some sense closely related to clauses from U.
- Generally A will be satisfiable, so that T will also be satisfiable. Let I be a model of T.

- Then if S is unsatisfiable, only clauses from U will contradict I. Thus one may consider that only clauses that contradict such an I are be relevant initially.
- Such an I will typically be a nontrivial model, that is, not obtained simply by choosing truth values of predicate symbols in a certain way.
- The Gelernter prover is an example of a prover using nontrivial semantics essentially for Horn clauses.

There are various ways to decide which derived formulas are relevant for a proof.

- One approach is to assign each formula a *relevance attribute*. The attribute can be true, indicating that the formula is relevant, or false, indicating that the formula is not relevant.
- These attributes are assigned initially so that only formulas related to the particular theorem are relevant.
- An inference is considered to be relevant if at least one of the hypotheses used in the inference is relevant.
- After each inference, the relevance attributes of formulas involved in the inference are updated. The conclusion of a relevant inference is always relevant, and the relevance attributes of the non-relevant hypotheses are changed from false to true.
- Some relevance strategies may also assign a numerical relevance distance attribute to formulas, indicating how relevant they are. Smaller distances indicate greater relevance.
- A method is *goal sensitive* if it is a relevance strategy, that is, it only performs relevant inferences. Such a strategy only generates relevant formulas.

Thus the result of an inference rule such as resolution, applied to a relevant clause and another clause, is also relevant. Operations other than resolution, such as instantiation, can also create a relevant instance $C\Theta$ of a clause C if they unify a literal of C with the complement of a literal of a relevant clause D.

Relevance in Equational Proofs

- For equational proofs of an equation $s = t$ from a set E of equations, if one can complete E then applying rewriting and narrowing (paramodulation) to s and t using the completed E suffices for completeness; $s = t$ is a logical consequence of E iff s and t rewrite to the same term.
- Thus such rewriting proofs are automatically goal sensitive, assuming that the theorem $s = t$ is selected to be relevant initially. Such goal-sensitivity is a tremendous advantage.
- If unfailing completion is used, the completion steps may not be relevant, but steps involving rewriting and narrowing of s and t will be relevant.

Relevance Distances

- There are also methods that compute at the start which clauses and even which instances of clauses are closely related to the particular theorem, so that proof strategies can concentrate on such clauses.
- These methods typically compute a relevance distance d of each clause from the particular theorem, with smaller distances indicating clauses that are more closely related to the particular theorem.
- Then these methods compute a set $R_d(S)$ of clauses and instances of clauses of S such that all clauses in $R_d(S)$ are at relevance distance d or less.
- This set $R_d(S)$ is computed so that it is unsatisfiable if there is an unsatisfiable set of ground instances of S of cardinality d or less.
- Typically $R_d(S)$ is computable from S in polynomial time, or at worst in exponential time, and $R_d(S) \subseteq R_{d+1}(S)$ for all $d \geq 0$.

Importance of Goal-Sensitivity

- For very large axiom sets, or axiom sets with many consequences, relevance methods are especially important.

- If there is no particular theorem, and one simply wants to test an axiom set A for satisfiability, then one can still apply relevance methods by choosing a known satisfiable subset B of A as the general axioms, and one can then consider $A - B$ as the particular theorem.
- This approach will at least avoid combining axioms of A in the search for a contradiction.
- Perhaps the concepts of goal relevance and semantics could even help propositional provers.

Summary

We have presented three properties that would be desirable in a theorem proving method. We would like a strategy that has all three properties, namely, first-order, model-based, and goal-sensitive. It seems that not many strategies have all three properties. Perhaps it would be worthwhile to develop such strategies.

Now we turn to a quantitative approach to analyzing the search space of theorem proving methods.

More Search Space Discussion

How can we give a rigorous complexity theoretic answer to the question whether one theorem proving strategy is better than another?

- For example, is resolution for first-order logic better than enumerating propositional instances and applying DPLL, and if so, by how much?
- We can test provers on specific examples to see which one is faster, but it would be better to have a more general method of evaluation.
- Theoreticians are highly interested in the complexity of resolution on propositional calculus problems.
- Is there some way to interest them in its complexity for first-order logic?

Even though first-order validity is only partially decidable, so that one cannot recursively bound the running time of a prover in terms of the length of the input, one can still discuss the asymptotic complexity of various theorem proving methods in terms of the size of a minimal Herbrand set.

Terminology

- Define a Herbrand set for a set S of clauses to be a set T of ground instances of clauses in S such that T is unsatisfiable.
- Let $H(S)$ be the minimal size of a Herbrand set for clause set S in some reasonable size measure.
- We define a complexity bound for a theorem proving method M to be a function F such that for all unsatisfiable clause sets S, the time taken by M on S is bounded by $F(H(S))$.
- Because the size of $H(S)$ is not recursively bounded in terms of S, it is possible to give recursive bounds on the running time of M in terms of $H(S)$ without implying that validity is decidable.

Now, the complexity of M can be highly dependent on details of the implementation, so to abstract this away,

- we define a function T_M which is closely related to the running time of M,
- say, within a small polynomial of the running time of M,
- and such that T_M has a simple mathematical definition.

Then we define a complexity bound for a theorem proving method M as follows:

If M is a theorem proving method then a function F is a complexity bound for M if for all unsatisfiable clause sets S, $T_M(S) \leq F(H(S))$.

The minimal such F can be considered to be the complexity of M. This is the approach taken in the proceedings paper. This approach can also be applied to incomplete methods such as UR resolution, where one only requires the inequality $T_M(S) \leq F(H(S))$ to hold for clause sets S whose unsatisfiability can be proved by M.

DPLL can be applied to first-order clause sets by enumerating ground instances of the clauses and periodically applying DPLL to test for satifiability. The proceedings paper applies the above search space analysis approach to resolution, UR-resolution, and a couple of versions of DPLL to attempt to draw some conclusions about their relative efficiency.

Summary

Here is a summary of our results:

- If one bounds the maximum literal size in a Herbrand set, DPLL seems to have an advantage over resolution.
- If one bounds the size of the Herbrand set as a whole, DPLL still seems to have an advantage but not as much.
- For resolution refutations having a small depth (which corresponds to a small number of literals in a Herbrand set T), resolution looks better.
- For sets S having unit refutations, resolution looks much better.
- We did not analyze methods besides resolution and DPLL.

As additional evidence that DPLL can sometimes be faster than resolution,

- SATCHMO has much the flavor of DPLL applied to an enumeration of ground instances, and in its day it frequently beat the best resolution provers.
- Another evidence is the good performance of instance-based methods on the function-free fragment in the annual TPTP competitions, and of DPLL for hardware verification applications.
- Some tests of mine showed DPLL to be superior to resolution on the pigeonhole problems.

As evidence for the superiority of resolution, one has the increasing performance of provers like Vampire, E, and Spass, among others.

Our analysis suggests to look for methods that are asymptotically efficient

- for small Herbrand sets,
- for sets having shallow resolution refutations,
- for sets having unit refutations, and
- for sets of clauses in which the maximum literal size in a Herbrand set is bounded.

This analysis does not include equality, however, which is easy to include in a resolution prover but possibly more difficult for an instance-based strategy. It also does not consider strategies other than resolution and DPLL.

Summary

So we have presented an approach to give an asymptotic analysis of the time complexity of theorem proving methods. Now we make some final comments about the state of the field.

Additional Comments

- Generating conjectures is also a fruitful area of research. This area is related to interpolation and automatic program verification, as well as being important for mathematical discovery in general.
- Model finding is another important area, which can be used to show that a formula is satisfiable. However, model finding is especially difficult, and for first-order logic is not even partially decidable.
- Another area of interest is using resolution provers as decision procedures for subsets of first-order logic. Maslov's method can also be used in this way.
- As another issue, what can we do to improve the status of theorem proving in the general computer science community, as well as the funding situation?
- It appears from personal experience that more and more people outside of computer science are becoming interested in automated deduction.
- Should we be thinking about the social consequences of theorem proving work and of artificial intelligence work in general?
- If computers learn to prove hard theorems, will they replace humans?
- What needs to happen for computer theorem provers to be able to prove hard theorems often, without human interaction?

Summary

We have considered the following topics related to the history and prospects for first-order theorem proving.

- History of theorem proving before and after resolution
- Significance of resolution
- Personal reflections
- Instance-based methods
- Term rewriting methods
- Model based methods
- Goal sensitivity
- Analysis of search space size
- A few final thoughts

These notes may be available on the internet soon.