01 Jan 1992

Proving Functionally Difficult Problems through Model Generation

Richard Rankin

Ralph W. Wilkerson
Missouri University of Science and Technology, ralphw@mst.edu

Recommended Citation

R. Rankin and R. W. Wilkerson, "Proving Functionally Difficult Problems through Model Generation," *Applied Computing: Technological Challenges of the 1990's*, pp. 526 - 529, Association for Computing Machinery, Jan 1992.
The definitive version is available at https://doi.org/10.1145/143559.150707

Proving Functionally Difficult Problems through Model Generation

Richard Rankin, Ralph Wilkerson

Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401

Abstract

Satchmo [MA88] is a theorem prover implemented in Prolog which attempts to provide satisfiability checking through model generation. This paper gives a brief introduction to SATCHMO and reports extensions to the original work which allow SATCHMO to solve problems previously considered to be finitely unprovable within the SATCHMO system. The specific problems are from [PE86, MO85, LU85] and were designed to convert simple propositional logic problems into functionally difficult first order problems. Although the benefits of using the SATCHMO system are many, the fact that it could not offer proofs for a set of problems provable in other systems is troublesome, especially in light of the claim that the system is correct and complete. This paper shows that SATCHMO can in fact, generate these proofs when the problems are correctly represented.

Introduction

SATCHMO, which is an acronym for SATisfiability CHecking by MOdel generation, is a theorem prover introduced by Manthey and Bry. For a full description of the program, see [MA88], and for corrections, [RA91]. It consists of a small set of short Prolog programs which prove or refute the satisfiability of a clause set using model generation techniques.

SATCHMO uses the concept of violated clauses to construct a model. A violated clause is a clause of the form $A_1, A_2,\ldots, A_n \rightarrow B$ where A_1 through A_n are provable in the program, but B is not present as a ground instance in the database. Such violated clauses are used to extend the database. Whenever the addition of a new clause allows the goal *false* to be proved, the database has become inconsistent, and backtracking is employed to select alternate paths. If all violated clauses can be added to the database without causing inconsistency, then a model of the clause set exists. If one or more of the violated clauses cannot be added without causing the database to be inconsistent, then the clause set is unsatisfiable.

The method used to construct SATCHMO clauses is as follows:

Convert the clauses, if necessary, to Skolemized Conjunction Normal Form. Clauses entirely composed of positive literals, $L_1 \vee L_2 \vee \ldots \vee L_n$ are re-written as true $\rightarrow L_1;L_2;\ldots;L_n$, where ';' is the disjunction operator in Prolog. Clauses consisting of entirely negative literals, $\neg L_1 \vee \neg L_2 \vee \ldots \vee \neg L_n$ are re-written as: $L_1, L_2, \ldots, L_n \rightarrow$ false. Clauses containing both positive and negative literals $\neg N_1 \vee \neg N_2 \vee \ldots \vee \neg N_n \vee P_1 \vee P_2 \vee \ldots \vee P_m$ are re-written as $N_1,N_2,\ldots N_n \rightarrow P_1; P_2; \ldots; P_m$. Using this method, negation never explicitly occurs.

It should be noted that satisfiability checking, in general, is undecidable. Therefore, there are cases where the generation of new clauses from the violated clauses will not terminate.

Figure 1 lists the clauses required for the SATCHMO system. In order to achieve soundness for satisfiability in SATCHMO, a technique called range-restriction of variables is also employed. For a discussion of range restriction, see [MA88]. The method of converting clauses to their range restricted form, as presented in the original SATCHMO paper, using A and C to represent literals in a clause is:

1) Every clause (true $\rightarrow$ C) that contains variables X_1 to X_n is transformed to $(dom(X_1),dom(X_2),\ldots,dom(X_n) \rightarrow C)$.
2) Every other clause (A $\rightarrow$ C) such that C contains variables Y_1 to Y_m not occurring in A is transformed to: $(A,dom(Y_1),dom(Y_2),\ldots,dom(Y_m) \rightarrow C)$.

```
satisfiable:-
        is_violated(C), !,
        satisfy(C),
        satisfiable.
satisfiable.

is_violated(C):-
        (A -> C),
. . . . . A, not C.
on_back(X).
on_back(X) :-
        X,!,fail.

satisfy(C):-
        component(X,C),
        asserta(X),
        on_back((retract(X)),
        not(false).

component(X,(Y;Z)):-
        !, (X=Y;
        component(X,Z)).
component(X,X).
```

SATCHMO Basic Program

Figure 1

3) For every constant c occurring in the clause set S, a clause (true → dom(c)) is added. If S does not contain any constants, an arbitrary artificial constant, 'a', is added, of the form:
(true → dom(a)).
4) For every n-ary function symbol *f* occurring in S, one adds a clause: $(dom(X_1),...,dom(X_n) \rightarrow dom(f(X_1,...,X_n))$.

The SATCHMO program listed in Figure 1 is not refutationally complete, due to the method that Prolog uses as a search strategy. A complete version of SATCHMO can be created with only a slight increase in code length, but at a substantial degradation in run-time performance. In practice, however, most problems are amenable to the more efficient techniques used in Figure 1. When disjunctions are present in the clauses created for SATCHMO, one also needs to change the 'component' rules. Refer to [MA88] for code and details on these items.

SATCHMO has been shown to be a versatile and fast theorem prover for many types of problems. See [MA88,RA91] for timing comparisons, tests and results.

Discussion of the Problem

Manthey and Bry list several problems which SATCHMO could not solve from the Pelletier collection of problems for testing automatic theorem provers [PE86]. The problems considered here are numbers 66-68, which are a collection of problems concerned with changing simple propositional logic problems into difficult function problems. Pelletier's problems are a re-statement of [MO85], and are discussed in [LU85]. In the original SATCHMO paper, the authors speculated that SATCHMO could not solve these problems due to the explosion of the domain rules. In effect, they felt that useful theorems asserted during execution were buried in the domain assertions. This, however, is not the case. The problem is not that the proliferation of new domain assertions prevents SATCHMO from finding a solution. The problem is that the terms generated will always be too deeply nested for SATCHMO to be able to advantageously use any of them in the proof by contradiction.

This discussion will center on Pelletier's problem 66, although problems 66 through 68 are similar, and can all be solved using the method outlined below. The clause set for Pelletier problem 66 is shown in Figure 2. The problems represented by these clause sets are relatively easily solved with many conventional theorem provers using non-ground inferences. For an example, see [LU85], in which problem 67 was solved, and the resulting theorem was used to solve problem 66.

1) dom(X), dom(Y) → t(i(X,i(Y,X)).
2)
dom(X),dom(Y),dom(Z)→t(i(i(X,i(Y,Z)),i(i(X,Y),i(X,Z))).
3) dom(X), dom(Y) → t(i(i(n(X),n(Y)),i(Y,X)).
4) t(i(X,Y)),t(X) → t(Y).
5) t(i(a,n(n(a)))) → false.
6) dom(X) → dom(n(X)).
7) dom(X), dom(Y) → dom(i(X,Y)).
8) true → dom(a).

SATCHMO Clause Set for Problem 66

Figure 2

The clause set is formulated by taking propositional logic theorems, treating the propositional letters as quantifiable variables, encoding operators as functions, and treating the expression "is a theorem" or "it is true" as a functor 't'. Modus Ponens becomes: $\forall_{xy}$ t(i(X,Y)),t(X) → t(Y). Clause 3 above, would be interpreted as (¬X → ¬Y) → (Y → X).

Consider clause number 5, which represents the negation of the conclusion. This is the clause used to refute the assertion that the clause set is satisfiable. In SATCHMO

terms, if *false* can be derived and is not discarded on backtracking through an alternate interpretation, the refutation has been found. If the goal t(i(a,n(n(a)))) is to be refuted, there must be a term unifiable with this goal generated and asserted during execution so that *false* may be derived. The question is actually dual in nature: 1) can the proper term be generated, and 2) can the proper term be asserted through a violated clause.

Upon inspection, it is obvious, that if a refuting term is to be constructed, it cannot come, in a single step, from clauses 1,2,3 and trivially, 5. The 'dom' rules cannot themselves generate appropriate terms, so clauses 6, 7, and 8 cannot be the direct generator. This leaves clause 4. Can clause 4 cause the assertion of t(i(a,n(n(a))))? The answer is in the affirmative with the substitution set { _/X, i(a,n(n(a)))/Y | _ is a "don't care" substitution}. This would cause the assertion of the clause required. Therefore, the proper term can be generated since the domain of Y includes {i(a,n(n(a))) } (this can easily be seen through inspection of the domain rules). When t(i(X,Y)) and t(X) have been proven, and t(Y) is not present in the database, the violated clause can assert t(Y). Since it appears that the substitution set for X is irrelevant, there should be a large, possibly infinite, number of substitutions for X such that the needed clause can be generated. The substitutions and resulting assertion of the term required, however, can never take place in any version of the SATCHMO system.

What is the clause used as a direct ancestor in generating one of the acceptable clauses required? Again, from inspection, one can see that it must have been clause 1 or 4, as no other appropriate clause would have generated such a term. The other possible clauses, 2 and 3, create new terms which are lexically too large, and thus, could not have supplied the needed instantiations. As will be shown shortly, Clause 4 cannot be the direct ancestor.

For clause 1 to have generated the term: t(_,i(a,n(n(a)))), it would have had substitution set { n(n(a))/X, a/Y} from its domain rules. This is the only case in which it would generate the i(Y,X) subterm to be i(a,n(n(a))), the term needed. Therefore, the X component is n(n(a)). It follows that to get the asserted clause from clause 1 to unify with the first term of clause 4, clause 1 used the domain rules, and became, in its instantiated form: dom(n(n(a))), dom(a) -> t(i(n(n(a)),i(a,n(n(a))))).
This term, when retrieved for use in clause 4, yields an instantiated form of: t(i(n(n(a)),i(a,n(n(a)))),t(n(n(a))) -> t(i(a,n(n(a)))).

The difficulty is that, although the substitutions needed in order to generate the desired clause are available, the clause cannot be asserted. The generation of the desired term can only be completed when both portions of the antecedent can be proved. The first part can be proved (obviously since it was just asserted into the database as a fact and retrieved as an instance of t(i(X,Y))). The second portion cannot be proved. There is no way to prove t(n(n(a))) in the clause set. No 't' predicates can be proven at *all* except those of the form t(i(_,_)) since only rules of that form appear on the right hand side except in rule number 4. As shown, this rule effectively strips away an 'i' functor, eventually resulting, after possibly repeated applications, in a term without an 'i' function symbol. And 't' terms without 'i' in them cannot be proven by the system. Therefore, any attempt to use Clause 4 whereby Y becomes instantiated to a subterm without 'i' is guaranteed to fail. In either version, SATCHMO does, in fact, reach this point in the computation, fail to prove t(n(n(a))), and continue on.

Discussion of the Solution

The solution to this problem turns out to be relatively simple. In an H-interpretation I, an atom set of a clause set S is constructed by using every element of the Herbrand Universe with every predicate. The H-interpretation is constructed by assigning each element of the atom set to {TRUE,FALSE}. All that is needed is to find a proper representation of this concept within SATCHMO.

The SATCHMO approach of applying every element in the current domain (generated through the 'dom' rules) to every SATCHMO rule at every level effectively builds the atom set for the set of clauses. The Herbrand Universe is constructed one level at a time through repeated applications of the 'dom' rules.

The idea of introducing the 't' functor was to complicate relatively simple theorem-proving problems. The purpose was to remove the problem from the form of : (((A → B) ∧ A) → B) with some H-interpretation, to that of (t(i(A,B)),t(A) → t(B)). In effect, the H-interpretation must now be represented within the 't' functor.

Let A = {$Atom_1$, $Atom_2$, ...$Atom_n$, ...} be the atom set of a set of clauses S. An H-interpretation I can be conveniently represented by a set I = {m_1,m_2,...m_n,...} in which m_j is either A_j or $\neg A_j$, for j = 1,2,.... This is interpreted as meaning, if m_j is $Atom_j$, then $Atom_j$ is assigned 'TRUE', otherwise (i.e. $\neg A_j$), A_j is assigned 'FALSE'. [CH73]

In this problem set (Pelletier 66-68), 't' is interpreted as meaning "it is true that", 'i(X,Y)' as meaning "X implies Y"; and 'n' as meaning "not". This is the meaning first suggested when these problems were developed. This means that 't' can be used to represent the H-interpretation. For example, t(a) maps to: "it is true that: a" ; t(i(a,b)) maps to "it is true that a implies b"; t(n(a)) maps to: "it is false that a", etc. Therefore, using the notation of convenience mentioned above, with a single predicate, 't', and a Herbrand Universe, U, for every element E of U, the H-interpretation contains either t(E) or t(n(E)). The predicate t(E) corresponds to: t(E) is assigned to "TRUE"; and t(n(E)) corresponds to: t(E) is assigned to "FALSE". In SATCHMO syntax, this results in the additional rule: dom(X) → t(X);t(n(X)).

Due to the processing methodology of SATCHMO, for any given X, either t(X) exclusively-or t(n(X)) can be considered 'TRUE' at any point using this rule. If t(X) is considered as 'TRUE' during a certain stage of processing, and failure occurs, backtracking may cause t(X) to be considered 'FALSE' and t(n(X)) to be

considered 'TRUE'. This is true for every element of the domain. In case of a failure node in the search tree, backtracking will consider both alternatives to every element of the domain before reporting ultimate failure. This backtracking, and alternative selection of paths, results in an exhaustive attempt to build an H-interpretation before failure is reported. Therefore, if failure is ultimately reported, S is false under all H-interpretations, which in turn, means S is unsatisfiable.

The introduction of the additional rule to the SATCHMO clause set allows SATCHMO to quickly find the refutation for the negation of the conclusion in problems 66-68.

Semantic and Syntactic Aspects

The explanation provided in the previous section uses a semantic interpretation to justify the addition of a domain rule effectively providing the law of the excluded middle. Although this does not change the meaning of the clause set in the logical sense, the necessity for adding such a rule in this clause set is based on a semantic interpretation of the clauses.

The generality of this solution for similar problems which SATCHMO might attempt, therefore, seems open to question. The underlying syntactic problem of the clause sets discussed above is as follows. Theorem provers which use nonground assertions, in the SATCHMO sense, contain variables which may unify with any elements of the Herbrand Universe. By examining [LU85], it is obvious that simple solutions exist for these clause sets, yet they rely on nonground clauses. A universally quantified variable in a 'regular' theorem prover automatically ranges over the entire Herbrand Universe. SATCHMO, however, must be able to actually generate all the elements of the Herbrand universe if it is to make use of such elements.

When there is a clause set C with a clause C_i such that $C_i = L_1,L_2,...L_n,p(X) \rightarrow p(Y)$, and the domain of X includes some function $f(X_1,...X_n)$, then there must exist a clause $C_{j, j<>i}$ such that $C_j = L_1,...L_n \rightarrow p(f(X_1,..X_n))$. If there is not such a clause C_j, then SATCHMO will not be able to generate the entire Herbrand universe. Since the X in C_i always ranges over all functions which appear, this means that every function symbol must appear on the right hand side of some SATCHMO clause. Although this problem is syntactically easy to identify, there is no obvious syntactic solution to the problem. The actual solution may require a semantic view of the clause set in order ot add the proper clauses needed to generate the entire Herbrand universe.

Conclusion

SATCHMO acquits itself well as a theorem proving mechanism. With the exception of a few of Pelletier's collection of problems which have not been attempted due to the sheer volume of clauses necessary, most of the Pelletier collection of problems have been successfully executed. With the addition of the appropriate clause, even problems 66-68, previously thought finitely unprovable within SATCHMO, can, in fact, be successfully executed to completion. These necessary additions, however, are currently determined by the semantics of the problem, even though the potential problem can be discovered syntactically.

Acknowledgements

The authors would like to express their appreciation to M. Insall for his suggestions and assistance with this work.

Bibliography

[CH73] Chang, C., and R. Char-Tung Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, Inc., 1973, Orlando.

[LU85] Lusk, E., and R. Overbeek, "Logic Problems", Association of Automated Reasoning Newsletter, No. 4, 1985, p. 7, Association of Automated Reasoning.

[MA88] Manthey, R. and F. Bry, "SATCHMO: a Theorem Prover Implemented in Prolog", Lecture notes in Computer Science Vol. 310, E. Lusk and R. Overbeek, eds., from the Ninth International Conference on Automated Deduction, pp. 415-434, Springer-Verlag, NY, 1988.

[MO85] Morgan, C., "Logic Problems", Association of Automated Reasoning Newsletter, No. 3, 1985 p.4, Association of Automated Reasoning.

[PE86] Pelletier, F.J., "Seventy-Five Problems for Testing Automatic Theorem Provers", pp. 191-216, Journal of Automated Reasoning, Volume 2, 1986.

[RA91] Rankin, R., and R. Wilkerson, "Verification of Results and Correction of SATCHMO", Technical Report CSc-91-1, Department of Computer Science, University of Missouri-Rolla, March, 1991.