INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen

Oettingenstraße 67 D–80538 München

Ludwig-Maximilians-Universität München

The Theorem Prover Satchmo: Strategies, Heuristics, and Applications (System Description)

Slim Abdennadher **François Bry**
Norbert Eisinger **Tim Geisler**

Présentation d'un prototype de recherche aux
IVèmes Journées Francophones de Programmation en Logique
JFPL'95, Dijon, 17, 18 et 19 mai 1995
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1995-3
Mai 1995

The Theorem Prover SATCHMO: Strategies, Heuristics, and Applications

Slim Abdennadher, François Bry,
Norbert Eisinger, Tim Geisler

Institut für Informatik
Universität München
Germany

Présentation d'un prototype de recherche aux
Journées Francophones de Programmation en Logique 1995
JFPL'95, Dijon, 17, 18 et 19 mai 1995

Résumé SATCHMO est un démonstrateur automatique de théorèmes pour la logique du premier ordre. Son principe de raisonnement, la *géneration de modèles*, est plus puissant que la *refutation* traditionelle. SATCHMO peut être vu comme un moteur de système expert capable de traiter les spécifications non seulement constructives et définies, mais aussi normatives et indéfinies. L'implémentation de SATCHMO en PROLOG est extremement concise, élégante et cependant efficace.

Abstract SATCHMO is an automated theorem prover for first-order predicate logic. Its reasoning paradigm, *model generation*, is more powerful than the traditional *refutation* paradigm. SATCHMO can be seen as an inference engine for expert systems capable of handling not only constructive and definite specifications, but also normative and indefinite ones. The implementation of SATCHMO in PROLOG is possible in a remarkably concise, elegant, and efficient way.

Introduction

SATCHMO (SATisfiability CHecking by MOdel generation) was introduced in [1] and initiated a new reasoning paradigm called *model generation*. This nonconventional approach has since been pursued by labs in Europe, Japan, and the US.

Like most expert systems or decision support systems, SATCHMO accepts a specification expressed as a finite set of rules, which will then be processed by an "inference engine". While classical approaches are restricted to constructive and definite specifications, SATCHMO can also handle normative and indefinite specifications.

A constructive specification determines the components of solutions to such an extent that their values can be computed (example: *"there are flights from München to Berlin every three hours, the first at 6.00 am, the last at 6.00 pm"*); a normative specification gives only certain conditions those values have to meet (example: *"there are five flights from München to Berlin, the departure of the first is not before 6.00 am, the arrival of the last not after 8.00 pm"*). A definite specification describes a unique solution (example: *"flights from München to Edinburgh have a stopover in Frankfurt"*); an indefinite specification allows the description of alternative solutions (example: *"flights from München to Edinburgh have a stopover either in Frankfurt or in London"*).

This extension can be regarded from three different points of view:
1. Power of expressiveness: SATCHMO can handle full first-order predicate logic and not only a fragment thereof.
2. Logic Programming: SATCHMO uses well-established logic programming techniques with two natural and rather simple extensions: integrity constraints and disjunctive logic programming.
3. Applicability: the increased expressiveness of SATCHMO specifications enables the representation of knowledge for a wider range of applications.

SATCHMO as an Inference Engine

Syntax of SATCHMO Rules

A SATCHMO rule has the form *antecedent* $\longrightarrow$ *consequent*, where *antecedent* is either *true* or a single atomic formula or a conjunction of atomic formulae, and *consequent* is either false (then the rule is called an *integrity constraint*) or a single atomic formula (*definite rule*) or a disjunction of atomic formulae (*indefinite rule*).

This so-called implicational form can be used uniformly for all kinds of formulae. For example, a fact is expressed as $true \longrightarrow fact$. Explicit negation is not normally allowed, but its effect can always be achieved by an integrity constraint.

In general the rules have to be *range-restricted*, which means that whenever a variable occurs in the consequent of a rule, the same variable must also occur in the antecedent of that rule. For most domains the "natural" specification results in rules that are range-restricted anyway. If they are not, they can always be turned into range-restricted rules by a simple syntactic transformation. Thus range-restrictedness is not a real syntactic restriction. On the other hand its positive impact on the inference engine is quite substantial.

Model Generation and Refutation

For the processing of rules, SATCHMO uses a database of derived facts. The basic algorithm is to find repeatedly a new consequence and to satisfy it. Whenever this process stops, the database represents (the positive part of) a model that satisfies all conditions expressed by the rules.

A new consequence is the consequent C of a rule instance $A \longrightarrow C$, for which the antecedent A matches the current database, but the consequent C does not. Here the range-restrictedness of the rules ensures that (one-way) matching rather than (two-way) unification with occur check is sufficient.

A new consequence that is a single atomic formula is satisfied by adding it to the database (which may potentially trigger further rule instances). This is the standard forward reasoning mechanism of rule-based systems.

If a new consequence is (the atomic formula) $false$, it cannot be satisfied, and backtracking becomes necessary. This is all that is required to handle integrity constraints and thus normative rules.

If a new consequence is a disjunction of several atomic formulae, it can be satisfied by adding any of these atomic formulae to the database. SATCHMO selects the first of them and creates choice points for the others for possible backtracking. This simple technique takes care of disjunctive rules. It works because the range-restrictedness guarantees that the database and any new consequences are always variable free.

If all backtracking attempts fail, no model exists and the conditions expressed by the rules are unsatisfiable. Thus one gets traditional refutation style theorem proving as a special case. But in contrast to this classical approach there is no obligation to add a negated conjecture for the system to be able to work.

Embedding in PROLOG

The algorithm above can be embedded in PROLOG as follows: the SATCHMO rules are represented as PROLOG facts with functor `--->`, the database is the PROLOG database that can be manipulated by `assert` and `retract`, and everything else is taken directly from PROLOG.

The resulting implementation is amazingly concise and simple. For example, the PROLOG clause `new_consequence(C) :- (A ---> C), A, not C.` defines how to find a new consequence. Matching, backtracking etc. require no explicit implementation, but alternative syntaxes, tracing etc. do.

Taking as much as possible from PROLOG accounts for much of the efficiency of SATCHMO. It is efficient not in spite of, but because of its simplicity.

SATCHMO Illustrated with Logical Puzzles

SATCHMO can most naturally be illustrated with logical puzzles. They are instructive because they usually exhibit complex problems in a compact form and because their understanding requires no special background knowledge about an application domain.

Illustrating the Principles of SATCHMO

A lot of logical puzzles are well-suited to illustrate how SATCHMO works. The example of the murder of Aunt Agatha [2] contains, among others, an indefinite rule $true \longrightarrow charles\ killed\ agatha \vee butler\ killed\ agatha \vee agatha\ killed\ agatha$. Its treatment amounts to a case analysis where the three suspected killers are investigated separately. Two of the cases lead to contradictions and one to a model. If one adds the negation of this case to the rule base, SATCHMO finds a refutation. This (classical) approach requires that the solution is known or guessed beforehand.

Meta-Level and Higher-Order Reasoning

The example of the flag design for a new state ends with: *"Three designs complying with all requirements are submitted. The committee rejects two designs, which have the colour of the lowest stripe in common."* Such meta-level conditions about relationships between different models cannot be expressed in classical approaches. SATCHMO generates all possible models first and then selects the appropriate one by a query ranging over these models.

A kind of higher-order reasoning is required for the knights and knaves examples in [3]. This is possible with SATCHMO rules such as $X\ says\ A,\ X\ tells\_truth \longrightarrow A$.

Basic Satchmo and Complete Version

The simple algorithm described earlier covers the finite cases. But if a rule can create infinitely many new consequences, the selection is not fair and completeness is endangered. There is a modification of SATCHMO that copes with such cases, too.

Optimisations

An obvious improvement of SATCHMO is to consider rules with consequent *false* before other rules. This one-level look-ahead for contradictions avoids many redundancies and can be implemented by adding a trivial PROLOG clause.

A more advanced selection is to order applicable rule instances by increasing size of their consequent. Then indefinite rules with many alternatives are considered later. In some examples this results in a speed-up factor of 700.

The forward reasoning method of basic SATCHMO corresponds to what in the terminology of deductive databases is called computation of the least fixpoint by the naïve method. There is also a so-called semi-naïve method, which works similarly but avoids unnecessary rule applications. The incorporation of this approach in SATCHMO further increases its performance.

Perspectives

Some of the planned further developments concern syntactic extensions. For example, it is useful to allow conjunctions in the consequent of a rule. Such a rule is equivalent to several rules with identical antecedent. The difference is that the antecedent would have to be evaluated only once rather than several times. For the same reason and in a very similar way disjunctions in antecedent will be allowed.

Other plans have to do with the search strategy. It is well-known that bidirectional search has advantages over pure forward or backward reasoning. Bidirectional search in SATCHMO can be achieved easily by expressing definite SATCHMO rules as PROLOG clauses instead. Thus, we already have the mechanism for combining the two directions. The reasearch problem is to find criteria when which direction is better.

Yet another topic is the introduction of auxiliary predicate symbols. This technique can increase the efficiency, but it is not straightforward to include in a way that is transparent to the user.

Among the application areas of SATCHMO there are database systems with integrity constraints, deductive databases and information systems (e.g. construction of schedules), legal applications (e.g. separation regulations for the transportation of dangerous goods), or marketing applications (e.g. target group determination). An application in the domain of deductive databases is being developed in cooperation with an industrial partner.

References

[1] Rainer Manthey and François Bry. Satchmo: a theorem prover implemented in prolog. In Ewing Lusk and Ross Overbeek, editors, *Proceedings CADE-88*, volume 310 of *LNCS*, pages 415–434. Springer-Verlag, 1988.

[2] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986.

[3] Ramond Smullyan. *What is the Name of this Book?* Prentice-Hall, 1978.