INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67 D–80538 München

Ludwig
Maximilians–
Universität
München

Satchmo: Minimal Model Generation and Compilation (System Description)

Thomas Brüggemann François Bry
Norbert Eisinger Tim Geisler
Sven Panne Heribert Schütz
Sunna Torge Adnan Yahya

Présentation d'un prototype de recherche aux
Journées Francophones de Programmation en Logique et
programmation par Contraintes 1996
JFPLC'96, Clermont-Ferrand, France (5.–7. June 1996)
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1996-5
Mai 1996

Satchmo: Minimal Model Generation and Compilation (System Description)

Thomas Brüggemann François Bry Norbert Eisinger
Tim Geisler Sven Panne Heribert Schütz Sunna Torge
Adnan Yahya[1]

Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstraße 67
D–80538 München
{brueggem,bry,eisinger,geisler,panne,hschuetz,torge,yahya}@informatik.uni-muenchen.de

---

Résumé: *Satchmo est un démonstrateur automatique de théorèmes pour la logique du premier ordre implémenté en Prolog. Son principe de raisonnement, la* génération de modèles*, est plus puissant que la* refutation *traditionelle. Il rend possible une technique nouvelle et performante de génération de modèles de Herbrand minimaux qui évite la construction de modèles non-minimaux qui devrait être éliminés dans une phase ultérieure. Il favorise également le dévelopement de plusieurs techniques avancées d'optimisation qui permettent des performances très compétitives sur les jeux de tests courants.*

Mots-clés: *génération de modèles minimaux, compilation, incrementalité*

Abstract: *Satchmo is an automated theorem prover for first-order predicate logic implemented in Prolog. Its reasoning paradigm,* model generation*, is more powerful than the traditional* refutation *paradigm. It enabled the development of a novel and efficient technique to compute minimal Herbrand models, which prevents the generation of non-minimal models that would later have to be filtered out in a post-processing step. It also encouraged the development of several advanced efficiency enhancing techniques that result in a highly competitive performance on standard benchmark problems.*

Keywords: *minimal model generation, compilation, incrementality*

[1] Also: Electrical Engineering Department, Birzeit University, Birzeit, Palestine

1 Introduction to Satchmo

Satchmo (SATisfiability CHecking by MOdel generation) [4, 1] is based on the *model generation* reasoning paradigm. This un-conventional approach has been refined in recent years and applied to various problems (among others as an expert system engine for diagnosis, software verification and synthesis) in several labs in Europe, Japan, and the US.

Like most expert systems or decision support systems, Satchmo accepts a specification expressed as a set of rules, which is then processed by an "inference engine". A Satchmo rule has the form $A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$ with atomic formulae A_i and B_j. The rule body may be the empty conjunction, which represents truth. The rule head is either the empty disjunction, i.e., falsity (then the rule is an *integrity constraint*), or a single atomic formula (in which case the rule is *definite*), or a proper disjunction of atomic formulae (then the rule is *indefinite*).

The ability to handle integrity constraints and indefinite rules makes Satchmo significantly more powerful than traditional rule based systems. In fact it covers full first-order predicate logic, since the rules are actually clauses in implication form.

We report on two new developments, one improving the functionality and the other improving the efficiency. The first one is a Satchmo version that generates exactly the *minimal* models of a specification and no others. The second one comprises several partly independent techniques, most notably incrementality and compilation. Their combination results in a highly competitive performance on standard benchmark problems.

An Overview of the Inference Mechanism

A Satchmo specification is a set of rules in the implication form described above. Rules are required to be range-restricted, i.e., each head variable must also occur in the body. Range-restriction can always be achieved by a transformation which requires an extension of the language, but preserves models in a similar way as Skolemization does [2].

Satchmo implements the PUHR-tableaux proof method [2]. Branches of a PUHR tableau are expanded by two kinds of steps: A "positive unit hyper-resolution" step is applicable with a rule, if all atomic formulae in the rule body simultaneously match atomic formulae of the branch; the corresponding instance of the rule head (whose variable-freeness is ensured by range-restriction) is then appended to the branch. If the head is empty, the branch is closed. A "splitting" step (usually called β expansion for tableaux [3]), extends a branch containing a proper disjunction by adding a sub-branch for each atomic formula of the disjunction.

If a branch is saturated by these steps and is not closed, its atomic formulae represent a Herbrand model of the specification. All minimal Herbrand models (but possibly some non-minimal Herbrand models, too) occur as branches in every saturated PUHR tableau. Therefore, a saturated PUHR tableau is closed iff the specification is unsatisfiable.

The implementation of this procedure [4] is amazingly concise, simple, and efficient, since it reuses rather than reimplements Prolog capabilities: Tableaux are mapped to Prolog search trees in a rather direct way, and Satchmo's atomic formulae are represented as Prolog facts.

2 Minimal Model Generation

Like traditional tableaux-based model generators, Satchmo may produce non-minimal and duplicate, i.e., redundant models. If a model generator is used to refute a specification, non-minimal models are undesirable because they unnecessarily increase the search space. If it is used to find models, in applications such as diagnosis or program synthesis, non-minimal models are also undesirable, because they do not convey any information beyond what the minimal models provide, anyway.

An a-posteriori elimination of redundant models is tedious and potentially time consuming. The presentation will demonstrate a modification of Satchmo that generates minimal Herbrand models of specifications [2]. The procedure is optimal in the sense that it generates each minimal model only once and rejects non-minimal models as soon as possible, in general before their complete construction. First measurements on the demonstrated implementation point to the efficiency of the procedure.

2.1 Complement Splitting

In order to avoid some redundant models, we slightly modify the splitting step. Assume a PUHR step has appended $B_1 \vee B_2$ to a branch. We split by adding B_2 as the second sub-branch, as before. The first sub-branch, however, contains B_1 *and in addition* $\neg B_2$. Branches containing an atomic formula and its negation are now considered closed, too.

This so-called *complement splitting* prunes the search space. Furthermore, it guarantees that a generated model M is never preceded by a model which is a superset of M. Thus, the first generated model is minimal. Complement splitting can nicely be integrated into Satchmo [2].

2.2 The Minimal Model Generation Procedure

Complement splitting is not always sufficient to prevent the generation of all non-minimal models. In order to prune the remaining redundant models as early as possible, another refinement is needed.

Complement splitting already excludes cases where a generated model is a superset of model generated *later*. There only remains to avoid generation of models which are supersets of *previously* generated models. This can easily be done "on the fly" by adding integrity constraints which exclude the already generated minimal models.

This procedure is sound and complete for minimal model generation in the sense that it generates exactly the minimal Herbrand models of a specification [2]. A further result, established in the full version of [2], ensures termination for specifications without infinite minimal models: such specifications always have finitely many (finite) minimal models.

3 Efficiency Enhancing Techniques

Several simple but powerful techniques have been developed for enhancing the efficiency of Satchmo. For a more detailed description of these techniques see [5]. The techniques fit elegantly in Satchmo's approach of reusing Prolog capabilities rather than reimplementing them.

All of these techniques can be implemented in any standard Prolog system. They have been evaluated with a large collection of standard benchmark problems for automated theorem provers. The results compare very favorably with those of the most powerful existing systems [5].

3.1 Incremental Evaluation

A major drawback of the original implementation of Satchmo is that in later tableau expansion steps it repeats work already done in earlier steps. The search for a rule usable in a hyper-resolution step does not take into account that many rule instances are not affected by atomic formulae recently appended to the branch. The incremental version of Satchmo utilizes information about recently added atomic formulae to improve the search for applicable rule instances in a way similar to semi-naïve evaluation in deductive databases. It has been implemented as a meta-interpreter program which is only slightly more complex than the original version.

3.2 Compilation

The additional interpretation level introduced by Satchmo is avoided by compiling a set of Satchmo rules into a Prolog program, which is then compiled by the Prolog system. The compilation of a set of Satchmo rules into a Prolog program can be seen as partial evaluation. However, instead of relying on a standard partial evaluator, a specialized compiler has been developed. This specialized compiler turns out to be far more efficient than a general-purpose partial evaluator, and is able to perform more far-reaching optimizations that are specific to Satchmo's model generation algorithm. Besides the obvious speedup for the individual deduction steps, advantages of compilation include the possibility to perform some decisions and transformations at compile time, thus improving the runtime performance.

3.3 Fairness

Fair selection of rule instances is needed for soundness and refutation completeness of a model generation method. Efficient implementation of fairness turns out to be easier and more elegant with the incremental version of Satchmo than with the non-incremental initial program.

A fair selection of rules is often ensured by relying on a purely breadth-first search strategy. This approach, however, is in general inefficient. A significantly more efficient strategy has been developed, which uses breadth-first search only when needed and relies on depth-first search otherwise. Thus, the developed system interleaves breadth-first and depth-first search within a single process.

4 Further Developments

Incrementality, compilation, and complement splitting are pairwise orthogonal techniques and can therefore be chosen independently for a Satchmo implementation, whereas the efficient implementation of fairness depends on incrementality and compilation. While complement splitting could be elegantly introduced into the compiled Satchmo system, compilation of the full minimal model generation procedure still remains to be investigated.

One of the motivations for the development of the minimal model generation procedure was the need for non-redundant models in some applications. In fact, even more is needed. Practical applications often require *finite models*, i.e., models with a finite domain. Thus the generator should construct such models rather than Herbrand models. The existence of finite models for a given specification—also called finite satisfiability—is semi-decidable, which suggests a semi-decision procedure testing finite satisfiability of specifications. Being based upon a particularly efficient model generation method, the algorithm underlying the Satchmo programs is a good starting point for developing such a procedure. Our group has started research activities on this subject.

Many applications do not rely on classical logic but on nonclassical logics such as *modal* and *temporal logics*. Tableaux methods are especially appropriate for designing theorem provers for such logics. Our group has started research on adapting the Satchmo approach and its simple implementation in Prolog to some modal and temporal logics.

The Satchmo paradigm has turned out to be not only powerful, but also quite fertile. Because it can be implemented with rather simple and short Prolog programs, the Satchmo paradigm is also well-suited for experimentation.

References

[1] Abdennadher, S., Bry, F., Eisinger, N., and Geisler, T. The theorem prover SATCHMO: Strategies, heuristics, and applications. In *IVèmes Journées Francophones de Programmation en Logique* (1995), Teknea, pp. 349–352.

[2] Bry, F., and Yahya, A. Minimal model generation with positive unit hyper-resolution tableaux. In *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods* (1996), Springer LNCS. Full version: http: //www.informatik.uni-muenchen.de/pms/publikationen/berichte/minimal-models.ps.gz.

[3] Fitting, M. *First-Order Logic and Automated Theorem Proving.* Springer, 1987.

[4] Manthey, R., and Bry, F. SATCHMO: A theorem prover implemented in Prolog. In *9th Int. Conf. on Automated Deduction (CADE)* (1988), Springer LNCS 310, pp. 415–434.

[5] Schütz, H., and Geisler, T. Efficient model generation through compilation. In *13th Int. Conf. on Automated Deduction (CADE)* (1996), Springer LNCS.