Integrating a SAT Solver with an LCF-style Theorem Prover*

Tjark Weber
Institut für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748 Garching b. München, Germany
webertj@in.tum.de

Abstract

This paper describes the integration of a leading SAT solver with Isabelle/HOL, a popular interactive theorem prover. The SAT solver generates resolution-style proofs for (instances of) propositional tautologies. These proofs are verified by the theorem prover. The presented approach significantly improves Isabelle's performance on propositional problems, and furthermore exhibits counterexamples for unprovable conjectures.

1 Introduction

Interactive theorem provers like PVS [19], HOL [10] or Isabelle [20] traditionally support rich specification logics. Proof search and automation for these logics however is difficult, and proving a non-trivial theorem usually requires manual guidance by an expert user. Automated theorem provers on the other hand, while often designed for simpler logics, have become increasingly powerful over the past few years. New algorithms, improved heuristics and faster hardware allow interesting theorems to be proved with little or no human interaction, sometimes within seconds.

By integrating automated provers with interactive systems, we can preserve the richness of our specification logic and at the same time increase the degree of automation [22]. This is an idea that goes back at least to the early nineties [14]. However, to ensure that a potential bug in the automated prover does not render the whole system unsound, theorems in Isabelle, like in other LCF-style [8] provers, can be derived only through a set of core inference rules. Therefore it is not sufficient for the automated prover to return whether a formula is provable, but it must also generate the

*This work was supported by the PhD program Logic in Computer Science of the German Research Foundation.

actual proof, expressed (or expressable) in terms of the interactive system's inference rules.

Formal verification is an important application area of interactive theorem proving. Problems in verification can often be reduced to Boolean satisfiability (SAT), and recent SAT solver advances have made this approach feasible in practice. Hence the performance of an interactive prover on propositional problems may be of significant practical importance. In this paper we describe the integration of zChaff [17], a leading SAT solver, with the Isabelle/HOL [18] prover. We show that using zChaff to prove theorems of propositional logic dramatically improves Isabelle's performance on this class of formulas. Furthermore, while Isabelle's previous decision procedures simply fail on unprovable conjectures, zChaff is able to produce concrete counterexamples.

The next section describes the integration of zChaff with Isabelle/HOL in more detail. In Section 3 we evaluate the performance of our approach, and report on experimental results. Related work is discussed in Section 4. Section 5 concludes this paper with some final remarks and points out directions for future research.

2 System Description

To prove a propositional tautology ϕ in the Isabelle/HOL system with the help of zChaff, we proceed in several steps. First ϕ is negated, and the negation is converted into an equivalent formula ϕ^* in conjunctive normal form. ϕ^* is then written to a file in DIMACS CNF format [6], the input format supported by zChaff (and many other SAT solvers). zChaff, when run on this file, returns either "unsatisfiable", or a satisfying assignment for ϕ^*.

In the latter case, the satisfying assignment is displayed to the user. The assignment constitutes a counterexample to the original conjecture. When zChaff returns "unsatisfiable" however, things are more complicated. If we have confidence in the SAT solver, we can simply trust its result and accept ϕ as a theorem in Isabelle. The theorem is tagged with an "oracle" flag to indicate that it was proved not through Isabelle's own inference rules, but by an external tool. In this scenario, a bug in zChaff could allow us to derive inconsistent theorems in Isabelle/HOL.

The LCF-approach instead demands that we verify zChaff's claim of unsatisfiability within Isabelle/HOL. While this is not as simple as the validation of a satisfying assignment, the increasing complexity of SAT solvers has before raised the question of support for independent verification of their results, and in 2003 zChaff has been extended by L. Zhang and S. Malik [27] to generate resolution-style proofs that can be verified by an independent

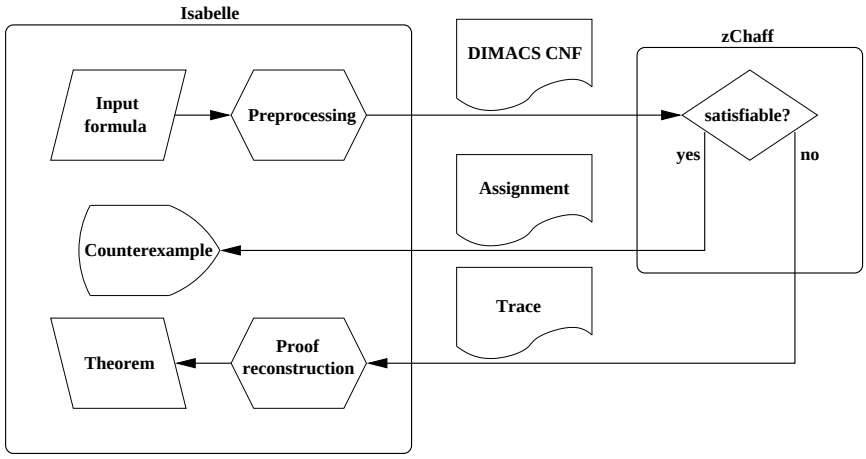

Figure 1: System Architecture

checker.[1] Hence our main task boils down to using Isabelle/HOL as an independent checker for the resolution proof found by zChaff.

zChaff stores this proof in a text file that is read in by Isabelle, and the individual resolution steps are replayed in Isabelle/HOL. Section 2.1 describes the necessary preprocessing of the input formula, and details of the proof reconstruction are explained in Section 2.2. The overall system architecture is shown in Figure 1.

2.1 Preprocessing

Isabelle/HOL offers higher-order logic (on top of Isabelle's meta logic), whereas zChaff only supports formulas of propositional logic in conjunctive normal form. Therefore the (negated) input formula ϕ must be preprocessed before it can be passed to zChaff.

First connectives of the meta logic, namely meta implication ($\Longrightarrow$) and meta equivalence ($\equiv$), are replaced by the corresponding HOL connectives $\longrightarrow$ and $=$. This is merely a technicality. Then the Boolean constants True and False are eliminated from ϕ, as are implication, $\longrightarrow$, and equivalence, $=$. The only remaining connectives are conjunction, disjunction, and negation. Finally ϕ is converted into negation normal form, and then into conjunctive normal form (CNF). The naive conversion currently implemented may cause an exponential blowup of the formula, but a Tseitin-style encoding [25] could easily be used instead (introducing existentially quantified variables at the

[1] This is the very reason why we chose zChaff as the SAT solver to be integrated with Isabelle/HOL. Extending other DPLL-based solvers with proof-generating capabilities should be relatively simple [27], but despite some work in this direction [7], zChaff, to our knowledge, is currently the only proof-generating SAT solver that is publicly available.

HOL level, cf. [9]). Quantified subformulas of ϕ are treated as atomic.

Note that it is not sufficient to convert ϕ into an equivalent formula ϕ' in CNF. Rather, we have to *prove* this equivalence inside Isabelle/HOL. The result is not a single formula, but a theorem of the form $\phi = \phi'$. Our main workhorse for the construction of this theorem is a generic function `thm_of`, proposed by A. Chaieb and T. Nipkow [5]:

```
thm_of decomp t =
  let
    (ts, recomb) = decomp t
  in recomb (map (thm_of decomp) ts)
```

It takes a decomposition function `decomp` of type $\alpha \to \alpha\ list \times (\beta\ list \to \beta)$ and a problem `t` of type α, decomposes `t` into a list of subproblems `ts` and a recombination function `recomb`, solves the subproblems recursively, and uses `recomb` to combine the recursive solutions into an overall solution. In our setting, `t` is a formula, `decomp` will look at its syntactic structure (i.e. its outmost connectives), and β is the type of theorems. When `t` is just a literal, we use reflexivity of $=$ to derive `t = t`. Tautologies like $\neg P = P' \Longrightarrow \neg Q = Q' \Longrightarrow \neg(P \wedge Q) = P' \vee Q'$ (which are easily provable in Isabelle/HOL) are used to implement the recombination function. This tautology, for example, corresponds to one of de Morgan's laws, and is part of the conversion into negation normal form. All of the conversions mentioned above can then be handled with proper instantiations for `decomp`.

zChaff treats clauses as sets of literals, making implicit use of associativity, commutativity and idempotence of disjunction. Therefore some further preprocessing is necessary, aside from conversion to CNF. Using associativity of conjunction and disjunction, we rewrite ϕ' into an equivalent CNF formula with unnecessary parentheses removed. In a second step, we remove duplicate literals, so that every clause contains each literal at most once. Finally, using $P \vee \neg P = \text{True}$, we remove every clause that contains both a literal and its negation. Each preprocessing step yields an equivalence theorem that was proved in Isabelle/HOL, and transitivity of $=$ allows us to combine these theorems into a single theorem $\phi = \phi^*$, where ϕ^* is the final result of our conversion. Unless ϕ^* is already syntactically equal to True or False, it is then written to a file in DIMACS CNF format, and zChaff is invoked on this file.

2.2 Proof Reconstruction

When zChaff returns "unsatisfiable", it also generates a resolution-style proof of unsatisfiability and stores the proof in a text file [27]. This file consists of three sections: clauses derived from the original problem by resolution, the values of variables implied by these clauses, and a conflict clause,

i.e. a derived clause in which all literals are false. The text file is parsed by Isabelle, and the relevant information contained in it is used to reconstruct the unsatisfiability proof in Isabelle/HOL. Proof reconstruction is based on two simple functions: a function `prove_clause` that uses resolution to derive new theorems of the form $\phi^* \longrightarrow c$ from existing theorems $\phi^* \longrightarrow c_1$, ..., $\phi^* \longrightarrow c_n$ (where c and c_1, ..., c_n are single clauses), and another function `prove_literal` that proves $\phi^* \longrightarrow l$ (where l is a single literal) from l's antecedent $\phi^* \longrightarrow c$. Here c must be a clause that contains l, and for all other literals l' in c a theorem of the form $\phi^* \longrightarrow \neg l'$ must be provable. These functions correspond to the first and second section, respectively, of the text file generated by zChaff.

```
prove_clause clause_id =
  resolution (map prove_clause (resolvents_of clause_id))

prove_literal var_id =
  let
    th_ante = prove_clause (antecedent_of var_id)
    var_ids = filter (fn i => i <> var_id)
                     (var_ids_in_clause th_ante)
  in resolution (th_ante :: map prove_literal var_ids)
```

`resolvents_of` and `antecedent_of` are auxiliary functions that rely on the information provided by zChaff to return the IDs of a clause's resolvents or a variable's antecedent, respectively. `var_ids_in_clause`, when applied to a theorem of the form $\phi^* \longrightarrow c$, returns the IDs of variables occurring in clause c.

Resolution between two clauses c_1 and c_2 is always performed with the first literal in c_1 that occurs with opposite polarity in c_2. Note that `resolution` must internally use associativity and commutativity of disjunction to reorder clauses, and idempotence to ensure that the resulting clause contains each literal at most once.

Proof reconstruction proceeds in three steps. First the conflict clause is proved by a call to `prove_clause`. Then `prove_literal` is called for every literal in the conflict clause, to show that the literal must be false. Finally resolving the conflict clause with these negated literals yields the theorem $\phi^* \longrightarrow$ False.

For efficiency reasons, the actual implementation is slightly different from what is shown above. Some clauses that were derived by zChaff may be used many times during the proof, while others are perhaps not used at all. Theorems that were proved once are therefore stored in two arrays (one for clauses, one for literals), and simply looked up – rather than reproved – should they be needed again. Hence our implementation is not purely functional.

2.3 A Simple Example

In this section we illustrate the proof reconstruction using a small example. Consider the following input formula

$$\phi \equiv (\neg v1 \vee v2) \wedge (\neg v2 \vee \neg v3) \wedge (v1 \vee v2) \wedge (\neg v2 \vee v3).$$

Since ϕ is already in conjunctive normal form, preprocessing simply yields the theorem $\phi = \phi$. The corresponding DIMACS CNF file, aside from its header, contains one line for each clause in ϕ:

```
-1  2 0
-2 -3 0
 1  2 0
-2  3 0
```

zChaff easily detects that this problem is unsatisfiable, and creates a text file with the following data:

```
CL: 4 <= 2 0
VAR: 2 L: 0 V: 1 A: 4 Lits: 4
VAR: 3 L: 1 V: 0 A: 1 Lits: 5 7
CONF: 3 == 5 6
```

This tells Isabelle that first a new clause (with ID 4) is derived by resolving clause 2, $v1 \vee v2$, with clause 0, $\neg v1 \vee v2$. The first variable that occurs both positively and negatively in clause 2 and clause 0 is $v1$; this variable is eliminated by resolution.

Now the value of variable 2 (`VAR: 2`) can be deduced from clause 4 (`A: 4`). $v2$ must be true (`V: 1`). Clause 4 contains only one literal (`Lits: 4`), namely $v2$ (since $4 \div 2 = 2$), occuring positively (since $4 \bmod 2 = 0$). This decision is made at level 0 (`L: 0`), before any decision at higher levels.

Likewise, the value of variable 3 can then be deduced from clause 1, $\neg v2 \vee \neg v3$. $v3$ must be false (`V: 0`).

Finally clause 3 is our conflict clause. It contains two literals, $\neg v2$ (since $5 \div 2 = 2$, $5 \bmod 2 = 1$) and $v3$ (since $6 \div 2 = 3$, $6 \bmod 2 = 0$). But we already know that both literals must be false, so this clause is not satisfiable.

Note that information concerning the level of decisions, the actual value of variables, or the literals that occur in a clause is redundant in the sense that it is not needed by Isabelle to validate zChaff's proof. This information can always be reconstructed from the original problem.

Also note that the actual proof reconstruction in Isabelle proceeds backwards, starting from the conflict clause. This has the advantage that resolution steps that are recorded by zChaff, but not needed to show unsatisfiability are not replayed in Isabelle. In our example, first clause 3

is proved (which is trivial, since it is one of the original clauses). Then `prove_literal 2` is called, immediately leading to a call of `prove_clause 4`. Clause 4 is proved by resolving clause 2 with clause 0. Now `prove_literal 3` is called, and since clause 1, the antecedent of variable 3, also contains variable 2, this leads to another call of `prove_literal 2`. Finally a contradiction can be derived by resolving the conflict clause, clause 3, with the results of `prove_literal 2` and `prove_literal 3`.

3 Evaluation

Isabelle/HOL offers three major automatic proof procedures: *auto*, which performs simplification and splitting of a goal, *blast* [21], a tableau-based prover, and *fast*, which searches for a proof using standard Isabelle inference. Details can be found in [18]. We compared the performance of our approach to that of Isabelle's existing proof procedures on all 42 problems contained in version 2.6.0 of the TPTP library [24] that have a representation in propositional logic. The problems were negated, so that unsatisfiable problems became provable. All benchmarks were run on a machine with a 3 GHz Intel Xeon CPU and 1 GB of main memory.

19 of these 42 problems are rather easy, and were solved in less than a second each by both the existing procedures and the SAT solver approach. Table 1 shows the times in seconds required to solve the remaining 23 problems. An **x** indicates that the procedure ran out of memory or failed to terminate within an hour.

Proof reconstruction in Isabelle/HOL is currently several orders of magnitude slower than proof verification with an external checker [27] written in C++. While there may still be potential for optimization in the Isabelle/HOL implementation, profiling indicates that this difference must mainly be attributed to the data structures and functions provided by Isabelle's LCF-style kernel, which are not geared towards clausal reasoning.

The SAT solver approach dramatically outperforms the automatic proof procedures that were previously available in Isabelle/HOL. The other procedures combined solved only 8 of the harder problems. Running times between the different procedures vary wildly, and they all fail to terminate for the 7 satisfiable (i.e. unprovable) problems. In contrast, the SAT solver approach solves *all* problems, takes less than two seconds on all but two problems, and provides actual counterexamples for the unprovable problems. Furthermore, the rightmost column of Table 1 already shows the total (combined) time for the invocation of zChaff and the following proof reconstruction in Isabelle/HOL. zChaff alone terminates after a usually negligible fraction of this time, at which point a definite answer can already be displayed to the user – a feature that is particularly useful in our interactive setting.

Problem	Status	auto	blast	fast	zChaff
MSC007-1.008	unsat.	x	x	x	726.5
NUM285-1	sat.	x	x	x	0.2
PUZ013-1	unsat.	0.5	x	5.0	0.1
PUZ014-1	unsat.	1.4	x	6.1	0.1
PUZ015-2.006	unsat.	x	x	x	10.5
PUZ016-2.004	sat.	x	x	x	0.3
PUZ016-2.005	unsat.	x	x	x	1.6
PUZ030-2	unsat.	x	x	x	0.7
PUZ033-1	unsat.	0.2	6.4	0.1	0.1
SYN001-1.005	unsat.	x	x	x	0.4
SYN003-1.006	unsat.	0.9	x	1.6	0.1
SYN004-1.007	unsat.	0.3	822.2	2.8	0.1
SYN010-1.005.005	unsat.	x	x	x	0.4
SYN086-1.003	sat.	x	x	x	0.1
SYN087-1.003	sat.	x	x	x	0.1
SYN090-1.008	unsat.	13.8	x	x	0.5
SYN091-1.003	sat.	x	x	x	0.1
SYN092-1.003	sat.	x	x	x	0.1
SYN093-1.002	unsat.	1290.8	16.2	1126.6	0.1
SYN094-1.005	unsat.	x	x	x	0.8
SYN097-1.002	unsat.	x	19.2	x	0.2
SYN098-1.002	unsat.	x	x	x	0.4
SYN302-1.003	sat.	x	x	x	0.4

Table 1: Running times (in seconds) for TPTP problems

4 Related Work

Michael Gordon has implemented *HolSatLib* [9], a library which is now part of the HOL 4 theorem prover. This library provides functions to convert HOL 4 terms into CNF, and to analyze them using a SAT solver. In the case of unsatisfiability however, the user only has the option to trust the external solver. No proof reconstruction takes place, "since there is no efficient way to check for unsatisfiability using pure Hol98 theorem proving" [9]. A bug in the SAT solver could ultimately lead to an inconsistency in HOL 4.

Perhaps closer related to our work is the integration of automated first-order provers, recently further explored by Joe Hurd [12, 13] and Jia Meng [15, 16]. Proofs found by the automated system are either verified by the interactive prover immediately [12], or translated into a proof script that can be executed later [16]. The main focus of their work however is on the necessary translation from the interactive prover's specification language to first-order logic. In contrast our approach is so far restricted to instances of

propositional tautologies, but it avoids difficult translation issues, and uses a SAT solver, rather than a first-order prover.

A custom-built SAT solver has been integrated with the CVC Lite system [3] by Clark Barrett et al. [4]. While this solver produces proofs that can be checked independently, our work shows that it is possible to integrate an existing, highly efficient solver with an LCF-style prover: the information provided by recent versions of zChaff is sufficient to produce a proof object in a theorem prover, no custom-built solver is necessary.

Other applications of SAT solvers in the context of theorem proving include SAT-based decision procedures (e.g. [2, 23]), as well as SAT-based model generation techniques (e.g. [1, 26]). These applications again require involved translations, and a correctly implemented SAT solver is usually taken for granted.

5 Conclusions and Future Work

Our results show that the zChaff-based tactic is clearly superior to Isabelle's built-in tactics for propositional formulas. With the help of zChaff, many formulas that were previously out of the scope of Isabelle's built-in tactics can now be proved – or refuted – automatically, often within seconds. Isabelle's applicability as a tool for formal verification, where large propositional problems occur in practice, has thereby improved considerably.

However, Isabelle's performance is still not sufficient for problems with thousands of clauses, like some of those found in the SATLIB library [11]. Their sheer size currently does not permit an efficient treatment in Isabelle/HOL. Further work is necessary to investigate if this issue can be resolved by relatively minor optimizations to Isabelle's kernel, or if an extension of the LCF-style kernel with optimized data structures and algorithms for propositional logic is more promising.

The approach presented in this paper has applications beyond propositional reasoning. The decision problem for (fragments of) richer logics can be reduced to SAT [2, 23]. Consequently, proof reconstruction for propositional logic can serve as a foundation for proof reconstruction for other logics. Based on our work, one only needs a proof-generating implementation of the reduction to integrate the whole SAT-based decision procedure with an LCF-style theorem prover.

Acknowledgments The author would like to thank Sharad Malik and Zhaohui Fu for their help with zChaff, and Tobias Nipkow and the anonymous referees for their valuable suggestions.

References

[1] Pranav Ashar, Malay Ganai, Aarti Gupta, Franjo Ivancic, and Zijiang Yang. Efficient SAT-based bounded model checking for software verification. In *1st International Symposium on Leveraging Applications of Formal Methods, ISOLA 2004*, 2004.

[2] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT based approach for solving formulas over Boolean and linear mathematical propositions. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 195–210, Copenhagen, Denmark, July 2002. Springer.

[3] Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV 2004)*, Boston, Massachusetts, USA, July 2004.

[4] Clark Barrett, Sergey Berezin, and David L. Dill. A proof-producing Boolean search engine. In *Proceedings of the Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR 2003)*, Miami, Florida, USA, July 2003.

[5] Amine Chaieb and Tobias Nipkow. Generic proof synthesis for Presburger arithmetic. Technical report, Technische Universität München, October 2003.

[6] DIMACS satisfiability suggested format, 1993. Available online at ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc.

[7] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Design, Automation and Test in Europe (DATE 2003)*, pages 10886–10891. IEEE Computer Society, 2003.

[8] M. J. C. Gordon. From LCF to HOL: a short history. In G. Plotkin, Colin P. Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*. MIT Press, 2000.

[9] M. J. C. Gordon. HolSatLib documentation, version 1.0b, June 2001. Available online at http://www.cl.cam.ac.uk/~mjcg/HolSatLib/HolSatLib.html.

[10] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[11] Holger H. Hoos and Thomas Stützle. SATLIB: An online resource for research on SAT. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT 2000*, pages 283–292. IOS Press, 2000. Available online at http://www.satlib.org/.

[12] Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs '99*, volume 1690 of *Lecture Notes in Computer Science*, pages 311–321, Nice, France, September 1999. Springer.

[13] Joe Hurd. An LCF-style interface between HOL and first-order logic. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 134–138, Copenhagen, Denmark, July 2002. Springer.

[14] R. Kumar, T. Kropf, and K. Schneider. Integrating a first-order automatic prover in the HOL environment. In M. Archer, J. J. Joyce, K. N. Levitt, and P. J. Windley, editors, *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications*, pages 170–176, Davis, California, USA, August 1991. IEEE Computer Society Press, 1992.

[15] Jia Meng. Integration of interactive and automatic provers. In Manuel Carro and Jesus Correas, editors, *Second CologNet Workshop on Implementation Technology for Computational Logic Systems, FME 2003*, September 2003.

[16] Jia Meng and Lawrence C. Paulson. Experiments on supporting interactive proof using resolution. In David Basin and Michaël Rusinowitch, editors, *Automated Reasoning: Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 372–384. Springer, 2004.

[17] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, Las Vegas, June 2001.

[18] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[19] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Au-*

tomated Deduction (CADE), volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer.

[20] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[21] Lawrence C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3):73–83, 1999.

[22] Natarajan Shankar. Using decision procedures with a higher-order logic. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings*, volume 2152 of *Lecture Notes in Computer Science*, pages 5–26. Springer, 2001.

[23] Ofer Strichman. On solving Presburger and linear arithmetic with SAT. In M. D. Aagaard and J. W. O'Leary, editors, *Formal Methods in Computer-Aided Design: 4th International Conference, FMCAD 2002, Portland, OR, USA, November 6-8, 2002, Proceedings*, volume 2517 of *Lecture Notes in Computer Science*, pages 160–169. Springer, 2002.

[24] Geoff Sutcliffe and Christian Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998. Available online at http://www.cs.miami.edu/~tptp/.

[25] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation Of Reasoning: Classical Papers On Computational Logic, Vol. II, 1967-1970*, pages 466–483. Springer, 1983. Also in *Structures in Constructive Mathematics and Mathematical Logic Part II*, ed. A. O. Slisenko, 1968, pp. 115–125.

[26] Tjark Weber. Bounded model generation for Isabelle/HOL. In *2nd International Joint Conference on Automated Reasoning (IJCAR 2004) Workshop W1: Workshop on Disproving – Non-Theorems, Non-Validity, Non-Provability*, Cork, Ireland, July 2004.

[27] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Design, Automation and Test in Europe (DATE 2003)*, pages 10880–10885. IEEE Computer Society, 2003.