A comparison of four propositional theorem provers

(Revised version)

Lars-Henrik Eriksson
Industrilogik L4i AB
P.O. Box 21024
SE-100 31 STOCKHOLM
SWEDEN

lhe@L4i.se

Document L4i-00/111.
March, 17, 2000

1. Introduction

Propositional theorem provers – also known as satisfiability (SAT) solvers – are receiving increased attention in the area of formal verification since recent developments in algorithms has made it possible to use propositional theorem proving to a variety of large-scale problems, such as symbolic model checking, circuit equivalence and formal verification by refinement proof.

Greentech Computing Ltd[1]. was seeking industrial-scale problems not taken from the circuit verification domain to try out on their GSVT theorem prover. As Industrilogik works with problems of this kind, we were given the opportunity to evaluate GSVT. To get a general idea of the current state of the art in propositional theorem proving, we carried out the evaluation as a comparison of GSVT and three other theorem provers: HeerHugo [5], NP-Tools [4] and SATO [6].

2. The problems

2.1. Background

With the exception of two prime number problems supplied by Greentech Computing, all the problems were taken from Industrilogik's work on formal specification and verification of railway interlockings [2,3].

In this work, a generic specification in temporal predicate logic of safety requirements of interlockings was developed. The specification is generic in the sense that it does not express requirement for any given interlocking system, but general requirements applicable to all systems. Supplying information about the structure of a particular rail yard yields the specific requirements for an interlocking intended for that particular rail yard ("instantiating" the specification). When this information has been supplied, it is also possible to translate the predicate logic expressions into propositional logic amenable to analysis by a propositional theorem prover.

Using the instantiated specification and a model in propositional logic of an actual interlocking system for the site in question, it is possible to form a propositional theorem proving problem to show whether or not the interlocking system fulfils the requirements of the specification.

The particular rail yard used in the problems used in this comparison is that of the station in Brunna, outside Uppsala in Sweden.

Two distinct sets of problems were derived from the specification:

1) A set of specification validation problems, which shows that the instantiated specification itself fulfils certain correctness conditions.

2) A set of system verification problems, showing that the actual interlocking installation at Brunna station fulfils the requirements of the instantiated specification.

2.2. A note on the formulation of theorem proving problems

A theorem proving problem typically has the form $A \rightarrow P$, where A is a conjunction of axioms and P is the theorem to prove. Since all four theorem provers basically attempt to find satisfying assignments to a formula, the actual formula given to the provers is $\neg (A \rightarrow P)$. If a satisfying assignment to this formula can be found, P is not a logical consequence of A

[1] Greentech Computing Ltd., Garden Flat, 47 Frognal, London NW3 6YA, `http://www.greentech-computing.co.uk`.

and the resulting assignment shows why it is not. If this formula is inconsistent, then P is actually a logical consequence of A.

Is should thus be kept in mind that if P is actually a theorem, then the problem as seen by the theorem prover will be *inconsistent* and if P is not a theorem, then the problem will be *satisfiable*.

(The user interface of NP-Tools can present the user with the possibility to either find a satisfying assignment of a formula or to show directly that the formula is valid. To make the presentation consistent, we have chosen to ignore the latter possibility.)

2.3. Test set 1: Correctness conditions.

The problems *test1ok*, *test2ok* and *test3ok*, each state that the instantiated specification fulfils some correctness condition (the three properties in section 5.3 of [2]). For each of these three problems, there are also strengthened versions, *test1fel*, *test2fel*, *test3fel*, where the correctness conditions have (somewhat arbitrarily) been strengthened so that they no longer hold. The purpose is to test the ability of the theorem provers to generate a satisfying value assignment.

2.4. Test set 2: Verification

The problems with names beginning with *brunna* all relate to the formal verification of the particular interlocking system at Brunna station.

brunnabug is the actual formal verification problem. It exhibits an error in the Brunna interlocking, so this problem is satisfiable.

brunnatotal is the same formal verification problem, where the situation leading to the error has been defined away, so that the verification is successful, i.e. the problem formula is inconsistent.

The other *brunnaXXX* problems are subproblems of *brunnatotal*. For the analysis of the test results, it is of some importance to understand the structure of *brunnatotal* and in what sense the *brunnaXXX* are subproblems.

brunnatotal has the form $A \rightarrow P$. A is a conjunction of axioms, including definitions of about 40 verification conditions. P is a conjunction of the propositional variables defined as each of the verification conditions. In greater detail, the problem formula is:

$$\ldots \wedge (c_1 \leftrightarrow \ldots) \wedge \ldots \wedge (c_n \leftrightarrow \ldots) \wedge \ldots \rightarrow c_1 \wedge \ldots \wedge c_n$$

where the $c_1, \ldots, c_n$ are particular propositional variables[2].

The *brunnaXXX* subproblems are identical except that they include only a subset of the $c_1, \ldots, c_n$ to the right of the implication, i.e. the *definitions* of all verification conditions are still included in the subproblem.

There is also a problem named simply *brunna*. This is a pseudo-verification problem where the right-hand side of the implication is the logical constant FALSE, rather than a conjunction of propositional letters. This problem is satisfiable and has been included as a test of how fast a satisfying assignment can be found.

[2] The definiens of a c_i may itself be a conjunction.

2.5. Test set 3: Prime numbers

The final test set comprises to problems supplied by Greentech Computing, *prim1* and *prim2*. *prim1* states that the number 3476741 is prime, while *prim2* states that the number 58697731 is prime. Both problem formulae are inconsistent.

3. The theorem provers tested

3.1. GSVT

GSVT is a commercial theorem prover based on a novel proprietary algorithm developed by Greentech Computing. Very little has been disclosed about the properties of the algorithm or how the algorithm works. We have been told that it operates in two stages: A simplification stage which runs in about linear time in the size of the problem formula and a proper theorem proving stage. For simple problems, the simplification stage dominates.

The version of GSVT tested was 0.8.

3.2. NP-Tools

NP-Tools [5] is a powerful commercial modelling and verification tool developed by Prover Technology. It is based on a theorem prover employing the patented ”Stålmarck method”. This method uses a combination of an incomplete proof procedure of linear time complexity with a branch/merge rule. The latter rule splits the proof in two branches, one where some propositional variable is assumed to be true and one where it is assumed to be false. The two branches are later joined by discharging the assumptions and keeping the intersection of the conclusion sets of the two branches.

The minimum number of nested instances of the branch/merge rule required in any proof of a problem formula is called the *degree of hardness* of that formula. The Stålmarck procedure is exponential in the hardness of the formula, but polynomial in the size of the formula assuming a maximum degree of hardness. What makes the procedure interesting is that problems encountered in practice generally have low degrees of hardness (0 – 2).

When carrying out a proof using NP-Tools, the user sets a *saturation level* which is the largest number of nested branch/merge instances to be attempted. This means that NP-Tools will only find proofs of formulae with at most the corresponding degree of hardness (however, see 6.3).

If NP-Tools fails to prove a formula inconsistent given a particular saturation level, it assumes that a satisfying assignment can be found and sets out to find such an assignment by using a backtracking procedure to try different assignments to the variables until a satisfying assignment has been found. In practice, every possible assignment need not be tried as there will be dependencies between variables which can rule out whole classes of assignments.

The version of NP-Tools tested was 2.4.

3.3. HeerHugo

HeerHugo [4] is an academic theorem prover developed by Jan Friso Groote of the CWI, the Netherlands. It was inspired by the Stålmarck method but differs substantially from it[3].

[3] Although inventors sometimes want to believe that a patent would prevent this, it is precisely to make this kind of development possible that society grants an inventor exclusive rights in return for a full public description of the invention.

HeerHugo shares the branch/merge rule with the Stålmarck method but uses different data structures and a different, more powerful, linear procedure.

When the problem formula is satisfiable, HeerHugo sometimes does not find values for all variables. By adding the values found to the problem formula and running HeerHugo again, a complete assignment is obtained.

In a test carried out in 1997, the then current version of HeerHugo was slightly faster than the then current version of NP-Tools. However, no development of HeerHugo has been done for some years, so is has fallen behind in the competition. The version of HeerHugo tested was 0.3.

3.4. SATO

SATO [6] is an academic theorem prover developed by Hantao Zhang of the University of Iowa. It is based on the Davis-Putnam rule [1]. The version of SATO tested was 2.3.

SATO requires its input to be in clausal form, while the problems to be tried were all expressed in full propositional logic. The problems were converted to 3-CNF before being given to SATO.

4. Test environments and timing procedures.

The tests of GSVT, HeerHugo and SATO were carried out on IBM PC-type computer with a 450 MHz Pentium III processor and 384 Mbyte primary memory. GSVT was run under Windows, while HeerHugo and SATO were run under Linux. HeerHugo and SATO were compiled using gcc version 2.91 with the `-O3` option.

While NP-Tools does run under Windows, we did not have access to a Windows version of that program. Instead, it was run on a SUN Ultra 5 workstation. In the tables below, the NP-Tools run times have been scaled to be directly comparable to those of the other three theorem provers. By examining various benchmarks, we concluded that the Ultra 5 ran at a speed of between 50% and 100% of the speed of the Pentium system. The figure 70% was used for scaling. This does give a major uncertainty of the timing results, but as can be seen from the timing figures, this uncertainty in no way affects the conclusions drawn from the comparison.

The run time figures given by the theorem provers themselves were used, rounded to the nearest second. This time includes not only theorem proving time, but also the time taken to parse the input data etc. The time taken to convert the problem formula into clausal form for SATO (see 3.4) or running HeerHugo a second time (see 3.3) is also included.

For GSVT, HeerHugo and SATO, each test was run four times and the average taken. The inherent uncertainty of the NP-Tools timing did not warrant more than one run of NP-Tools for each problem.

For all four provers, all settings relating to proof search were left at their default values, except on NP-Tools which was set to try the value TRUE first on backtracking. A HeerHugo option relating to logging the progress of the proof was disabled.

NP-Tools was initially run with the saturation level set to 1. If it did not find a proof, the problem was retried with the saturation level set to 2. In this case the times refer only to the second run. HeerHugo manages saturation levels automatically.

5. Test results

The following tables shows the test results. All run times are given in seconds rounded to whole numbers. For NP-Tools and HeerHugo the saturation level needed to solve the problem is also given. ”1+BT” means that the problem was solved using saturation level 1 followed by backtracking (see 3.2 and 6.3).

Unsatisfiable problems

Problem	GSVT	NP-Tools		HeerHugo		SATO	Conn.	Vars.
		Time	Level	Time	Level			
test1ok	1	0	1	1	1	1	14077	1286
test2ok	2	0	1	1	1	1	14127	1286
test3ok	2	2	1	1	1	6	18021	1286
brunnatotal	12	218	2	1288	2	(>23h)	22549	2422
brunna6	5	1	1	5	1	1	22510	2422
brunna21	8	1	1	7	1	1	22511	2422
brunna26	8	1	1	7	1	1	22511	2422
brunna27	7	1	1	6	1	1	22511	2422
brunnasimple	6	1	1	8	1	(>23h)	22533	2422
brunna1-3	(<6)	(<1)		(<8)		3	22512	2422
brunna4-5	(<6)	(<1)		(<8)		85	22511	2422
brunna4	(<6)	(<1)		(<8)		11	22510	2422
brunna5	(<6)	(<1)		(<8)		12	22510	2422
brunna7-12	(<6)	(<1)		(<8)		11	22515	2422
brunna13-18	(<6)	(<1)		(<8)		10	22515	2422
brunna19-23	(<6)	(<1)		(<8)		11	22513	2422
brunna242528	(<6)	(<1)		(<8)		3	22512	2422
brunnaind	(<12)	0	1	1	1	1	22510	2422
brunnas1	(<12)	21	2	19	2	2	22511	2422
brunnas1a	(<12)	1	1	8	1	(<2)	22511	2422
brunnas1b	(<12)	1	1	8	1	(<2)	22511	2422
brunnas2	(<12)	0	1	1	1	1	22510	2422
brunnas3	(<12)	117	2	152	2	2	22510	2422
brunnas3a	(<12)	1	1	15	1	(<2)	22510	2422
brunnas3b	(<12)	1	1	8	1	(<2)	22510	2422
brunnas3c	(<12)	1	1	14	1	(<2)	22510	2422
brunnas3d	(<12)	1	1	15	1	(<2)	22510	2422
brunnas3e	(<12)	1	1	15	1	(<2)	22510	2422
brunnas4	(<12)	173	2	362	2	2	22510	2422
brunnas4a	(<12)	1	1	28	2	(<2)	22510	2422
brunnas4b	(<12)	1	1	10	1	(<2)	22510	2422
brunnas4c	(<12)	1	1	10	1	(<2)	22510	2422
brunnas4d	(<12)	1	1	24	2	(<2)	22510	2422
brunnas5	(<12)	1	1	7	1	1	22510	2422
brunnas6	(<12)	1	1	2	1	1	22510	2422
brunnas7	(<12)	0	1	0	0	1	22510	2422
prim1	17	1	1+BT	3869	2	31	4139	551
prim2	85	4	1+BT	≈ 3 d.	3	211	5772	755

Subproblems are shown indented. E.g. *brunnasimple* is a subproblem of *brunnatotal* and *brunna1-2* is a subproblem of *brunnasimple*. The subproblems given are disjoint and exhaustive, e.g. *brunna4* and *brunna5* are disjoint and together form *brunna4-5*.

”(>23h)” in the case of SATO means that SATO aborted the proof search after 23 hours without having solved the problem.

”≈3 d” means that approximately three days of run time was required.

”(<N)” means that this problem was not attempted, but that it was a subproblem of problem solved in N seconds. Thus the theorem prover in question could reasonably be expected to solve the given problem in at most N seconds.

The columns ”Conn.” and ”Vars.” give some metrics for the size of the problems. ”Conn.” gives the number of logical connectives in the problem formula, while ”Vars.” is the number of distinct propositional variables in the formula.

Satisfiable problems

Problem	GSVT	NP-Tools		HeerHugo		SATO	Conn.	Vars.
		Time	Level	Time	Level			
brunna	7	3	1+BT	10	1	1	22508	2422
brunnabug	8	3	1+BT	62	2	2	22549	2422
test1fel	2	1	1+BT	1	1	1	14079	1286
test2fel	2	1	1+BT	2	1	1	14125	1286
test3fel	2	1	1+BT	2	1	1	16741	1286

6. Analysis

6.1. General

It can be seen that in general the four theorem provers are in good agreement on what problems are more or less difficult. Of course, the relative difference between two problems differs greatly for different theorem provers.

There are, however, some interesting anomalies which will be discussed in the sequel.

6.2. The conjunction effect

Looking at the relative times for the various problems of the *brunna* family, it can be seen that the time it takes NP-Tools to solve *brunnatotal* is considerably higher than the sum of the times it takes to solve its subproblems. In this case the subproblems could be solved in a total of 18 seconds compared to 218 seconds for the composite problem (a factor 12 slowdown). This is an example of a known property of the Stålmarck method: the hardness of the problem formula $p \wedge q$ can be higher than the hardness of each of the problem formulae p and q, respectively.

HeerHugo, using the same ideas as NP-Tools, exhibits the same behaviour,although the difference is less dramatic: 199 seconds for the subproblems compared to 1288 seconds for the composite problem (a factor 6.5 slowdown).

What is perhaps more surprising is that SATO, which uses completely different principles also exhibits this behaviour, and to an even greater degree. We speculate that the reason for this is the preferential handling of unit clauses employed by SATO and other theorem provers working with data in clausal form. When transforming a problem formula $\neg (A \rightarrow c)$ to clauses, the clause $\neg c$ will be generated. Being a unit clause, it will be given preferential treatment. If the problem formula is instead $\neg (A \rightarrow c_1 \wedge \ldots \wedge c_n)$ the corresponding clause will be $\neg c_1 \vee \ldots \vee \neg c_n$, which is not a unit clause. This could result in poor directionality of the proof search.

Of course, a very simple way of eliminating this behaviour would be to break up a problem into its conjunctive subproblems and solving the subproblems separately. In many cases this would give a considerable decrease in total run time.

The question is, of course, how far to carry the breaking up of the problem, as unnecessarily breaking up a problem will incur an extra overhead.

According to Greentech Computing, about half the runtime of GSVT for *brunnatotal* was spent in the simplification stage (see 3.1). If we assume that the GSVT runtime for *brunna6* is all spent in simplification and subtract this from all GSVT runtimes we obtain approximate runtime figures for the proper theorem proving stage. From this it appears that GSVT does not exhibit the conjunction effect, or at least to a much smaller degree than the other theorem provers.

6.3. Solution by backtracking

As remarked above, the four theorem provers are in general agreement on the relative difficulty of the various problems.

A notable exception to this is the two prime number problems, which are surprisingly easy for NP-Tools. It turns out that these problems actually have a higher degree of hardness than 1, and if solved in the normal way would take a comparably long time for NP-Tools to solve. Instead, NP-Tools solves the problems by backtracking (see 3.2).

If no satisfying assignment is found during backtracking, the formula must be inconsistent. This is in general an intractable way of proving a formula inconsistent due to the exponentially large number of different assignment.

The reason NP-Tools succeeds in these two cases to solve an inconsistent problem rapidly with backtracking is likely the comparatively small number of propositional variables in the problems combined with a suitable internal structure which creates many dependencies between variables, enabling NP-Tools to eliminate large sets of possible assignments.

6.4. Finding satisfying assignments

Generally, all four theorem provers find satisfying assignments quickly. A notable exception is the *brunnabug* problem, which is solved rapidly by NP-Tools, GSVT and SATO, but takes an order of magnitude longer with HeerHugo. We have no explanation for this behaviour.

7. Conclusions

For the problems tried, the major factor affecting performance seems to be the ”conjunction effect”. SATO appears to be by far the fastest of the theorem provers as long as the conjunction effect is not manifest. When it is, however, SATO become the slowest of the theorem provers tried. The point where the conjunction effect becomes manifest also differs

between the various theorem provers. SATO is affected already when a few subproblems are composed, while HeerHugo and NP-Tools can accept a larger number of composite problems.

For the *brunnatotal* problem, GSVT is the fastest of the theorem provers tested. It appears to not have the conjunction effect, which would make it outstanding in this comparison. On the other hand, the evidence is rather vague and tests with more difficult problems would be required to make any definite statements.

According to Greentech Computing, GSVT displays its strength best when applied to inherently difficult (as opposed to simply large) problems. It might well be that difficulty of the problems from the railway interlocking domain which were tried stem from their size and not from any inherent difficulty.

The conclusion in that case must be that although performance differ, all four theorem provers are capable of handling real problems from the domain in question. The inconvenient decomposition of the problem into subproblems which is necessary in the case of SATO and useful in the cases of NP-Tools and HeerHugo could be done automatically.

8. References

[1] Davis, M., Logemann, G. and Loveland, D.: *A machine program for theorem-proving*, Comm. ACM, vol 5, no 7 (July 1962), pp 394-397.

[2] Eriksson, L.-H.: *Formalising Railway Interlocking Requirements*, Technical report 1997:3, The Swedish National Rail Administration 1997.

[3] Eriksson, L.-H.: *Formal Verification of Railway Interlockings*, Technical report 1997:4, The Swedish National Rail Administration 1997.

[4] Groote, J.F. and Warners, J.P.: *The propositional formula checker HeerHugo*, Technical Report SEN-R9905, Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, 1999.

[5] Stålmarck, G., and Säflund, M.: *Modelling and Verifying Systems and Software in Propositional Logic*, In: Proceedings of SAFECOMP '90, pp.31-36, Pergamon Press, 1990.

[6] Zhang, H.: *SATO: An Efficient Propositional Prover*, In McCune (ed.): *Proc. 14th International Conference on Automated Deduction* (CADE-14), Springer Lecture Notes in Computer Science, 1997.

