Guiding a Theorem Prover with Soft Constraints

John Slaney and **Arnold Binas** and **David Price** [1]

Abstract. Attempts to use finite models to guide the search for proofs by resolution and the like in first order logic all suffer from the need to trade off the expense of generating and maintaining models against the improvement in quality of guidance as investment in the semantic aspect of the reasoning is increased. Previous attempts to resolve this tradeoff have resulted either in poor selection of models, or in fragility as the search becomes over-sensitive to the order of clauses, or in extreme slowness. Here we present a fresh approach, whereby most of the clauses for which a model is sought are treated as *soft* constraints. The result is a partial model of all the clauses rather than an exact model of only a subset of them. This allows our system to combine the speed of maintaining just a single model with the robustness previously requiring multiple models. We present experimental evidence of benefits over a range of first order problem domains.

1 THE PROBLEM: INTELLIGENT PROOF SEARCH

First order theorem proving is the traditional core of automated reasoning, of importance not only for pure mathematics but also for its applications to AI—to planning, for instance—and to software engineering among other fields. The search spaces encountered in theorem proving are typically infinite, and for reasons related to undecidability there is little regularity to their structure. Proof search therefore relies on rules of thumb backed by little but empirical wisdom. It is therefore somewhat surprising, and certainly disappointing, that attempts to search intelligently for proofs continue to be less successful in practice than brute force.

Over the last few years, there have been several attempts to inject intelligence into the search by combining a theorem prover with some module that turns sets of first order clauses into constraint satisfaction problems and then uses a finite domain CSP solver to generate models relevant to the theorem being sought. These models somehow represent information about the meaning of the problem—that is, they give the prover a rudimentary understanding of the problem—and so may be used to guide the proof search. The general technique is to concentrate the search on clauses which are false in the guiding model or models. This is intuitively reasonable: the aim is to show that the input clause set is *necessarily* false, so it makes sense to seek anomalies among the consequences of those clauses which are *actually* false.

Slaney, Lusk and and McCune [5] proposed the system SCOTT which uses a model to restrict the inference rules such as resolution by requiring in each inference that one of the parent clauses be false in the guiding model. According to [4] that system was quite fragile, in that small changes in the order in which clauses are processed have large effects on its behaviour, and it was shown to be incomplete for some inference rules only slightly more interesting than binary resolution.

Hodgson and Slaney [4] later produced a version of SCOTT in which robustness was secured by maintaining several models rather than just one, and completeness was achieved by using the models to guide clause selection without restricting the inference rules. That system, however, is extremely slow because of the overheads incurred in generating and maintaining many models. It did compete several times in CASC, as reported in [4], achieving performance marginally better than that of OTTER.

Choi and Kerber [2, 3] propose a different technique whereby models related to the input clauses are generated in a preprocessing phase and are then again used to guide clause selection. Their system uses the clause graph method, and suffers from the fact that only very small models (with domain size 2) can be generated and used in that way. More problematically, only the input clauses are considered when the models are being generated, so that properties that emerge only after some consequences have been deduced are likely to be ignored.

Brown and Sutcliffe [1, 7] have developed a system PTTP+GLiDeS in which models are generated with a propositional SAT solver and used to constrain the inferences made in the course of proofs by linear input resolution. The system shows interesting efficiency gains over PTTP on some non-Horn problems, but cannot improve on it in the Horn case and has yet to demonstrate general usefulness in comparison with conventional provers.

In the present paper, we present a new approach within this overall research program. The paper is organised as follows. In section 2 we outline the method and what makes it distinctive. Then in section 3 we examine a small example of a proof search in order to illustrate the idea. Section 4 contains experimental results on the "hard" problems from the TPTP library, and we conclude with an indication of planned further work.

2 A FRESH APPROACH

We follow [4, 5] in basing our system on the pre-existing theorem prover OTTER [8]. This is no longer the fastest theorem prover in its class, but is still a high performance system which is well known, widely used, well maintained and stable. Like Choi and Kerber [2] we also follow [4, 5] in using FINDER [11] as the constraint solver.[2] While not comparable with the state of the art in CSP, FINDER is suitable for generating small models quickly, accepts first order

[1] Australian National University and National ICT Australia Ltd, Canberra. Email: John.Slaney@nicta.com.au.
National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

[2] It might have been neater to use MACE, following the example of [7], but MACE does not appear to exist in a version that supports soft constraints by solving MAX-SAT problems.

clauses as input and comes with functions designed for linking it to external software in just this way. For our system, we used a version of FINDER which allows constraints to be soft, treating the problem as essentially a weighted MAX-CSP. The details are not so important to the theorem-proving application: for our purposes, it is enough that it accepts as input a set of first order clauses each marked either as hard or as soft and returns a model of the hard clauses satisfying as many as possible of the *instances* of the soft ones interpreted over a given finite domain. We use this model to guide OTTER's search.

2.1 The given clause algorithm

The core of OTTER, as of most other high performance first order theorem provers,[3] is the given clause algorithm. For this, the clauses are partitioned into an active set (the *usable list* in OTTER parlance) and a passive set or *set of support*. The main part of the algorithm is a loop executed indefinitely until either the set of support is empty or the goal (usually the empty clause) is deduced:

Procedure **GivenClauseLoop**
  While the set of support is not empty do
    Select given clause g from the set of support
    Move g to the usable list
    For each immediate consequence c of the usable list that has g as a parent do
      If c is the goal then
        Return `success`
      If c passes the filters then
        Add c to the set of support
  Return `failure`
End **GivenClauseLoop**

Not specified here are the steps of back subsumption and back demodulation, whereby the existing clauses are rewritten using new clauses. This may be done eagerly before the new clause c is kept in the set of support, or lazily when it is selected as the given clause. OTTER is eager; Waldmeister, for example, is lazy. Also unspecified are the criteria for given clause selection, the choice of rules defining "immediate consequence" and the filters used to remove unwanted clauses. Details of the rules and filters will not be given here, except to note that some filters such as subsumption preserve the completeness of the method when rules such as resolution are used, while others such as deleting all clauses above a certain length limit lead to incompleteness.

2.2 Semantic guidance

OTTER's default criterion for selection of the next given clause is to choose one of those with the smallest number of constituent symbols (function symbols and variables), breaking ties by choosing the oldest—i.e. the one which was placed in the set of support first. This criterion is tempered by the "pick-given ratio" which stipulates that every k-th clause is chosen to be simply the oldest, without regard to its length, so that in effect a breadth-first search is interleaved with the best-first one.

Clearly, the criteria for given clause selection make no reference to the meaning of the clause or its likely rôle in any proof. In the hope of improving the quality of selection, and therefore the efficiency of the search, we augment the criterion with a semantic component. For this purpose, let M be a model—that is, an interpretation of the language of the problem in which each clause in the language is either true (for all assignments of values to its variables) or false (for at least one assignment). Our suggestion is to choose most of the given clauses from among those false in the guiding model. That is, by default we choose the oldest of the shortest of the false clauses. Because some true clauses may also be required for a successful proof, choosing *only* false clauses would lead to incompleteness,[4] so every so often we follow OTTER's clause selection without regard to the semantics. The ratio of semantic to non-semantic clause selections is controlled by another magic number, the "semantic-given ratio" which is input as a parameter like the pick-given ratio.

Semantically guided choice requires a decision as to whether a clause is "true" or "false", which is given by testing against a (small) finite model. Sophisticated model checking is not required, as the model is very small (we rarely use a domain of more than three or four elements) and the first order clauses occurring in feasible proof searches are typically rather short, so crude enumeration of valuations of the constituent variables suffices.

The more interesting question is how to determine the guiding model. For this, first note some vocabulary. Let c be a first order clause containing variables $x_1, \ldots, x_n$ and let M be a model with a finite domain which may as well consist of the integers $\{1, \ldots, k\}$. In addition to the function symbols, constants and the like in the first order language in question, let there be special constants $\underline{1}, \ldots, \underline{k}$ whose interpretation is fixed in the obvious way. Now among the ground instances of c, of which there will normally be infinitely many, there are those in which only the constants $\underline{1}, \ldots, \underline{k}$ are substituted for the variables. We call these *domain-grounded* instances and obviously there are only finitely many of them (k^n in fact). Trivially, c is equivalent in the model M to the conjunction of its domain-grounded instances.

Example: let c be the clause $\neg P(f(g(x))) \vee P(f(x))$ and let the domain of M be $\{1, 2, 3\}$. Then while c has infinitely many ground instances $\neg P(f(g(g(g(a))))) \vee P(f(g(g(a))))$ etc, it has only three domain-grounded instances relative to the domain of M, *viz.* $\neg P(f(g(1))) \vee P(f(1))$, $\neg P(f(g(2))) \vee P(f(2))$ and $\neg P(f(g(3))) \vee P(f(3))$.

The CSP corresponding to a set of clauses is obtained by first *flattening* the clauses by introducing extra variables and setting them equal to the subformulae. We say that a term is *flat* if it contains no nested function symbols—i.e. if the only terms inside a function symbol are variables—and that an atomic formula is flat if either it contains no function symbols at all or else it is an equation between a variable and a flat term. A literal is flat if the atom in it is flat, and a clause if every literal in it is flat. Then every clause has a flat equivalent. Now for the CSP, the domain variables correspond to the domain-grounded instances of flat terms and flat atoms. Each domain-grounded instance of the flattened clause then states a constraint or a relation (usually non-binary) between domain variables of the CSP.

Example: Let c and M be as above. Then the result of flattening c is $v_1 \neq g(x) \vee v_2 \neq f(v_1) \vee v_3 \neq f(x) \vee \neg P(v_2) \vee P(v_3)$. Any domain-grounded instance of this flattened clause, such as $\underline{3} \neq g(\underline{1}) \vee \underline{2} \neq f(\underline{3}) \vee \underline{3} \neq f(\underline{1}) \vee \neg P(\underline{2}) \vee P(\underline{3})$, constrains the possible values for a 5-tuple of the CSP domain variables.

[3] For example, Vampire [9], Gandalf [14], SPASS [15], Waldmeister [6] and E [10] all use variants of the given clause algorithm. See the CASC results [12] for a rough but revealing comparison of the best systems.

[4] —unless we were to use full semantic resolution with the dynamic model, of course. We allow the prover to use ordinary rules such as binary resolution, hyperresolution, paramodulation, etc. With these, true given clauses are sometimes needed for completeness.

At any stage in the proof search, let the clauses in the usable list be partitioned into "hard" and "soft". By an *approximate model* of the usable list we mean any model of the hard clauses. By the *badness* of an approximate model M we mean the number of domain-grounded instances (relative to M) of the flattenings of soft clauses which are false in M.[5] M is an optimal model over a given domain if its badness is minimal among models over that domain. That is, we model the usable list as a MAX-CSP with mixed hard and soft constraints—we can view it as a weighted MAX-CSP with the hard constraints having infinite weight—where each constraint is given by a domain-grounded instance of one of the flattened clauses. Note that each clause in the language is either (absolutely) true in M or (absolutely) false in M despite the appeal to matters of degree in the generation and evaluation of M.

The hard clauses are those initially in the usable list; the soft ones are those which have been in the set of support. There are two reasons for requiring some of the constraints to be hard. Firstly, it is much more efficient to search for models of soft constraints within a tightly constrained search space than in a totally unconstrained one. Secondly, the clauses initially in the usable list are special in that they cannot interact with each other to produce consequences since they never get chosen as the given clause. They define the backgrouond theory of the proof search, and in many cases also define in effect the goal. Hence by requiring them to be true in the guiding model, we constrain that model to cohere with the implicit semantics of the search. At any rate, it seems empirically to be the case that making them into hard constraints tends to avoid useless models such as those which make the entire set of support true and so give no guidance.

We model only the usable list, not the set of support. This greatly reduces the number of clauses which have to be modelled. Each given clause, if it is true in the current model, is simply added to the usable list. If it is false, a new model is sought which should be better than the current one taking the new clause into account along with the rest of the usable ones.[6] This is the core difference between our system and previous semantically guided provers. Our single approximate model is chosen to capture as much as possible of all the usable clauses, whereas their exact models each capture just an aspect of the problem, since clauses which are false in an exact model contribute nothing to it. Another difference is that we allow the model to change in whatever way helps to make more domain-grounded clause instances true. In [2] the models never change once the search starts. In [4] and [5] the models do change, but subject to the condition that clauses labelled as true by one model must continue to be labelled as true by later ones. We see no need for such a condition, and indeed consider it harmful since it "locks in" a bad choice made early in the search whereas our system has the option of undoing any choice of model once its badness becomes apparent. We do, however, have to re-test the clauses in the set of support after each model update. At some point (after a time limit or after a number of given clauses specified as a parameter with a somewhat arbitrary default value of 250) the generation of models is disabled because it is expensive and the returns diminish rapidly after a while.

For the experiments reported below, we fixed the domain size to be 3, not for any interesting reason but just because models of that size fairly often give decent guidance and are small enough to be found quickly. We also fixed a limit of 10^5 variable instantiations for each model search, so that the system should not spend too much of its time trying to improve the model. Thus the model returned at each point is the best found within a cutoff and not necessarily the optimal model on the domain.

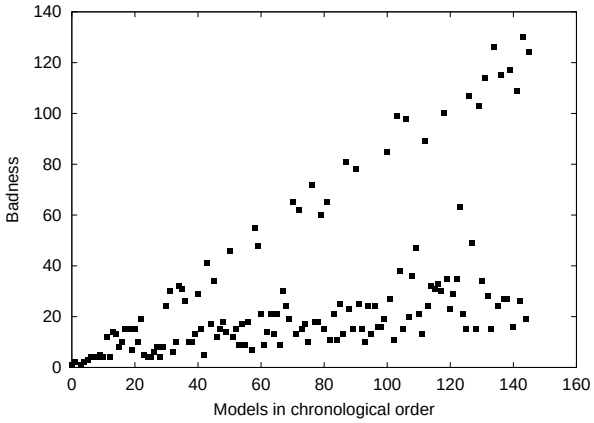

Figure 1. The badness of soft models in a search for a proof of problem FLD049-4. "Badness" is the number of falsified constraints.

3 AN EXAMPLE

Before reporting the experiments, it is useful to illustrate with an example. The problem chosen for this is FLD049-4 from TPTP: there is no deep reason for this choice except that it is one of the problems in the "eligible problems" list for CASC in 2003 which both our system SOFTIE and the underlying prover OTTER can solve in a reasonably short time. The theorem is that in any field, for any elements a and c and for any nonzero elements b and d, if $ab^{-1} = cd^{-1}$ then $ad = bc$. This fairly basic fact of field theory is made awkward to prove in FLD049-4 by being expressed in terms of ternary relations `sum` and `product` as well as functions `add` and `multiply`, the relation of equality being axiomatised rather than written explicitly as '=' to prevent provers from using equational reasoning directly. SOFTIE found a proof in 244 seconds after 184 given clauses of which 25 (15 input clauses and 10 derived ones) are in the proof. By way of comparison, OTTER takes only 1.58 seconds to find a different proof, also of length 25, after 250 given clauses.

The reason why SOFTIE is so slow is that it re-generates the guiding model after almost every selection of given clause, resulting in 145 changes of model for 184 given clauses. Of those 184 clauses, only 27 are true in the model at the time when they are added to the usable list. The models vary in how much of the set of support they verify, but during most of the search about 90% of the clauses in the set of support are marked as true, so the preference for false given clauses clearly focuses the search considerably. The cost, however, is that the program spends almost all of its time searching for models as opposed to making inferences.

The penultimate model (one of the best used) has these tables for addition and multiplication:

$+$	0	1	2
0	0	1	0
1	1	1	1
2	0	0	0

$\times$	0	1	2
0	0	1	0
1	2	1	0
2	0	0	0

The additive and multiplicative identities are 0 and 2 respectively.

[5] The number of such false instances may be greater than the number of constraints resulting from them, because different domain-grounded instances may yield the same constraint. Each constraint is therefore weighted by the number of clause instances which are false if it fails.

[6] This is not completely accurate: in the present implementation, long clauses (generating constraints of cardinality greater than 4) are not modelled, because FINDER has no good way of treating large constraints as soft. This *ad hoc* restriction will be removed in a more mature implementation.

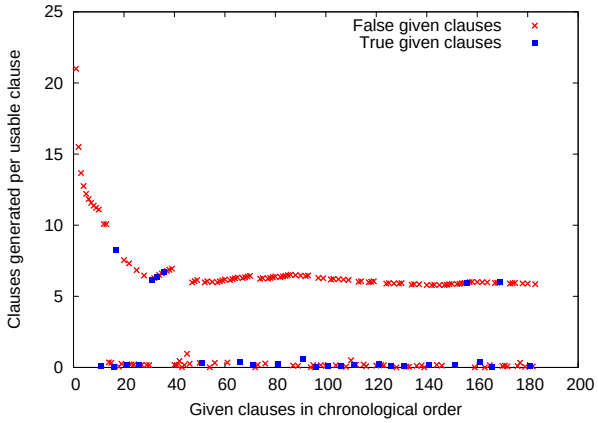

Figure 2. Number of clauses generated by each successive given clause during the search for a proof of TPTP problem FLD049-4.

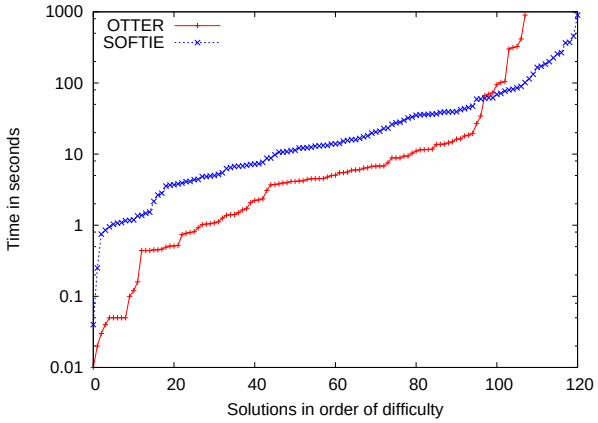

Figure 3. Times taken to find proofs, with and without semantic guidance, for Horn problems (the HNE and HEQ sections) of the CASC 'eligible' problems. The solved problems and the order of in which they come is different for the two lines, so that they are both monotone increasing. Cutoff is 900 seconds. A 'timeout' data point at 900 seconds has been added in each case.

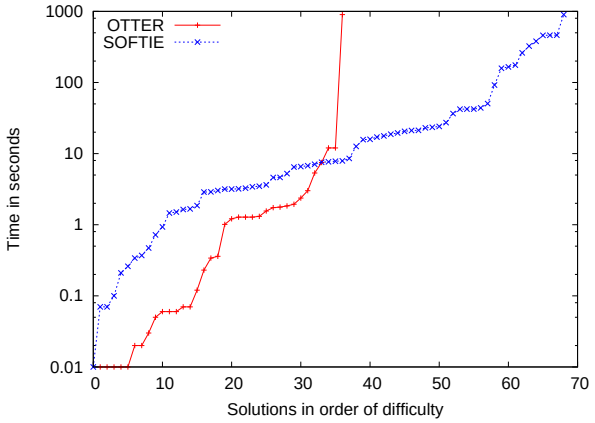

Figure 4. UEQ: Time taken to find a proof. These are the 'unit equality' problems in the CASC 'eligible problems' list.

While this structure is certainly not an accurate reflection of addition and multiplication as taught in the Schools, it does verify all but 19 domain-grounded instances of the usable clauses, as well as over 90% of the set of support. Hence it describes the problem pretty well.

Figure 1 shows the numbers of violated soft constraints in each of the 145 successive models generated. It seems that the choice of model fluctuates irregularly between two or more different cases, which presumably represent variations on different approximate models, one of which is much worse than the other(s). We have not ascertained how many isomorphism classes there are among the 145 models.

Figure 2 shows the number of clauses generated at each step—that is, as a result of adding each given clause. Interestingly, there are two sorts of given clause: those which have many immediate consequences (about 5 per usable clause) and those which have few. The clauses marked as true in the model at the time when they are selected are almost all in the latter category. Why this should be we do not know, but it hints at a genuine link between evaluation in models and deductive properties.

4 EXPERIMENTS

We have attempted to evaluate SOFTIE experimentally by running it on problems from the TPTP library [13]. This is not as simple as it may seem, because SOFTIE is very new and has no settled default values as yet for parameters like the semantic-given ratio. The termination condition for model searches is no better than a guess, as is the limit of 250 given clauses before model generation stops. The interaction of the semantic component with the settings of OTTER, from the choice of inference rules to the way of constructing the initial usable list, is also uninvestigated. Therefore the results of running SOFTIE with some parameters or other on a large problem set have to be seen as very rough.[7]

To bring the task within bounds, we concentrated on the 'hard' problems (i.e. neither trivial nor impossible) which are the 'eligible' problems for CASC [12]. Running the prover repeatedly over the whole set of eligible problems takes too long to be feasible, so Figure 3 shows timings for just two syntactic classes: those problems consisting of Horn clauses with and without equality. These are the two sections in which OTTER performs best.

[7] [4] reports similar frustration with the attempt to create an autonomous mode for SCOTT-5.

Here we give results for runs of the prover with the following settings:

- A time limit of 15 minutes (cpu time) per problem.
- A weight limit of 50 symbols per clause.
- 'Ratio' settings such that of every 5 given clauses, 3 are chosen to be the oldest of the shortest of the *false* clauses, 1 is the oldest of the *shortest* clauses without regard to semantics, and 1 is the *oldest* without regard to semantics or weight.
- A maximum of 450 seconds (50% of the time limit) to be spent in generating models.
- Otherwise the default "auto" settings that come with OTTER.

We have no confidence that these settings are in general ideal—in fact, we are sure they can be improved—but they give reasonable results in a good proportion of cases and are fairly simple.

As is evident from Figure 3 there is a time cost associated with model generation and testing. However, note that overall the number of proofs obtained within one minute is roughly similar with or without he investment in semantics, but that SOFTIE begins to dominate OTTER from about that point on. This suggests that semantic guidance in the style of SOFTIE pays its way more often as the problems become harder.

Performance is by no means even across problem categories. TPTP problems may be classified syntactically according to whether they contain equality, whether all the clauses are Horn and so forth, or semantically according to whether they are problems of geometry, group theory, planning, etc. Briefly, we find that soft semantic guidance as we have implemented it does better with equational reasoning than with pure first order logic. This was a surprise, as we developed it with rules such as resolution in mind, rather than term rewriting. Semantically, it suits some algebraic domains, notably group theory, better than it does other domainis. This may be because the model generator can efficiently find structures similar to groups, and therefore give high quality guidance, whereas it is too slow in modelling theories with large numbers of different predicates and function symbols, functions with many argument places and so forth.

In one syntactically defined problem class, the unit equality problems, SOFTIE clearly dominates OTTER. These problems require equational reasoning and many have an algebraic flavour. Just why semantic guidance should be more effective for these problems than for others is unclear, but the results shown in Figure 4 are striking. SOFTIE solves 68 of these 138 problems within the time limit, against OTTER's 36, and it is clear that if the time limit were increased the difference between the two provers would widen. As it is, the extra time spent finding models has an adverse effect on overall time only for problems which are solved in under ten seconds anyway.

5 CONCLUSION

We have presented a model-guided theorem prover using a MAX-CSP solver to generate models in which all of the usable clauses have as many true instances as possible. The guiding model is revised whenever a given clause is chosen which is false in the current model. This makes the theory determined by the model non-monotonic, as changing the model to minimise the violation of instances of the usable clauses may cause some currently true clauses to become false. We are not aware of any previous system for guiding a theorem prover which has this feature.

While the system is still very new, and much work remains to be done to remove the causes of abnormal termination of searches, and then to fine-tune the many settings and details of the algorithm for clause selection, we believe that SOFTIE already shows more than preliminary promise. It is complete, unlike the system described in [5], appears to be more generally applicable and more powerful than those in either [1] or [3], and faster than that in [4]. One interesting line of future work is to learn the features of problems which are correlated with the best ways of applying the model, thus allowing SOFTIE to adapt itself to different problem classes without requiring the intervention of a user.

More serious limitations on the current system arise from those of the components OTTER and FINDER. OTTER is extremely slow on some classes of problems in TPTP: on many problems in the PEQ section, for instance, it gets stuck for hours in the processing of one given clause, and on most problems containing clauses with many literals it is slow compared with more modern provers.Future work therefore includes trying semantic guidance of faster provers. The most annoying limitation of FINDER, for present purposes, is that it cannot treat constraints of cardinality greater than about 4 as soft because "grounding out" longer clauses is too inefficient. More future work includes overcoming this difficulty by incorporating a more sophisticated algorithm for handling soft constraints.

Meanwhile, our investigations have already uncovered some new questions about automatic first order proof search. What determines the two types of given clause shown clearly in Figure 2, for example? Is this a general phenomenon, or one local to specific problem types? Is it really related to semantics, or is the correletion with true and false clauses an accident of the particular case? More to the point of the present paper, we could wish for a theoretically compelling reason why semantic guidance works at all, and also, while positive effects are detectable, why they are not more dramatic. We have no answers to offer at present: in keeping with the field of theorem proving, our work has been strongly empirical in character. However, the questions have at least been opened.

ACKNOWLEDGEMENTS

We wish to thank the previous participants in the SCOTT project: Bill McCune, Ewing Lusk, Tim Surendonk and especially Kahlil Hodgson to whom we are indebted in many ways that may not all be obvious. Finally, we are grateful to National ICT Australia for its generous support of this project during 2003.

REFERENCES

[1] M. Brown and G. Sutcliffe, 'PTTP+GLiDeS – semantically guided PTTP', in *Proceedings of the 17th Conference on Automated Deduction (CADE)*, pp. 411–416, (2000).
[2] S. Choi, 'Towards semantic goal-directed forward reasoning in resolution', in *Proceedings of the 10th International Conference on Artificial Intelligence: Methods, Systems and Applications (AIMSA)*, pp. 243–252, (2002).
[3] S. Choi and M. Kerber, 'Semantic selection for resolution in clause graphs', in *Proceedings of the Australian Joint Conference on AI*, pp. 83–94, (2002).
[4] K. Hodgson and J. Slaney, 'TPTP, CASC and the development of a semantically guided theorm prover', *AI Communications*, **15**, 135–146, (2002).
[5] E. Lusk J. Slaney and W. McCune, 'SCOTT: Semantically constrained otter', in *Proceedings of the 12th Conference on Automated Deduction (CADE)*, pp. 764–768, (1994).
[6] B. Löchner and T. Hillenbrand, 'A phytography of WALDMEISTER', *AI Communications*, **15**, 127–133, (2002).
[7] M.Brown and G. Sutcliffe, 'PTTP+GLiDeS – guiding linear deductions with semantics', in *Proceedings of the Australian Joint Conference on AI*, pp. 244–254, (1999).
[8] W. McCune. Otter 3.3 reference manual. `http://www-unix.mcs.anl.gov/AR/otter/`.
[9] A. Riazanov and A. Voronkov, 'The design and implementation of VAMPIRE', *AI Communications*, **15**, 91–110, (2002).
[10] S. Schulz, 'E: A brainiac theorem prover', *AI Communications*, **15**, 111–126, (2002).
[11] J. Slaney, 'FINDER: Finite Domain Enumerator', in *Proceedings of the 12th Conference on Automated Deduction (CADE)*, pp. 798–801, (1994).
[12] G. Sutcliffe and C. Suttner. CASC: CADE Automated Systems Competition. `http://www.tptp.org/CASC`.
[13] G. Sutcliffe and C. Suttner. TPTP: Thousands of Problems for Theorem Provers. `http://www.tptp.org`.
[14] T. Tammet, 'Gandalf', *Journal of Automated Reasoning*, **18**, 199–204, (1997).
[15] C. Weidenbach, 'SPASS–vewrsion 0.49', *Journal of Automated Reasoning*, **18**, 247–252, (1997).