SRI International

Technical Note 523 • March 1993

Automated Theorem-Proving Research in the Fifth Generation Computer Systems Project: Model Generation Theorem Provers

Prepared by:

Mark E. Stickel
Principal Scientist
Artificial Intelligence Center
Computing and Engineering Sciences Division

This research was supported by the National Science Foundation under Grant CCR-8922330. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the National Science Foundation or the United States government.

333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

Abstract

One of the successful outcomes of the Fifth Generation Computer Systems Project is the development of *Model Generation Theorem Provers (MGTPs)*. MGTPs have solved previously open problems in finite algebra, produced rapid proofs of condensed detachment problems, and are providing an inferential infrastructure for knowledge-processing research at ICOT. They successfully exploit the Fifth Generation Project's KL1 logic programming language and parallel inference machines to achieve high performance and parallel speedup. This paper describes some of the key properties of MGTPs, reasons for their successes, and possible areas for future improvement.

Keywords

automated theorem proving, model generation, MGTP, SATCHMO

1 Introduction

Research on *Model Generation Theorem Provers (MGTPs)* is a recent activity of the Fifth Generation Computer Systems Project. It started only in the 3-year final stage of the 10-year program, but became one of the principal research initiatives of ICOT's Fifth Laboratory. This research effort has been quite successful. Hasegawa and Fujita [7] provide an extensive summary of MGTPs and their applications.

There is no single Model Generation Theorem Prover. Instead, two families of theorem provers (ground and nonground MGTPs) share the MGTP name but little else. In particular, ground and nonground MGTPs are conceptually quite different and have essentially no code in common.

Ground MGTP (MGTP/G) has the more innovative design and has the distinction of having solved a number of open problems in finite algebra. Specifically, it was able to find or determine the nonexistence of qnasigroups (Latin squares) with certain additional properties. This is basically a finite (but huge!) enumeration task, which benefited substantially from execution on ICOT's Parallel Inference Machine (PIM) with 256 processing elements (PEs). In the ICOT environment of the KL1 language running on PIMs, signal features of MGTP/G are its ability to use KL1's restricted form of unification and its suitability for parallel execution.

Nonground MGTP (MGTP/N) is more conventional and offers a small subset of the capabilities of the powerful OTTER theorem-proving pro-

gram [9] developed at Argonne National Laboratory. MGTP/N has been tested primarily on problems involving condensed detachment, a recent problem domain for OTTER. MGTP/N has in a number of cases solved problems with much smaller search spaces than OTTER, and has sometimes solved problems that OTTER failed to solve. OTTER and MGTP/N execute similar deductive closure algorithms for these problems, but MGTP/N often finishes sooner after generating fewer clauses because it employs lookahead search for a solution. Achieving good parallel performance in MGTP/N was quite a challenge that required substantial software engineering and experimentation.

Although these systems are quite different, their joint development forms a coherent research program. A major objective of the research on MGTPs was to combine logic programming and automated theorem proving. MGTP/G has achieved this to some extent and its extension to a new bottom-up programming language system is contemplated. However, MGTP/G has some deficiencies that complementary research on MGTP/N is meant to overcome. MGTP/G requires that all inferred results be ground. This is frequently an acceptable restriction, but far from always. MGTP/N allows nonground consequences. MGTP/G is designed so that case-splitting on non-Horn clauses is the principal source of parallelism. Thus, little parallel speedup exists for the important case of Horn problems. MGTP/N, which is restricted to Horn problems, is designed to achieve good parallel speedup on them. The techniques to enable parallel speedup in MGTP/N can be folded into MGTP/G to provide it good parallel performance on problems with little or no case-splitting.

2 Ground Model Generation Theorem Prover

The inference system of ground MGTP (MGTP/G) is basically the same as that of the SATCHMO theorem-proving program [8]. The overall objective of these theorem provers is to generate a set of models of an input set of clauses (hence the name *model generation*). Model generation is not the objective of most theorem-proving programs. However, many theorem-proving programs prove theorems by demonstrating the unsatisfiability (absence of a model) of their negations. Model generation is more general, demonstrating absence of a model for unsatisfiable sets of clauses, and producing models for satisfiable ones.

The operation of MGTP/G can be understood partially in terms of the

hyperresolution inference rule [14] that employs rules (implications) to derive new facts from old ones. Let $A_1, \ldots, A_m, A'_1, \ldots, A'_m$ and $C_1, \ldots, C_n$ ($m \geq 0$ or $n \geq 0$) be atomic formulas. Then the inference

$$\frac{\begin{array}{c} A'_1 \\ \vdots \\ A'_m \\ A_1 \wedge \cdots \wedge A_m \rightarrow C_1 \vee \cdots \vee C_n \end{array}}{C_1\sigma \vee \cdots \vee C_n\sigma}$$

where σ is the most general unifier of the pairs $\langle A'_1, A_1 \rangle, \ldots, \langle A'_m, A_m \rangle$ is a special case of hyperresolution. If $n = 0$, the empty disjunctive fact is derived signifying a contradiction. If $n = 1$, an atomic formula is derived; it can be used as the input to further hyperresolution inferences. If $n > 1$, a disjunctive fact has been derived; it cannot be used as input to this restricted form of hyperresolution, which requires facts to be atomic formulas.

MGTP/G resolves the mismatch between the atomic-formula inputs to this restricted hyperresolution rule and the possibly non-atomic-formula output by employing a case-splitting rule as well as hyperresolution. Disjunctive facts $C_1 \vee \cdots \vee C_n$ ($n > 1$) result in the problem space being split into n cases, one for each C_i. Let $\mathcal{M}(S)$ denote the set of models of set of clauses S. The logical justification for the case-splitting rules is that $\mathcal{M}(S \cup \{C_1 \vee \cdots \vee C_n\}) = \mathcal{M}(S \cup \{C_1\}) \cup \cdots \cup \mathcal{M}(S \cup \{C_n\})$. However, this is true only if $C_1 \vee \cdots \vee C_n$ is *ground* (contains no variables); otherwise the cases may contain shared variables that must have consistent value assignments. MGTP/G requires facts to be ground; this is ensured by the range-restrictedness property.

The input to MGTP/G is *range-restricted* if every fact A'_i is ground and every rule $A_1 \wedge \cdots \wedge A_m \rightarrow C_1 \vee \cdots \vee C_n$ has the property that every variable in $C_1 \vee \cdots \vee C_n$ also occurs in $A_1 \wedge \cdots \wedge A_m$. When regarding facts and rules as clauses, range-restrictedness requires that every variable that occurs in a positive literal of a clause must occur in a negative literal of the clause. With range-restrictedness, facts derived by the hyperresolution operation are always ground, and case-splitting becomes feasible.

Case-splitting is the principal source of parallelism in MGTP/G. There is virtually no overhead for processing cases in parallel, as there are no shared variables and the separate searches for models proceed independently. Problems with enough cases run with near-linear speedup on PIMs with as many as 256 PEs.

Non-range-restricted problem	Problem in range-restricted form
`d(X,X)`	`dom(X) -> d(X,X)`
`d(X,Y) , d(Y,Z) -> d(X,Z)`	`d(X,Y) , d(Y,Z) -> d(X,Z)`
`p(X) ; d(g(X),X)`	`dom(X) -> p(X) ; d(g(X),X)`
`p(X) ; l(1,g(X))`	`dom(X) -> p(X) ; l(1,g(X))`
`p(X) ; l(g(X),X)`	`dom(X) -> p(X) ; l(g(X),X)`
`l(1,X) , l(X,a) -> p(f(X))`	`l(1,X) , l(X,a) -> p(f(X))`
`l(1,X) , l(X,a) -> d(f(X),X)`	`l(1,X) , l(X,a) -> d(f(X),X)`
`l(1,a)`	`l(1,a)`
`p(X) , d(X,a) -> false`	`p(X) , d(X,a) -> false`
	`dom(1)`
	`dom(a)`
	`dom(X) -> dom(f(X))`
	`dom(X) -> dom(g(X))`

Figure 1: Example of transformation to range-restricted form.

An extremely fortunate side effect of the range-restrictedness condition is that the hyperresolution operation no longer requires full unification since facts are always ground. MGTP is implemented in the KL1 language that provides pattern matching but not full unification when matching a goal with a clause head. KL1's "one-way unification" is sufficient for MGTP/G. Full unification with the occurs check is unnecessary, and all necessary unification is performed with high efficiency by KL1's built-in unification.

Non-range-restricted problems can be transformed into range-restricted problems by addition of a "dom" predicate. Negative dom predicate literals are used to qualify clauses with non-range-restricted variables. The dom predicate is defined to include all ground terms of the problem. Figure 1 contains an example of the transformation to range-restricted form.[1]

For what class of problems is MGTP/G well-suited? MGTP/G offers no advantage for Horn problems. Horn problems are those for which $0 \leq n \leq 1$ for every rule $A_1 \wedge \cdots \wedge A_m \rightarrow C_1 \vee \cdots \vee C_n$. There is no case-splitting in Horn problems. For range-restricted Horn problems, MGTP/G inference is the same as hyperresolution. For non-range-restricted Horn problems, the transformation to range-restricted form results in all input and derived facts being ground, resulting in less general and compact results

[1]The problem itself is unimportant and will not be discussed further. For the curious, it is a small, standard theorem-proving exercise for proving existence of prime divisors. The predicates "p", "d", and "l" stand for "prime", "divides", and "less than"; "f" and "g" are Skolem functions.

```
dom(1).
dom(2).
:
dom(12).
dom(M) , dom(N) -> p(M,N,1) ; p(M,N,2) ; ...  ; p(M,N,12).
dom(M) -> p(M,M,M).
p(M,N,K1) , p(M,N,K2) , {K1≠K2} -> false.
p(M,N1,K) , p(M,N2,K) , {N1≠N2} -> false.
p(M1,N,K) , p(M2,N,K) , {M1≠M2} -> false.
p(Y,X,U) , p(U,Y,V) -> p(V,Y,X).
p(M,12,K) , {K+1<M} -> false.  % omit some isomorphic copies
```

Figure 2: Clauses for generating idempotent $(yx.y)y = x$ quasigroups.

than hyperresolution operating directly on non-range-restricted clauses.

MGTP/G is usually unsuitable for problems with function symbols, and especially so for non-range-restricted problems with function symbols. Such problems have infinite search spaces and the transformation to range-restricted form yields a definition for the dom predicate that blindly generates elements of the Herbrand universe, as in the example of Figure 1. Infinite search spaces in general require fairness to ensure completeness. The issue is not addressed in MGTP/G, whose design is restricted to finite problems.

Thus, MGTP/G is particularly suitable for function-free, non-Horn problems. The following is a slight variant of the MGTP/G input for one of the previously open problems in finite algebra that it recently solved [2, 6]. These problems fall neatly into the ideal class of function-free, non-Horn problems, with finite, but huge and branchy search spaces. Figure 2 contains the clauses for constructing idempotent quasigroups of order 12 that satisfies the equation $(yx.y)y = x$. The formula `p(X,Y,Z)` is interpreted as $xy = z$ in the quasigroup. MGTP/G was able to determine that no such quasigroup exists, after exploring nearly 3,000,000 branches in the search space. This took less than four hours on a 256-PE PIM and would have required weeks on a single processor.

MGTP/G is essentially an "industrial strength" version of SATCHMO. SATCHMO and MGTP/G attempt to produce the deductive closure of a set of facts under a set of rules. The growing set of facts comprise a possible partial model. The key operation for deriving new facts is the conjunctive matching of facts $A'_1, \ldots, A'_m$ in the current candidate model with rule an-

tecedent $A_1 \wedge \cdots \wedge A_m$. It is vital that rules not be reapplied to the same facts. For example, suppose there are M facts and a single rule that has already been applied to all M^m combinations of the facts. The naive strategy for computing the new closure of those M facts plus one additional one would involve examining $(M+1)^m$ combinations, M^m of which had already been done. SATCHMO does this recomputation; MGTP/G does not. MGTP/G computes only the new combinations. In database system terminology, SATCHMO uses the naive strategy for computing a deductive closure, while MGTP/G uses the seminaive strategy [3].

"Partial falsification" is a second substantial engineering improvement of MGTP/G over SATCHMO. Instead of eagerly splitting on disjunctive facts $C_1 \vee \cdots \vee C_n$ $(n \geq 2)$ as soon they are generated, it is generally better to choose from a set of them one with smallest n for splitting. Moreover, prior to making this selection, it is best to eliminate atoms C_i that a little lookahead reveals will lead immediately to failure. The disjunctive fact with smallest n after such obviously false literals are eliminated by this partial falsification step is chosen for splitting. C_i is eliminated by partial falsification if it plus facts of the current candidate model match the antecedent of some rule $A_1 \wedge \cdots \wedge A_m \rightarrow \mathit{false}$. Partial falsification resulted in orders of magnitude of improvement in performance on the finite algebra problems.

MGTP/G has become a core technology for the Fifth Generation Computer Systems Project. It has been used in applications such as truth maintenance, legal reasoning, program synthesis, and natural-language parsing [7]. Excellent work has also been done on encoding in MGTP/G other forms of reasoning such as abduction, modal reasoning, nonmonotonic reasoning, and logic programming with negation as failure [7]. MGTP/G is important in part because case-splitting search in MGTP/G is dual to backtracking search in Prolog, a feature that was lost when ICOT shifted focus to committed choice languages to facilitate parallel execution. There is a question of whether programming in committed choice languages should be regarded as *logic* programming. But programming for MGTP, written in the KL1 committed choice language, can certainly be considered logic programming.

3 Nonground Model Generation Theorem Prover

MGTP/G has some deficiencies: it is applicable only to range-restricted problems and has poor parallel performance in the absence of case-splitting. It is thus notably weak in the very important case of non-range-restricted

Horn problems. Nonground MGTP (MGTP/N) is designed for this class of problems. MGTP/N also uses the hyperresolution inference rule, but input and derived facts need not be ground, and there is no case-splitting since consequences must be single atoms. Unit hyperresolution is the only inference rule. MGTP/N has been extensively tested on problems involving condensed detachment [11], a recent problem domain for OTTER, and the subject of several challenge problems for theorem-proving programs posed by Ross Overbeek [13], who brought these problems to ICOT's attention. An example is to prove that XGK is a single axiom for the equivalential calculus by proving known single axiom PYO from it. This is done by refuting:

```
p(X) , p(e(X,Y)) -> p(Y).
p(e(X,e(e(Y,e(Z,X)),e(Z,Y)))). %XGK
p(e(e(e(a,e(b,c)),c),e(b,a))) -> false.  %PYO
```

The first clause is the condensed detachment rule that states that y is provable in the equivalential calulus if x and $e(x, y)$ are. These problems had several advantages as test problems in a new theorem-proving project. Their inference system requirements are small: hyperresolution is sufficient and no equality reasoning is required. The problems have few clauses; this was beneficial in early stages of development of the theorem prover when the input was effectively hand compiled. Finally, the problems are hard: they have large search spaces and long proofs. This forced researchers to focus on search strategy and efficiently coping with large numbers of clauses.

Ground MGTP fit the KL1 and PIM environment well. KL1 unification sufficed for range-restricted problems, and case-splitting provided ample parallelism with little overhead. Neither is the case for nonground MGTP.

The hyperresolution rule in MGTP/N requires full unification with the occurs check. MGTP/G was able to represent logic variables by KL1 variables; MGTP/N could not. For MGTP/N, it was necessary to represent variables as integers and substitutions as arrays indexed by those integers and to code a unification algorithm, just as one would do in a language like C. Not being able to rely on KL1's built-in and having to program unification in such a high-level language resulted in remarkably little performance loss—unification on a PIM processing element has been measured at a respectable 1/4 of the speed of unification in OTTER (written in C) running on the approximately 2-times faster SPARC-2 processor.

The MGTP/N inference system lacks case-splitting as a source of lots of low-overhead parallelism. The operation of the system had to be decomposed into separate stages—generation of new clauses, determining whether

they are subsumed, and testing if they complete a refutation—and the system constructed as a collection of generator, subsumer, and tester components. High speedup factors have been achieved, but only after much experimentation with different ways of organizing these tasks.

MGTP/N computes the deductive closure of a set of clauses in the same manner (by the seminaive strategy again) as OTTER. MGTP/N operates on derived facts in strict first-in-first-out order, unlike OTTER, which ordinarily orders them by increasing size (although OTTER's ratio strategy used for condensed detachment problems interleaves selecting clauses in order by size and in first-in-first-out order). Clauses exceeding a user-specified size are discarded; this is the only heuristic control in MGTP/N. Size limits are used in OTTER as well. In OTTER, size limits primarily serve to control memory usage and have little effect on search order since clauses are ordinarily operated on in order by size. Large clauses that might be deleted would in any case be operated on with low priority. The choice of size limit is more critical in MGTP/N with its first-in-first-out processing order—clauses will be processed in the order in which they were generated, unless they are deleted because of the size limit.

The use of first-in-first-out order is controversial. In OTTER, ordering by size generally works much better than first-in-first-out. One motivation for MGTP/N's use of first-in-first out ordering is the desire to process clauses in parallel coupled with the desire to have exactly reproducible results regardless of the number of PEs being used (this is a strong, possibly unncessary, constraint that nevertheless facilitated experimentation). In the purely sequential execution of OTTER, the smallest, previously unselected clause is selected for processing, and all its consequences are inserted into the size-ordered list; the process is then repeated, with the next selected clause often being one of the consequences of the immediately previously selected clause. Reproducing this behavior in MGTP/N would require excessive serialization, forcing selection of which clause to process next to be delayed until processing of the previous clause is completed.

Lazy model generation and lookahead represent a significant departure from the functioning of hyperresolution in OTTER. They account for MGTP/N's ability to prove theorems after having stored far fewer clauses than OTTER. Hyperresolution derives positive clauses. The final inference of a hyperresolution refutation matches unit positive clauses with all the literals of a negative input clause.

OTTER repeatedly selects a clause for inference and derives new clauses by hyperresolution between the selected clause and all previously selected

clauses (i.e., those in the axioms or usable list); these new clauses become candidates for future selection for inference if not subsumed (i.e., they are placed on the set of support list). If the empty clause is derived, the refutation is complete. In this procedure, a refutation cannot be completed until all the unit positive clauses necessary to match a negative input clause have been selected for inference. (Actually, OTTER is not quite this strict. It will recognize that the refutation can be completed immediately after generating a positive unit clause if it matches a unit negative input clause.) With lazy model generation, no new results are derived by hyperresolution until it has been determined that the current set of positive clauses, selected and unselected (i.e., those in the axioms or usable list *and* those in the set of support list), do not falsify a negative input clause. This extends OTTER's recognition of completability of a refutation from unit negative input clauses to nonunit negative input clauses. Being able to recognize completability of a refutation at the time the last necessary piece is derived instead of when it is selected can result in much faster proofs with many fewer stored clauses.

This improvement alone would have no effect on the solution of condensed detachment problems compared to OTTER, since condensed detachment problems have only unit negative input clauses, and the effect of lazy model generation for unit negative input clauses is already present in OTTER. The refutation-completability test can be enhanced by lookahead—allowing the test to use not only all derived clauses, but their immediate consequences as well. The consequences are derived solely to test for immediate completability of a refutation and are discarded afterward; they may be rederived when and if their parent clauses are selected for inference. Refutations can be completed after selecting far fewer clauses for inference, at the expense of some recomputation. Using lazy model generation and lookahead, proofs of condensed detachment problems have often been completed after storing far fewer clauses than OTTER. MGTP/N has also proved some condensed detachment problems that OTTER has not yet succeeded in proving.

4 Evaluation

4.1 Ground MGTP

MGTP/G is limited to finite problems and thus is capable of a limited style of theorem proving. Function-free theories can be finitely instantiated to ground theories and processed by theorem provers for the propositional

calculus. Although limited, such theories are often sufficient for applications like truth maintenance, natural-language parsing, and deductive databases. MGTP/G is a niche theorem prover, although the niche appears to be a useful one.

The basic idea of MGTP/G comes from SATCHMO, but MGTP/G has been refined into a powerful, practical tool, with obvious inefficiencies of the original SATCHMO overcome.

I would like to compare MGTP/G to the Davis-Putnam procedure [4], the classic method for propositional calculus theorem proving. The two have much in common—the Davis-Putnam procedure also attempts to construct a model of its input and employs case-splitting. The most obvious difference is that MGTP/G allows nonground input. The Davis-Putnam procedure can be applied to MGTP/G problems after first creating all ground instances of the input clauses. However, MGTP/G's nonground representation has substantial practical importance. For example, consider the ground instantiation of the clauses in Figure 2: the clause `p(Y,X,U) , p(U,Y,V) -> p(V,Y,X)` has four variables and can thus be instantiated in 12^4 ways. My input to the Davis-Putnam procedure for this problem contained more than 48,000 clauses and 118,000 literals. Coping with problems of this size demands good data structures and fast operations. I have found some problems of this size feasible using the discrimination tree representation for sets of propositional clauses recently proposed by de Kleer [5]. Problems with only slightly larger domains or more variables per clause could easily have ground instantiations too large to hold in computer memory.

The following table shows the number of branches in the search space and the time in seconds for MGTP/G and an implementation of the Davis-Putnam procedure on some of the harder quasigroup problems tested. The finite algebra problem in Figure 2 is named QG5.12 in the table. Problems flagged with an asterisk were listed as open problems in [2] before being solved by MGTP/G. Although the formulations given to the two systems differed slightly, the search space was basically the same, and the same naive heuristic for choosing case-splits (i.e., split a shortest positive clause after all unit simplifications have been done) was employed by both systems. MGTP times are in seconds for a 256-PE PIM; DP times are in seconds on a SPARC-2 workstation running Lucid Common Lisp.

Problem	MGTP Branches	MGTP Time	DP Branches	DP Time
QG1.8	180,430	1,894	236,851	20,906
QG3.9	312,321	1,022	83,630	7,158
QG4.9	315,025	1,127	116,476	10,114
*QG5.12	2,749,676	13,715	6,832	4,524
*QG6.12	2,420,467	8,300	13,488	10,487
*QG7.10	1,451,992	2,809	12,972	6,628

MGTP/G is clearly competitive. Several additional conclusions can be drawn from the table:

- The problems are apparently hard. The searches are large and time-consuming for both systems.
- MGTP often required much less wall-clock time. Researchers can obtain results more quickly using MGTP on PIM. This is an unarguable benefit.
- MGTP required much more total CPU time. MGTP's single processor performance is substantially less than DP's.
- MGTP's search space was often larger and sometimes much larger. DP is exploiting some problem constraints for more unit simplification that MGTP overlooks.

It should be noted that both MGTP/G and the Davis-Putnam procedure were designed as general-purpose theorem provers and are not specialized to quasigroup problems. Slaney's FINDER [15] has also been used to solve quasigroup problems. It is also a general-purpose program, but was designed for constraint satisfaction rather than theorem proving. MGTP/G and FINDER are probably not essentially very different from the Davis-Putnam procedure, although there may be substantial differences in detail.

One difference in detail that probably accounts for much of the difference in search-space size between MGTP/G and the Davis-Putnam procedure is the amount of simplification by unit clauses that is possible. All instances of partial falsification in MGTP/G are handled by unit simplification in the Davis-Putnam procedure, but the converse is not true.

Consider the rule `p(Y,X,U) , p(U,Y,V) -> p(V,Y,X)` in the quasigroup problem. In MGTP/G, this is used only for forward reasoning to derive instances of `p(V,Y,X)`. In the Davis-Putnam procedure, it is just a clause.

In particular, if an instance of p(Y,X,U) is true and p(V,Y,X) is false, the negative unit clause -p(U,Y,V) can be derived and used in unit simplification. MGTP/G assigns asymmetric roles to positive and negative clauses. Positive clauses are model elements, mixed clauses are generators, and negative clauses are constraints. But assigning roles of code versus data to clauses may result in a failure to recognize and exploit useful relationships among them. A challenge for MGTP/G is to be more effective in this regard without losing its procedural nature and efficiency.

A second critical challenge for MGTP/G is to be able to cope with or avoid irrelevant case-splitting. The search space can easily be made arbitrarily large by adding extra irrelevant positive clauses to split. In many applictions, it is not realistic to expect irrelevant clauses to be absent. Some work in this direction has been done for SATCHMO [17]. The Davis-Putnam procedure shares the same defect, but there is an elegant solution [12]. Case-splitting in the Davis-Putnam procedure uses the property that $\mathcal{M}(S) = \mathcal{M}(S \cup \{p\}) \cup \mathcal{M}(S \cup \{\neg p\})$. If $S \cup \{p\}$ has no models and keeping track of dependencies shows that the demonstration of its unsatisfiablity did not depend on p, then it can be recognized that $S \cup \{\neg p\}$ also has no models and search for them can be eliminated. A challenge in applying this rule to MGTP/G is that it is sequential: it analyzes the search for models of $S \cup \{p\}$ before deciding whether to search for models of $S \cup \{\neg p\}$, which would reduce parallelism in MGTP/G.

Finally, the efficiency of MGTP/G's implementation deserves further scrutiny. It is hard to evaluate the effects of the different representations (restricted first-order for MGTP, propositional for DP), and different hardware and software bases (KL1 on PIM versus Lisp on SPARC-2), but it is nevertheless true that, for the quasigroup problems, MGTP running on 256 processors was at best 10-times faster than DP running on a single processor. There has been much effort on speeding up the key "conjunctive match" operation—finding sets of positive clauses in the current model that simultaneously unify with all the negative literals in a negative or mixed clause—but I do not think the final answer has yet been found. I believe that term indexing will be part of the final answer.

4.2 Nonground MGTP

MGTP/N is not restricted to finite problems like MGTP/G. It is closer in spirit than MGTP/G is to conventional theorem-proving programs like OTTER, although it is restricted to Horn clauses and uses only the hyper-

resolution inference rule. Allowing infinite search spaces makes it closer to "real" theorem proving. MGTP/N has succeeded in proving a number of difficult (for automated theorem provers) condensed detachment problems. Despite its successes, MGTP/N is a less complete system than MGTP/G.

The most often used versions of MGTP/N lack some crucial features, such as backward subsumption and ordering clauses for inference in other than first-in-first-out order, let alone additional inference rules such as those for equality reasoning. All of these topics have received some investigation, but have not been elevated to common usage in MGTP/N.

Indexing clauses for fast, selective retrieval for unification and subsumption operations is critical to the performance of large-scale theorem proving [10], but has only recently been incorporated into some versions of MGTP/N, where it has produced substantial performance gains. Indexing is absent in MGTP/G as well, and its absence does create a performance problem. However, the problem is mitigated by the comparatively small number of model elements in examples of interest (e.g., a model of an order n quasigroup contains only n^2 literals) compared to MGTP/N, which may store thousands of clauses.

The major innovation of MGTP/N is lazy model generation and lookahead. These are the features that have enabled it to substantially outperform OTTER on a number of examples. ICOT has created in this approach a useful, substantial modification of the deductive closure operation of OTTER. This refinement is often stunningly effective. On condensed detachment problems when using first-in-first-out order, lazy generation and lookahead often enable MGTP/G running on a single PE to beat OTTER running on a SPARC-2 and to be 200–1000 times faster when running on a 256-PE PIM.

Such use of lookahead is not unprecedented, though it has rarely been so convincingly demonstrated. The TERMINATOR module [1] played a similar role in the Markgraf Karl refutation procedure. In that system, as each clause was derived, a bounded search for a unit refutation starting from the new clause was conducted before resuming generation of new clauses. The linked inference principle [16] also has lookahead properties. However, as it is used in OTTER, linked inference, like ordinary inference, draws conclusions only from previously selected clauses (i.e., those in the axioms or usable list). MGTP/N's lookahead only tries to derive the empty clause but operates on all clauses, not just those previously selected (i.e., it operates on clauses in the set of support list as well as those in the axioms or usable list).

In the future, support for ordering clauses for inference in non-first-in-

first-out order is vital. This means allowing best-first as well as breadth-first search. Not only does best-first search, which in practice usually means ordering clauses by their size, often perform much better, but ordering clauses by size instead of age makes the maximum size for stored clauses a much less critical parameter for performance of the system. It is harder to achieve high parallel speedup with best-first search than with breadth-first search. Although near-linear speedups on 256 PEs have been shown for MGTP/N using breadth-first search, to date this has been achieved on only 64 PEs for best-first search.

It would also be nice to see MGTP/N extended to additional inference operations, such as for equality, and to formulate equally effective lookahead strategies for these rules.

Again, as with MGTP/G, I believe the basic operation of MGTP/N can be improved. MGTP/N's superior performance compared to OTTER owes more to changed search strategy than efficiency. For example, prior to the recent experiments with term indexing, subsumption testing entailed matching newly derived clauses against every previous clause.

5 Conclusion

The Fifth Generation Computer Systems Project has created in MGTP/G a useful tool for reasoning in artificial intelligence applications and found a paradigm for their continued research in automated theorem proving. MGTP/G has already proven useful in solving open problems in finite algebra, an application that was not anticipated when the program was being developed. I see more analysis to reduce branching as an important area for future improvement.

MGTP/N is less mature. In many ways, developing MGTP/N is harder than developing MGTP/G. MGTP/N does not benefit as easily from the FGCS Project's KL1 and PIM technology. Full unification is necessary and lots of low-overhead parallelism from case-splitting is absent. Given the difficulties, getting near-linear speedup for MGTP/N running on large multiprocessors, as they sometimes have, is a heroic achievement. There are many requirements, such as indexing, backward subsumption, non-first-in-first-out clause order, and additional inference rules such as for equality, that will take much more time to adequately address. Nevertheless, even in its preliminary form, MGTP/N has demonstrated some success. The major contribution of MGTP/N to date is development and analysis of lazy model

generation and lookahead. They have allowed MGTP/N to occasionally beat OTTER by a substantial margin at its own game of solving condensed detachment problems. The methods are not entirely new, but are not as widely used as they should be.

As relatively little research in automated theorem proving has been done in Japan in the past, it is a pleasure to report on this work, which represents a substantial increase in Japanese interest and activity in automated theorem proving. The research on Model Generation Theorem Provers at ICOT has achieved a great deal in a short period of time. These systems are still young, but improving rapidly, so we can hope for more in the future.

References

[1] Antoniou, G. and H.J. Ohlbach. TERMINATOR. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, 916–919.

[2] Bennett, F.E. and L. Zhu. Conjugate-orthogonal Latin squares and related structures. In J.H. Dinitz and D.R. Stinson (eds.). *Contemporary Design Theory: A Collection of Surveys.* Wiley, New York, 1992.

[3] Date, C.J. *An Introduction to Database Systems.* Addison-Wesley, Reading, Massachusetts, 1986.

[4] Davis, M. Eliminating the irrelevant from mechanical proofs. *Proceedings of the Symposia of Applied Mathematics, Volume 15.* 1963, 15–30.

[5] de Kleer, J. An improved incremental algorithm for generating prime implicants. *Proceedings of the AAAI-92 Tenth National Conference on Artificial Intelligence*, San Jose, California, July 1992, 780–785.

[6] Fujita, M., J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. Unpublished, 1992.

[7] Hasegawa, R. and M. Fujita. Parallel theorem provers and their applications. *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, Tokyo, Japan, June 1992, 132–154.

[8] Manthey, R. and F. Bry. SATCHMO: a theorem prover implemented in Prolog. *Proceedings of the 9th International Conference on Automated Deduction*, Argonne, Illinois, May 1988, 415–434.

[9] McCune, W. OTTER 2.0 users guide. Technical Report ANL-90/9, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, March 1990.

[10] McCune, W. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning 9*, 2 (October 1992), 147–167.

[11] McCune, W. and L. Wos. Experiments in automated deduction with condensed detachment. *Proceedings of the 11th International Conference on Automated Deduction*, Saratoga Springs, New York, June 1992, 209–223.

[12] Lee, S.-J. and D.A. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning 9*, 1 (August 1992), 25–42.

[13] Overbeek, R. A proposal for a competition. Unpublished, 1990.

[14] Robinson, J.A. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics 1* (1965), 227–234.

[15] Slaney, J.K. FINDER, finite domain enumerator: notes and guide. Technical Report TR-ARP-1/92, Automated Reasoning Program, Australian National University, 1992.

[16] Veroff, R. and L. Wos. The linked inference principle, 1: the formal treatment.. *Journal of Automated Reasoning 8*, 2 (April 1992), 213–274.

[17] Wilson, D.S. and D.W. Loveland. Incorporating relevancy testing in SATCHMO. Technical Report CS-1989-24, Department of Computer Science Duke University, Durham, North Carolina, November 1989.