The Programmable Strategy Theorem Prover: An Implementation of the Linear MESON Procedure*

by

Mark E. Stickel

Abstract

The Programmable Strategy Theorem Prover (PSTP) is a theorem proving program for the first order predicate calculus using the linear MESON procedure as the inference system. The linear MESON procedure is a new variant of the model elimination theorem proving procedure for theories with or without equality which is an affirmation rather than a refutation procedure. It is profitably viewed as an extension of the problem-reduction method. The fundamental element of a linear MESON procedure deduction, the chain, is a representation of a set of goals to be solved and their supergoals. PSTP is designed to be used interactively or in a fully automatic mode. Some features of PSTP are a general mechanism for specifying which chains are to be retained and manipulated, an automatic procedure for storing and retrieving information about chains when this information is requested, the capability of specifying an ordering function which can be used for specifying search strategies, and a powerful set of commands. Results of an experiment testing some simple search strategies and comparisons with results from other theorem proving studies are presented.

*Research supported by NSF Grant GJ-28457X2 and by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-73-C-0074).

0. Introduction

The first section of this report describes a new theorem proving procedure for the first order predicate calculus, the linear MESON procedure, a variant of model elimination. Although the linear MESON procedure is not substantially more powerful than related resolution procedures except in the capability for restriction or differential treatment of inferences by particular implicative forms of the axioms, it represents a significant increase in the "naturalness" of proofs by complete inference systems because it can be viewed as an extension of the problem-reduction method [9].

In addition to the standard "subgoaling" mechanism of the problem-reduction method (by which a goal is replaced by a set of subgoals whose solution constitutes a solution to the goal) represented in the linear MESON procedure by the extension operation, the linear MESON procedure provides a mechanism for solving a goal one of whose subgoals is its logical negation requiring reasoning by contradiction (the reduction operation), an operation for eliminating duplicate goals (factorization), and operations for solving problems involving the equality relation (p-extension, p-reduction) without requiring full axiomatization of its properties. These added inference operations result in a complete inference system for the first order predicate calculus with equality.

The second section describes the underlying concepts and overall design of the Programmable Strategy Theorem Prover (PSTP), a theorem proving program employing the linear MESON procedure as its inference system and designed to be used interactively or fully automatically.

Although restricted to using the linear MESON procedure as its inference system, PSTP provides substantial flexibility in specification of search strategies, both in terms of deletion criteria (such as use of length, level, and depth bounds) and of the order in which inference operations are to be performed (such as depth first, breadth first, or diagonal search strategies).

The third section presents the results of a performance study on a set of problems previously tested in two other theorem proving performance studies. Some conclusions are drawn concerning relative merits of tested search strategies on empirical and philosophical grounds and the power of the linear MESON procedure relative to other procedures as exemplified by results from the other studies.

1. The Linear MESON Procedure

The linear MESON procedure is a variant of the model elimination theorem proving procedure [6,7,8] in which (1) each literal of the top chain and the derived chains of a deduction is replaced by its complement and (2) implications as well as disjunctions are permitted as axioms. It is also the linear form of the MESON procedure [9] specified for goal-subgoal trees (MESON stands for "model elimination subgoal oriented"). Advantages of the linear MESON procedure are (1) the linear MESON procedure, though logically equivalent to model elimination, has the form of an affirmation rather than a refutation procedure and its proofs have a very natural interpretation in terms of goal-subgoal trees and (2) the linear MESON procedure has greater expressive power than alternative procedures in the potential use of implications rather than disjunctions to restrict application of inference operations to certain literals of axioms or (by replacing a disjunction by more than one implication) to facilitate differential treatment of the various implicative forms of axioms during the search for a solution (this capability is shared by model elimination in which each length n input clause generates n auxiliary chains only the last literal of which can be matched in inference operations).

In a theorem proving program permitting interaction between human user and mechanical proof procedure, it is desirable that as human-oriented a procedure as possible be employed. While it is correctly argued that all resolution type theorem proving procedures are machine-oriented and notoriously unsuited to extensive human computation, it is our contention that the linear MESON procedure is more human-oriented than other resolution related procedures.

This is a direct consequence of the relationship of the linear MESON procedure with the problem-reduction method. The linear MESON procedure is an extension of the problem-reduction method which is complete for the first order predicate calculus with equality. It augments the problem-reduction method by inference operations which perform reasoning by contradiction (reduction), which eliminate duplicate subgoals (factorization), and which deal with the equality relation (p-extension, p-reduction).

The linear MESON procedure represents the state of a search for solution as a set of chains. Each chain represents a subtree of the search space. The solution of all the subgoals represented in a chain constitutes a solution of the top goal. Different chains represent different alternative partial solutions of the top goal.

More specific claims of the linear MESON procedure being human-oriented are (1) it is an affirmation rather than a refutation procedure, (2) in keeping literals ordered in a chain, it automatically prevents (in goal-subgoal tree terms) the start of an attempt to solve another subgoal in the chain until the current one is solved, and (3) it is a procedure which remains complete if only input deductions are used. An input deduction is a deduction in which each element of the deduction (a linear MESON procedure chain) is derived by an inference operation applied to its predecessor or its predecessor and an (input) axiom.

In combination, the last two items permit the user to focus his attention on a much smaller subset of the available data than is possible for many resolution based procedures.

In preparing a problem for input to the linear MESON procedure, the input formula is first converted to prenex form with its matrix in the form of a conjunction of assertions implying a conclusion. An assertion is a (possibly empty) conjunction of literals (antecedents) implying a disjunction of literals (consequents). Note that if there are no antecedents in the assertion, the assertion is just a disjunction of literals (disjuncts), i.e., a clause. A conclusion is a conjunction of literals. Schematically, the transformed formula is in the form

$$Q((A^1_1 \wedge \ldots \wedge A^1_{m_1} \rightarrow C^1_1 \vee \ldots \vee C^1_{n_1}) \wedge \ldots \wedge (A^p_1 \wedge \ldots \wedge A^p_{m_p} \rightarrow C^p_1 \vee \ldots \vee C^p_{n_p}) \rightarrow (G_1 \wedge \ldots \wedge G_q))$$

where A, C and G denote literals, Q denotes a list of quantifiers, and $p \geq 1$, $q \geq 1$, each $m_i \geq 0$, each $n_i \geq 1$.

The transformed formula is then Skolemized by (1) replacing for each universal quantifier in the prefix all occurrences of the quantified variable in the matrix by a unique Skolem function with all the existentially quantified variables whose quantifiers precede the universal quantifier in the prefix as arguments and (2) deleting the quantifier prefix.

An alternate input form for the linear MESON procedure is a conjunction of quantified assertions implying a conclusion. Schematically,

$$Q_1(A^1_1 \wedge \ldots \wedge A^1_{m_1} \rightarrow C^1_1 \vee \ldots \vee C^1_{n_1}) \wedge \ldots \wedge Q_p(A^p_1 \wedge \ldots \wedge A^p_{m_p} \rightarrow C^p_1 \vee \ldots \vee C^p_{n_p}) \rightarrow Q_{p+1}(G_1 \wedge \ldots \wedge G_q)$$

where A, C and G denote literals, Q denotes a list of quantifiers, and $p \geq 1$, $q \geq 1$, each $m_i \geq 0$, each $n_i \geq 1$.

This input form is Skolemized by (1) replacing in each assertion all occurrences of each existentially quantified variable by a unique Skolem function whose arguments are the variables of universal quantifiers preceding the existential quantifier, (2) replacing in the conclusion all occurrences of each universally quantified variable by a unique Skolem function whose arguments are the variables of existential quantifiers preceding the universal quantifier, and (3) removing all quantifiers from the formula.

Example. The problem is: if a is a prime number and a times the square of some number u is b then a divides b. The initial formula is:

$$
\begin{aligned}
&\forall x \forall y \forall z \forall w\ (\mathrm{prime}(x) \wedge y{*}z=w \wedge \mathrm{divides}(x,w) \\
&\qquad\qquad \rightarrow \mathrm{divides}(x,y) \vee \mathrm{divides}(x,z)) \\
&\wedge \forall x\ x{*}x=\mathrm{square}(x) \\
&\wedge \forall x \forall y \forall z\ (x{*}y=z \rightarrow y{*}x=z) \\
&\wedge \forall x \forall y \forall z\ (x{*}y=z \rightarrow \mathrm{divides}(x,z)) \\
&\wedge \exists u\ a{*}\mathrm{square}(u)=\mathrm{square}(b) \\
&\wedge \mathrm{prime}(a) \\
&\rightarrow \mathrm{divides}(a,b)
\end{aligned}
$$

The Skolemized form ready for input to the linear MESON procedure is:

$$
\begin{aligned}
&(\mathrm{prime}(x) \wedge y{*}z=w \wedge \mathrm{divides}(x,w) \rightarrow \mathrm{divides}(x,y) \vee \mathrm{divides}(x,z)) \\
&\wedge x{*}x=\mathrm{square}(x) \\
&\wedge (x{*}y=z \rightarrow y{*}x=z) \\
&\wedge (x{*}y=z \rightarrow \mathrm{divides}(x,z)) \\
&\wedge a{*}\mathrm{square}(c)=\mathrm{square}(b) \\
&\wedge \mathrm{prime}(a) \\
&\rightarrow \mathrm{divides}(a,b)
\end{aligned}
$$

Problems with equality (as above) can be introduced without the need for specifying the symmetry, transitivity, and substitutivity axioms if the special equality inference operations (p-extension, p-reduction) are used. The equality reflexive (x=x) and functionally reflexive axioms (e.g., square(x)=square(x), x*y=x*y) theoretically are required. The latter are not present in the above example since no special equality rules are required for the problem's solution. (This is the same as the NUM1 example studied in Section 3.)

By virtue of its derivation from model elimination, the linear MESON procedure is complete (given that a compatible set of inference and postprocessing operations are used) provided (1) the set of assertions is consistent (this requirement is equivalent to the requirement that the top chain of a model elimination deduction be in the minimally unsatisfiable set of input clauses) and (2) either the disjunctive axiom form is used or all implicative forms of each assertion are included among the assertions. The first condition can be eliminated by the addition of a special contradiction mechanism defined for the MESON procedure but not included in this formulation of the linear MESON procedure which permits the proof of any conclusion from an inconsistent set of assertions. The second condition can sometimes be eliminated in practice since it is often clear from the problem structure (as in the case of Horn formulas) that use of a subset of the implicative forms results in the possible deduction of all the chains that the disjunctive form would. Also, although the resulting procedure is, in general, incomplete it is sometimes desirable to restrict the search for a proof by not presenting the procedure with all the implicative forms of the assertions.

The fundamental element of a linear MESON procedure deduction is the chain. A chain is an ordered sequence of literals. Two types of literals are distinguished: A-literals and B-literals. B-literals correspond to the literals present in clauses in resolution theorem proving. A-literals record ancestry information and represent (in goal-subgoal tree terms) higher goals. All the literals in the theorem and axioms are B-literals. An A-literal is created in a newly derived chain from a B-literal in the parent chain when a set of literals whose conjunction implies the A-literal (a set of subgoals whose solution constitutes a solution to the goal represented by the A-literal) is added.

A linear MESON procedure deduction of chain K from problem P is a sequence $K_0,\ldots,K_n$ of acceptable chains where K_0 is the conclusion of P, K_n is K, each K_i ($1 \le i \le n$) is derived from K_{i-1} by extension (by an assertion C_{i-1}), factorization, reduction, p-extension (by an assertion C_{i-1}) or p-reduction, and each C_{i-1} is an assertion of P or a lemma (see lemma formation operation below). A <u>solution</u> of P (a <u>proof</u> of the conclusion of P) is a linear MESON procedure deduction of the empty chain from P. In general, for the inference system to be complete, the negation of the conclusion must be included among the assertions. The definitions of acceptable chains and the inference operations are given below. Example proofs illustrating most forms of the inference operations appear starting on page 17.

<u>Matching</u>. If the two arguments to the matching procedure are terms, the matching procedure returns the most general unifier of the terms. If the two terms are not unifiable, the matching procedure fails.

If the two arguments to the matching procedure are literals and are both <u>positive</u> (unnegated) or both <u>negative</u> (negated) literals with unifiable atomic formulae, the matching procedure returns the most general unifier of the atomic formulae. If the two arguments are not both positive or both negative or the atomic formulae are not unifiable, the matching procedure fails.

<u>Extension</u>. The extension operation takes an acceptable chain K and an implication (alt., disjunction) C as its arguments. Let K' and C' be variable disjoint variants of K and C. If the last literal of K' matches a consequent (alt., disjunct) of C', the chain consisting of K' followed by the antecedents and the complements of the remaining consequents (alt., the complements of the remaining disjuncts) of C' with matching substitution

applied can be inferred. Each literal of the derived chain descended from a literal of K' is designated to be the same type of literal as its ancestor except the last which is designated to be an A-literal; each literal of the derived chain descended from a literal of C' is designated to be a B-literal.

Factorization. The factorization operation takes an acceptable chain K as its argument. If the last literal of K matches a preceding B-literal of K, the chain consisting of K with the last literal removed and with matching substitution applied can be inferred. Each literal of the derived chain is designated to be the same type as its ancestor.

Reduction. The reduction operation takes an acceptable chain K as its argument. If the last literal of K matches the complement of a preceding A-literal of K, the chain consisting of K with the last literal removed and with matching substitution applied can be inferred. Each literal of the derived chain is designated to be the same type as its ancestor.

P-extension. The p-extension ("p-" for paramodulation) operation takes an acceptable chain K and an implication (alt., disjunction) C as its arguments. Let K' and C' be variable disjoint variants of K and C. (a) If a consequent (alt., disjunct) of C' is of the form a=b or b=a where a matches a term in the last literal of K', the chain consisting of K' followed by the antecedents and the complements of the remaining consequents (alt., the complements of the remaining disjuncts) of C' followed by a copy of the last literal of K' with a single instance of the term matching a replaced by b with matching substitution applied can be inferred. Each literal of the derived chain descended from a literal of K' is designated to be the same type of literal as its ancestor except the last which is designated to be an A-literal; each literal of the derived chain descended from a literal of C' is designated to be a B-literal; the last literal of the derived chain (in which an instance of a term matching a was replaced by b) is designated to be a B-literal. This form of p-extension is called p-extension <u>from</u> an assertion. (b) If the last literal of K' is of the form a≠b or b≠a where a matches a term in a consequent (alt., disjunct) of C', the chain consisting of K' followed by the antecedents and the complements of the remaining consequents (alt., the complements of the remaining disjuncts) of C' followed by a copy of the complement of the consequent (alt., disjunct) containing the term matching a with a single instance of that term replaced by b with matching substitution applied can be inferred. Each literal of the derived chain descended from a literal of K' is designated to be the same type of literal as its ancestor except the last which is designated to be an A-literal; each literal of the derived chain descended from a literal of C' is designated to be a B-literal; the last literal of the derived chain (in which an instance of a term matching a was replaced by b) is designated to be a B-literal. This form of p-extension is called p-extension <u>to</u> an assertion.

<u>P-reduction</u>. The p-reduction ("p-" for paramodulation) operation takes an acceptable chain K as its argument. (a) If the last literal of K contains a term matching the term a where a preceding A-literal of K is of the form a≠b or b≠a, the chain consisting of K followed by a copy of the last literal with a single instance of the term matching a replaced by b with matching substitution applied can be inferred. Each literal of the derived chain descended from a literal of K is designated to be the same

type of literal as its ancestor except the last which is designated to be an A-literal; the last literal of the derived chain (in which an instance of a term matching a was replaced by b) is designated to be a B-literal. This form of p-reduction is called p-reduction from an A-literal. (b) If the last literal of K is of the form a≠b or b≠a where a matches a term in a preceding A-literal of K or the last literal of K itself, the chain consisting of K followed by a copy of the preceding A-literal or last literal with a single instance of the term matching a replaced by b with matching substitution applied can be inferred. Each literal of the derived chain descended from a literal of K is designated to be the same type of literal as its ancestor except the last which is designated to be an A-literal; the last literal of the derived chain (in which an instance of a term matching a was replaced by b) is designated to be a B-literal. This form of p-reduction is called p-reduction to an A-literal or self.

If the parent chain K in the p-extension or p-reduction operation is itself derived by p-extension or p-reduction, the created A-literal in the derived chain may optionally be omitted with completeness unaffected (see [8]). There is a tradeoff here. If the A-literal is omitted, the derived chain is shorter and easier to read, and some future possible reductions and p-reductions may be eliminated. On the other hand, especially if the postprocessing operation specifies rejection of chains containing an A-literal followed by an identical A-literal or B-literal, retention of the A-literal may result in rejection of more chains as being unacceptable. For example, this could prevent repeated p-extension by a=b from creating an endless sequence of chains ending alternately in Pa or Pb.

Postprocessing. A postprocessing operation takes a chain K (output from the extension, factorization, reduction, p-extension or p-reduction operation) as its argument and either rejects K as being non-acceptable and thus unusable as input to any inference operation or transforms the chain into an acceptable chain. Many different postprocessing operations can be written with different effects regarding efficiency and completeness. Four postprocessing operations are described in the following table. The table expresses possible relationships between each pair of literals in the chain. All the actions corresponding to true conditions are to be performed on the chain, except, of course, that if the action is to reject the chain then no other conditions need be checked or actions need be performed.

Postprocessing operations

	STRONG-SAVE	STRONG-DELETE	WEAK-SAVE	WEAK-DELETE
Condition	Action	Action	Action	Action
A-literal followed by identical A-literal	reject chain	reject chain	reject chain	reject chain
A-literal followed by complementary A-literal	reject chain	reject chain	reject chain	reject chain
A-literal followed by identical B-literal	reject chain	reject chain	reject chain	reject chain
A-literal followed by complementary B-literal		delete following B-literal		delete following B-literal
B-literal followed by identical A-literal	reject chain	reject chain		
B-literal followed by complementary A-literal	reject chain	reject chain		
B-literal followed by identical B-literal		delete following B-literal		delete following B-literal
B-literal followed by complementary B-literal	reject chain	reject chain		

If all the actions of the postprocessing operation specified in the table have been performed and the chain is not rejected, all terminal A-literals of the chain are deleted. This terminal A-literal deletion is called contraction.

The deletion action associated with the "B-literal followed by identical B-literal" condition is called ground factorization since it represents an instantiation-free usage of a generalized factorization operation which can delete non-terminal B-literals. Similarly, the deletion action associated with the "A-literal followed by complementary B-literal" condition is called ground reduction since it represents an instantiation-free usage of a generalized reduction operation which can delete non-terminal B-literals.

Several additional conditions and actions can be used in postprocessing operations such as the following which are available in PSTP but were not used in the present study: (1) rejecting chains containing a non-terminal A-literal that is an instance of a unit axiom (single literal input assertion), (2) rejecting chains containing a non-terminal A-literal that is an instance of a unit lemma (single literal derived assertion, see lemma formation operation below) by which the ancestor chain could have been extended when the A-literal was created, (3) deletion of B-literals which are instances of unit axioms, (4) deletion of B-literals which are instances of unit lemmas, (5) removal of all literals including and following the A-literal in the case of a B-literal followed by an identical A-literal (this is called factorization truncation), (6) removal of all literals including and following the second A-literal in the case of an A-literal followed by a complementary A-literal (this is called reduction truncation). All of these actions can be shown to preserve completeness.

For problems not involving the equality relation, the WEAK-SAVE and WEAK-DELETE postprocessing operations yield a complete inference system when the extension and reduction operations are used. The WEAK postprocessing operations correspond closely to the chain admissability criteria for model elimination of [6]. Also for problems not involving the equality relation, the STRONG-SAVE and STRONG-DELETE postprocessing operations yield a complete inference system provided the factorization operation is used in addition to extension and reduction. The STRONG postprocessing operations correspond closely to the chain acceptability criteria for strong model elimination of [7]. Note that with these postprocessing operations, if the conclusion is variable free, its negation need not be included among the assertions since extension by the negation of the conclusion in any deduction from the conclusion would result in a non-acceptable chain with an A-literal followed by an identical B-literal or an A-literal followed by a complementary A-literal.

For problems involving the equality relation, a postprocessing operation which rejects a chain only if it has an A-literal followed by a complementary A-literal yields a complete inference system when the extension, reduction, p-extension and p-reduction operations are used. Also for problems involving the equality relation, a postprocessing operation which rejects a chain only if it has a A-literal followed by a complementary A-literal or a B-literal followed by an identical A-literal yields a complete inference system provided the factorization operation is also used. This corresponds closely to the chain permissability criteria for model elimination with paramodulation of [8]. We believe (but have no proof) that for problems involving the equality relation, the WEAK-SAVE and

WEAK-DELETE postprocessing operations yield a complete inference system when the extension, reduction, p-extension and p-reduction operations are used and the STRONG-SAVE and STRONG-DELETE postprocessing operations yield a complete inference system when the factorization operation is also used. The equality reflexive (x=x) and functionally reflexive axioms (e.g., f(x,y)=f(x,y)) theoretically are required in any case.

Lemma formation. An additional inference operation is used to create new assertions during contraction. A new assertion (called a lemma) consisting of the disjunction of the terminal A-literal being removed and all preceding A-literals whose scope (see below) exceeds the number of A-literals between them and the terminal A-literal and the complements of all preceding B-literals whose scope exceeds the number of A-literals between them and the terminal A-literal can be inferred.

The scope of each literal in the conclusion is 0 and the scope of each literal added to a chain in the extension and p-extension operations is 0. In the factorization and reduction operations (and also in ground factorization and reduction performed by the postprocessing operation), the scope of the leftmost involved literal is set to the maximum of its previous scope and the number of A-literals between it and the rightmost involved literal. In the p-reduction operation, the scope of a literal descended from an involved A-literal is set to the number of following A-literals in the derived chain. Each other literal has the same scope as its parent literal in the parent chain. After each contraction operation, the scope of each literal is set to the minimum of its previous scope and the number of A-literals following it in the chain, i.e., no literal will be allowed to have a scope which exceeds the number of following A-literals.

Lemma formation creates assertions from solved goals. Removal of an A-literal by contraction does not mean that the goal it represents has been solved globally, but only that it has been solved in the environment of the chain of which it was a part. Joined in disjunction with the A-literal is the negation of each of the assumptions from the chain used in the solution of the A-literal. Thus, the resulting lemma states either the "solved" A-literal is true or one or more of the assumptions was false. The assumptions which could be used in the solution of the A-literal are the negation of an A-literal (reduction, p-reduction) or a B-literal (factorization). The scope mechanism keeps track of the assumptions made with respect to the solution of each goal.

Lemma formation, while it may generate useful assertions during the search for a proof, is not required for completeness.

Several heuristics are available to eliminate the generation of redundant lemmas such as: (1) the first lemma to be generated after an extension operation followed by zero or more factorization or reduction operations is always redundant and (2) if the chain has two or more terminal A-literals and the lemma associated with one subsumes the lemma associated with another, the second need not be generated (specific cases of this condition can readily be checked by examining the scopes of the literals involved).

Subsumption. Redundant chains and assertions can be eliminated from future use by subsumption. One chain is subsumed by another and can thus be eliminated if an instance of the latter chain is an initial subsequence of the former (sequences of B-literals between A-literals may be freely reordered during the subsumption test). One assertion subsumes another if the chain corresponding to the former assertion subsumes the chain corresponding to the latter (the corresponding chain is formed by making a list of B-literals being the consequents and complemented antecedents of implications or disjuncts of disjunctions) and, in the case where the subsuming assertion is an implication, no disjunct or consequent of the subsumed assertion is matched to an antecedent of the subsuming assertion. The latter provision prevents the subsumption of an assertion by another implicative form of the same assertion both of which may be required for completeness. Stronger subsumption rules are possible (see [8]).

With relation to a search strategy, two additional classes of subsumption are recognized: (1) forward subsumption is the subsumption of a newly created chain or assertion by a previously available chain or assertion and (2) backward subsumption is the subsumption of a previously available chain or assertion by a newly generated chain or assertion. In general, completeness is assured only if, when backward subsumption is used, it is first checked whether the subsuming chain is eliminable by forward subsumption.

Examples. This and the following proofs illustrate the usage of most forms of the inference operations. Chains are represented as linear strings of literals with A-literals bracketed. A-literals represent "opened" goals, i.e., goals for which a solution is currently being attempted in the chain. B-literals represent "unopened" goals, i.e., subgoals for which an attempt for solution has not yet started. Each A-literal is a logical consequence of all the literals to its right; thus, the solution of each B-literal to the right of an A-literal solves the A-literal while also solving all the other A-literals to the right of the solved A-literal. Deductions are represented as a vertical sequences of chains, the ancestor of each derived chain being the chain above it. Each derived chain is annotated to describe its derivation from its ancestor. If a chain is the result of extension or p-extension by an assertion with more than one consequent or disjunct, an alphabetic index is used to designate which consequent or disjunct was used. Indices are "a", "b", "c", etc., reading from right to left. The unannotated chains at the top of each sequence of chains are the axioms. Here and elsewhere in this paper, the conclusion and assertions of a problem will frequently be referred to as theorem and axioms respectively.

This is a proof that $\neg(Pa \leftrightarrow Pb) \rightarrow a \neq b$.

1. $Pa \vee Pb$	first axiom from $\neg(Pa \leftrightarrow Pb)$
2. $\neg Pa \vee \neg Pb$	second axiom from $\neg(Pa \leftrightarrow Pb)$
3. $a \neq b$	theorem to be established
4. $[a \neq b]\ \neg Pb\ \neg Pb$	p-extend to 1b

This operation initiates a proof by contradiction. Assuming $a=b$ (the complement of the literal $a \neq b$), the truth of $\neg Pb \wedge \neg Pb$ contradicts $Pa \vee Pb$.

5. $[a \neq b]\ \neg Pb$	factor

It is only necessary to prove $\neg Pb$ once.

6. $[a \neq b]\ [\neg Pb]\ Pa$	extend by 2a

By axiom 2, if Pa is true then $\neg Pb$ is true.

7. $[a \neq b]\ [\neg Pb]\ [Pa]\ Pb$	p-reduce from A-literal

Again assuming $a=b$ to derive a proof by contradiction, if Pb is true then Pa is true.

8. empty	reduce

In chain 7, subject to the assumption that the conclusion $a \neq b$ is false and that $a=b$, there are the implications $Pb \rightarrow Pa$ and $Pa \rightarrow \neg Pb$. This leads to $Pb \rightarrow \neg Pb$, a contradiction if Pb is true.

Therefore, $\neg Pb$ must be assumed to be true. From chain 5, $\neg Pb \rightarrow a \neq b$, so the theorem is proved.

The following four examples are adapted with some modification from [9].

1. $B \rightarrow C$	
2. $A \wedge \neg D \rightarrow B$	
3. $A \rightarrow C \vee \neg D$	
4. A	
5. C	theorem
6. $[C]\ B$	extend by 1
7. $[C]\ [B]\ A\ \neg D$	extend by 2
8. $[C]\ [B]\ A\ [\neg D]\ A\ \neg C$	extend by 3a
9. $[C]\ [B]\ A\ [\neg D]\ A$	reduce
10. $[C]\ [B]\ A$	factor
11. empty	extend by 4

1. $b \leq a \wedge a \leq b \rightarrow a = b$	
2. $b \geq 0$	
3. $a \leq b$	
4. $b \leq a$	
5. $a \geq 0$	theorem
6. $[a \geq 0]\ b \leq a\ a \leq b\ b \geq 0$	p-extend from 1
7. $[a \geq 0]\ b \leq a\ a \leq b$	extend by 2
8. $[a \geq 0]\ b \leq a$	extend by 3
9. empty	extend by 4

1. $a > 0 \rightarrow a \geq 0$	
2. $\neg b \geq 0$	
3. $a > 0$	
4. $a \neq b$	theorem
5. $[a \neq b]\ a > 0\ \neg b \geq 0$	p-extend to 1
6. $[a \neq b]\ a > 0$	extend by 2
7. empty	extend by 3

1. $a+b=2*c \rightarrow a \neq b \vee a=c$	
2. $b \neq c$	
3. $a+b=2*c$	
4. $a \neq b$	theorem
5. $[a \neq b]\ a+b=2*c\ a \neq c$	extend by 1b
6. $[a \neq b]\ a+b=2*c\ [a \neq c]\ b \neq c$	p-reduce from A-literal
7. $[a \neq b]\ a+b=2*c$	extend by 2
8. empty	extend by 3

2. The Programmable Strategy Theorem Prover

The Programmable Strategy Theorem Prover (PSTP) is a program written in UCI LISP [3] for the DECsystem-10 computer implementing the linear MESON procedure as described in Section 1.

The linear MESON procedure is a good inference system for an interactive, programmable strategy theorem prover since, it being an extension of the problem-reduction method, it is more human-oriented than alternative systems, and its input procedure nature and the relatively small number of operations that can be performed on any chain facilitates the design and use of the programmable search strategy capability.

In addition, the linear MESON procedure is a suitable choice in terms of performance since it appears to perform competitively with other inference procedures when used as the inference system in a fully automatic system. An implementation of the parent model elimination procedure at New York University [4] using a depth first search strategy performed competetively with a theorem prover employing the set of support refinement and unit preference search strategy. Further evidence of the competitiveness of linear MESON procedure based systems will be presented in Section 3.

Chain properties. In the design of a theorem proving program, it is necessary to allow for the computation and retention of certain information about each chain (clause) generated during the search for a proof. An example is the necessity of retaining information on parentage of each chain so a proof can be traced when discovered. Another example is the computation of the length of a chain or the maximum level of function nesting in the chain if length or depth bounds are being employed. It would be wasteful to always compute and store such information since it may not always be needed. Also, the retention of computed information about chains should be contingent on such variables as the computational effort required to compute the information, frequency of use of the information, and cost in memory of storing the information. (A more fundamental objection to always storing computed information about a chain is that the information might change with time. For example, the fact that a particular chain is the shortest chain in memory will probably be falsified in the future.) Another important consideration in the design of an information storage and retrieval mechanism for chains is the ability to define new data which can be optionally computed for any chain.

This last consideration is especially important in an interactive theorem proving program so that the user can cause to be computed whatever information about a chain will be useful to him. It is also an important consideration in the design of a theorem proving program which allows user specification of search strategies.

The property storage and retrieval mechanism for chains in PSTP was designed to possess the following characteristics. With the obvious exceptions of the number of a chain and its ancestry information, no information about a chain is computed unless and until this information is requested. Retaining the information is a user option. A new computable datum about a chain can be defined be merely defining the LISP function which computes the information.

The mechanism used is based on the concept of a chain property list. This is a list of dotted pairs; the first component of each dotted pair is a property name (the access name of a datum about the chain); the second component is the value of the property. The information storage and retrieval mechanism functions in the following way. If the value of the property named, for example, NLIT (this represents in PSTP the number of literals in the chain) is interrogated for a chain, that chain's property list is examined for a dotted pair with first component NLIT. If such a dotted pair is found, its second component is the desired information. If property NLIT does not appear in the chain property list, the LISP function NLIT is evaluated with the chain (including chain property list) as its single argument. The value the function NLIT returns is then the desired information. Further, if the LISP atom NLIT has non-NIL value, the property name NLIT and newly computed value will be added to the chain property list.

New properties are defined by the DP ("define property") function. The DP function takes as arguments a function name, lambda variable list, and expression (just like the UCI LISP DE, DF, and DM functions). It creates a LISP function which performs all the chain property list lookup and modification operations, and evaluates expression for the argument if the property value is not found on the chain property list. For example, NLIT is defined in PSTP by evaluating (DP NLIT (CHAIN) (LENGTH (CDDR CHAIN))) where LENGTH is the LISP function which computes the length of a list and (CDDR CHAIN) is the location of the list of literals of chain CHAIN. (SETQ NLIT T) is then evaluated to order retention of values computed by the NLIT function.

Some of the property functions already defined in PSTP compute the number of A-literals in a chain (NALIT), the number of B-literals in a chain (NBLIT), the total number of literals in a chain (NLIT), the maximum function depth in a chain (DEPTH), the number of variables in a chain (NVAR), the number of LISP CONS operations required to construct a chain (SIZE) (this is a good size or complexity function), and the level (number of inference operations in the derivation) of a chain (NEXPAND).

<u>Chain</u> <u>filters</u>. This property storage and retrieval mechanism supports a higher level chain storage and retrieval mechanism. Filters provide a way of flexibly specifying which chains are to be operated upon and which derived chains are to be stored. Two types of filters are distinguished by usage: input filters and output filters. Input filters are employed by the user to specify which chains are to be operated upon. Only chains "selected by" an input chain filter will be processed. Output filters are used to specify which derived chains are to be retained. A chain must be "accepted by" an output filter to be stored. The general form for a filter is a unary LISP function name or lambda expression which returns a non-NIL value if the chain argument is selected or accepted, NIL otherwise. Several abbreviated forms are also available: (1) an integer selects or accepts a chain with that chain number, (2) a list of integers selects or accepts chains with chain numbers in the list, (3) a three element list (called a triple) consisting of a binary function name and 2 integers or property names selects or accepts chains for which the value of the function applied to the integers and property values is non-NIL, and (4) a list of triples which selects or accepts chains for which each triple has non-NIL value.

This chain storage and retrieval mechanism is very flexible. The user can designate chains for processing directly by number or by the properties they possess and can arbitrarily specify the necessary conditions for a newly derived chain to be stored. This user specification of output filters is a far more general form of the usual specification of bounds in theorem proving programs.

Search strategy specification. One of the most important features of PSTP is its capability for specifying the search strategy to be used in searching for a proof. Several theorem proving programs (e.g., QA3.6 [11]) permit the user to specify a particular combination of refinements of resolution (restrictions on pairs of clauses to be used as input to the resolution operation (e.g., linear, merging, set of support, model refinements)), but the capability for ordering inference operations given a particular refinement of resolution is uncommon. PSTP is, of course, restricted to using the linear MESON procedure with variations restricted to different postprocessing operations, but it does have a general capability for specifying search strategies.

Before describing the search strategy specification capability of PSTP, it is instructive to consider the proof strategy employed in many other theorem proving systems in which search strategy is fixed with possibly a few parameters which the user can specify to tailor the proof search to a particular problem. Thus, the search strategy may be fundamentally depth first or perhaps breadth first with a parameter specifying the permitted amount of look-ahead using unit preference. Much of the control the user has over such systems is the specification of which chains to discard. However, even this decision is severely constrained. Usually, the user is only permitted to specify the values of a few parameters such as the maximum length or function depth of clauses to be retained.

We have seen that output filters generalize the capability of specifying retention of chains. Chain order functions provide the capability of specifying the order of expansion of the search space.

Chain order functions. Associated with each list of chains is the name of an order function. (The order function is actually a chain property function as described above.) Whenever a chain is stored in a chain list, it is inserted according to the numerical value of the corresponding order function applied to the chain. The chains with the smallest values of the order function are stored at the top of the chain list (in case of ties, the more recently stored chains will be on top of the chain list). This maintenance of chain lists in sorted order in combination with the SEARCH and SEARCH2 commands provides a quite general capability for specifying search strategies.

Search commands. The SEARCH command is one of the fundamental functions for automatically expanding the search space. The normal mode of operation is for the SEARCH command to remove the top chain from a chain list, derive all possible immediate successor chains from this chain (this is known as expanding the chain), and store those successor chains selected by an output filter in the original chain list according to its order function. Thus by specifying an order function and using the SEARCH command, the user can specify in what order chains are to be expanded and thus partially control the search strategy. For example, if the default order function (which merely returns 0) is used, the search strategy is a depth first strategy. If the deduction level of a chain is used as order function, the search strategy is a pure breadth first strategy (level saturation). The SEARCH command can be viewed as an implementation of

Nilsson's A* algorithm [10] for graph searching as applied to theorem proving. Each derived chain is a node in the graph and the generation of all immediate successors to a chain by extension, factorization, etc., represents the expansion of a node in the A* algorithm.

Although the SEARCH command is very effective in ordering the expansion of chains, the full expansion of a chain at each step often results in generating a large number of chains that will not be used because the value of the order function for these chains exceeds the maximum order function value of any chain appearing in some proof. This presents two difficulties: (1) unused high order function valued chains fill up memory too quickly and (2) their generation requires extra, unnecessary work. The first of these problems could be solved by specifying an output filter that rejects chains with order function value exceeding a certain amount. However, this solution generates a bounded search strategy, i.e., a parameterized incomplete strategy which may fail to find a proof because the order function maximum is set too low. Moreover, the specification of a bounded search strategy fails to solve the problem of extra work required in the generation of rejected chains.

The solution adopted for PSTP includes a means for specification of ordering of individual inference operations rather than just chains. The form of the value of the order function was generalized to include sorted lists of operations with numerical values. For example, the order function value ((201 REDUCE) (302 EXTEND 6)) could represents the order function value for a chain with previously unperformed operations of reduction and extension by chain number 6. Chains with order function values of this form are inserted into chain lists according to the numerical value of the first specified operation (201 in the example).

The SEARCH2 command is designed to operate on chain lists with order functions of the new form. The SEARCH2 command, rather than deleting and expanding the top chain on the chain list, deletes and performs only the first inference operation of the order function value of the top chain on the chain list. If any inference operations remain in the order function, the top chain is reinserted in the chain list according to the value of the next specified inference operation. Thus, the chain list is always a list of chains ordered according to the minimum value of the unperformed inference operations for that chain. The SEARCH2 command will generate successors of a single chain uninterruptedly only so long as none of the successors of the chain or any other chain on the chain list has an inference operation with lower numerical value than the next inference operation to be performed on the current chain.

The SEARCH2 command can also be viewed as an implementation of Nilsson's A* algorithm for graph searching with each node being a chain and a single inference operation. Expansion of a node now merely consists of applying that inference operation to that chain.

Using the SEARCH2 command, for example, it is possible to use an order function which specifies a depth first search strategy that performs only one inference operation on each level (until a level bound is reached forcing a backup). A more realistic order function for use with the SEARCH2 command was used in the experiment described Section 3. Note that the performance of the single specified inference operation may actually result in the generation of more than one successor chain as, for example, when there are two ways of extending a chain by a particular axiom.

Format functions. An additional mechanism for altering the strategy used by PSTP is accomplished by the use of format functions. Format functions can be used to reformat (edit) chains prior to their storage. For example, a format function can be used to reorder the last B-literals of a chain to accomplish the effect of Kowalski and Kuehner's literal selection function mechanism [5]. A format function is associated with each chain list. Each chain is reformatted according to the order function of the chain list into which it is being stored unless it is already in that format. A chain may be stored in two different formats in two different chain lists (formats are permitted to alter the sequence of literals which constitutes the chain itself, but not the chain property list). Default format functions are those which convert chains to x-standardized or y-standardized form by renaming variables.

<u>Command</u> <u>summary</u>. Following is a brief description of each of most of the PSTP commands. These PSTP commands can be divided into four classes: declarative commands (CHAINLIST, PARAMETERS, POSTPROCESSING, PROBLEM), informative commands (ANCESTRY, COUNT, DISPLAY), manipulative commands (COPY, DELETE, FOR, TRANSFER), and inference commands (EXPAND, SEARCH, SEARCH2). An abbreviated syntax for each command is also presented; linguistic variables are enclosed in angle brackets (e.g., "<sources>") and optional command phrases are enclosed in square brackets (e.g., "[DELETE]" and "[TO <destinations>]"). If any phrase of a command is absent, a default value will be used.

In the descriptions of the commands, the most important linguistic variables are <sources> and <destinations>. A source or destination represents, in general, a chain list and a chain filter. For a chain to be used by a command specifying <sources>, it must be a member of one of the specified chain lists and be selected by the corresponding chain filter. For a chain to be stored in the chain list of a destination by a command specifying <destinations>, it must be accepted by the corresponding chain filter. Note that in the syntax, <sources> and <destinations> are never optional (except as part of an optional phrase). This is because the empty specification for <sources> and <destinations> is legal and has a default value.

Any of the non-declarative commands can be interrupted at any time by typing any character. The command completes processing of the current chain and then enters a "break". In this state, the user can execute any PSTP command or LISP function and continue or abort the processing of the interrupted command.

ANCESTRY command. The (ANCESTRY [DELETE] <sources>) command prints the derivation of each chain designated by <sources>. If DELETE is specified, each designated chain is also deleted from its chain list.

CHAINLIST command. The (CHAINLIST [<declarations>]) command is used to declare chain lists that will be used and their format and order functions. If <declarations> is absent, the CHAINLIST command prints the list of previously declared chain lists and their format and order functions.

COPY command. The (COPY [DELETE] <sources> [TO <destinations>]) command copies each chain designated by <sources> to each of <destinations>. If DELETE is specified, each designated chain is also deleted from its chain list.

COUNT command. The (COUNT [DELETE] <sources>) command counts the number of chains designated by <sources>. If DELETE is specified, each designated chain is also deleted from its chain list.

DELETE command. The (DELETE <sources>) is the same as the (COUNT DELETE <sources>) command, i.e., the COUNT command with chain deletion specified.

DISPLAY command. The (DISPLAY [DELETE] <sources>) command prints each chain designated by <sources>. If DELETE is specified, each designated chain is also deleted from its chain list.

EXPAND command. The (EXPAND [EXTEND] [FACTOR] [REDUCE] [PEXTEND] [PREDUCE] [DELETE] <sources1> [BY <sources2>] [GIVING <destinations1>] [AND <destinations2>]) is the principal inference command for interactive use. It will perform the designated inference operations on each of the chains designated by <sources1> using each chain designated by <sources2> as second argument to binary inference operations (extension and p-extension). Derived chains will be stored in <destinations1> and lemmas will be stored in <destinations2> (chain filters permitting). If DELETE is specified, each designated chain in <sources1> is also deleted from its chain list. The EXPAND command is restricted to performing inference operations on chains existing at the time of its invocation, i.e., it will not perform any inference operations on chains it has just derived. The command terminates when (1) the empty chain is generated, i.e., a proof has been found, (2) all the specified operations have been performed, or (3) the user suspends processing of the command by typing any character. If no inference operations are designated, all inference operations will be used. If at least one inference operation is designated, the word EXPAND may be omitted.

FOR command. The (FOR [DELETE] <sources> DO <function>) command applies the unary LISP function <function> to each chain designated by

<sources>. If DELETE is specified, each designated chain is also deleted from its chain list.

PARAMETERS command. The (PARAMETERS [<index>]) command is used to declare the values of several global parameters. If an <index> (an arbitrary LISP atom) is specified, it designates for use the predefined set of parameter values associated with <index>. If an <index> is not specified or <index> has no previously defined meaning, the PARAMETERS command asks a series of questions requiring the user to define the value for each parameter. Parameters set by the PARAMETERS command include: whether newly generated chains are to be printed, the format in which chains are to be printed, whether lemmas are to be generated, and whether subsumption is to be performed.

POSTPROCESSING command. The (POSTPROCESSING [<index>]) command is used to declare what postprocessing operation is to be employed. If an <index> (an arbitrary LISP atom) is specified, this <index> designates the postprocessing operation that will be used. Allowed <index>s include WEAK-SAVE, WEAK-DELETE, STRONG-SAVE, and STRONG-DELETE, designating the postprocessing operations described in Section 1. If an <index> is not specified or <index> has no previously defined meaning, the POSTPROCESSING command asks a series of questions requiring the user to designate which action among a list of alternative actions is to be taken for a given condition. For example, the POSTPROCESSING command may ask whether, in the case of an A-literal followed by an identical B-literal, the B-literal should be saved, deleted, or deleted with the reduction operation recorded in the ancestry of the chain.

PROBLEM command. The (PROBLEM [<declarations>]) command sets up a problem for the theorem prover. It first makes the chainlist declarations specified by <declarations> (if fewer than two chain list declarations are specified, up to two default declarations will be made) and then asks the user to type in the theorem and each axiom. The theorem is stored in the first declared chain list; its negation and the axioms are stored in the second declared chain list. The input format for the theorem and axioms is the same as was used in the description of the linear MESON procedure except that prefix form for predicate and function symbols is required, and (due to character set limitations) $\wedge$ and $\vee$ are omitted and --> is substituted for $\rightarrow$. Thus, $Pab \wedge Pba \rightarrow a=b \vee Qabx$ is typed in as Pab Pba --> =ab Qabx. The theorem and axioms are then encoded into internal list form. The PROBLEM command permits the user to save the encoded axioms so that they will not need to be retyped in future proofs of the same problem.

<u>SEARCH</u> <u>command</u>. The (SEARCH [EXTEND] [FACTOR] [REDUCE] [PEXTEND] [PREDUCE] <sources1> [BY <sources2>] [GIVING <destinations1>] [AND <destinations2>]) repeatedly deletes the first chain from <sources1> (a chain with lowest order function value) and performs on it each designated inference operation with the chains designated by <sources2> as second argument to binary inference operations (extension and p-extension). Derived chains will be stored in <destinations1> and lemmas will be stored in <destinations2> (chain filters permitting). So that newly generated chains can be used as input to inference operations by the SEARCH command, <sources1> and <destinations1> will ordinarily specify the same chain lists. The command terminates when (1) the empty chain is generated, i.e., a proof has been found, (2) <sources1> is empty meaning no more operations can be performed and no proof could be found within the constraints of the

specified operations, initial chains, and chain filters, or (3) the user suspends processing of the command by typing any character. If no inference operations are designated, all inference operations will be used.

<u>SEARCH2</u> <u>command</u>. The (SEARCH2 [EXTEND] [FACTOR] [REDUCE] [PEXTEND] [PREDUCE] <sources1> [BY <sources2>] [GIVING <destinations1>] [AND <destinations2>]) repeatedly deletes the first chain from <sources1> (a chain with lowest order function value) and performs on it the first designated inference operation in the order function value. This inference operation is then deleted from the order function value and, if any inference operations remain in the order function value, the chain is reinserted in <sources1> (now with the numerical value associated with the next inference operation as the numerical value of the chain for insertion into the sorted chain list). Derived chains will be stored in <destinations1> and lemmas will be stored in <destinations2> (chain filters permitting). So that newly generated chains can be used as input to inference operations by the SEARCH2 command, <sources1> and <destinations1> will ordinarily specify the same chain lists. The order function, using variables of the SEARCH2 function, will construct a list of inference operations with (in the case of binary inference operations) second arguments as specified in <sources2> for derived chains as they are stored. The command terminates when (1) the empty chain is generated, i.e., a proof has been found, (2) <sources1> is empty meaning no more operations can be performed and no proof could be found within the constraints of the specified operations, initial chains, and chain filters, or (3) the user suspends processing of the command by typing any character. If no inference operations are designated, all inference operations will be used.

TRANSFER _command_. The (TRANSFER <sources> [TO <destinations>]) command is the same as the (COPY DELETE <sources> [TO <destinations>]) command, i.e., the COPY command with chain deletion specified.

3. Performance Study

In order to give some idea of the performance of PSTP with some simple search strategies and to make some points about relative merits of some of these strategies, the results of PSTP runs on 9 examples using 4 strategies are presented here. Results are compared to results for two other theorem proving programs tested on the same examples.

The examples. The examples are taken from a comparative study of theorem proving strategies used by QA3.6 by Reboh et al [11] (additional information on sources, theory, and previous uses of these examples are in [11]); the same examples were also run for an SL-resolution theorem prover (here called SLRTP) by Aubin [1,2]. The examples are axiomatized just as for QA3.6 with an occasional substitution of a disjunction for an implication and, in the cases of unsatisfiable sets of axioms, the use of the negation of one of the axioms as the theorem.

Inference operations used. All the examples were run with extension as the only rule of inference except the NUM1 example for which reduction was also necessary. The WEAK-DELETE postprocessing operation was used for all the examples. Its use, of course, permits ground factorization and reduction. In some examples (BURSTALL, SHORTBURST, GROUP1, GROUP2), it is readily apparent from the structure of the problem that no reduction is possible (since every chain derived from the theorem has only positive literals eliminating any possibility of matching an A-literal with a complementary B-literal). The ANCES1 example is propositional and thus the ground reduction in the WEAK-DELETE postprocessing operation is sufficient. In the remaining three problems (HAS-PARTS1, HAS-PARTS2, PRIM) for which reduction was not employed (although ground reduction was used in each), the use of the reduction operation resulted in the generation of no additional chains. Lemmas were not generated for any of the examples.

Search strategies used. The strategies used are characterized by 4 parameters: length multiplier, level multiplier, length maximum, and level maximum. The length of a chain is defined to be its number of B-literals. This is consistent with the notion of the length of a clause in resolution theorem proving being its number of literals since in a chain A-literals record ancestry information and would not be present in the corresponding clause form. The level of a chain is defined to be the number of inference operations employed in deriving it from the alleged theorem excluding those operations (ground factorization and reduction) automatically performed by the postprocessing operation.

The SEARCH2 search command was employed with projected inference operations ordered according to the minimum values of a weighted sum of the expected length and level of the result. The expected length of a chain derived by extension is the length of its parent chain being extended plus the length of the axiom minus 2. The expected length of a chain derived by factorization or reduction is the length of its parent chain minus 1. The actual length may be less (but never more) due to removal of B-literals by the accepting transformation. The expected and actual level of a chain is the level of its parent plus 1. Only inference operations whose results have expected lengths and levels not exceeding the length or level maxima will be attempted (this way of implementing length and level maxima was also used by QA3.6 and SLRTP).

Two sets of length and level multipliers were tried. The first has a length multiplier of 101 and a level multiplier of 100 and is called the 101/100 strategy. In the 101/100 strategy, the projected inference operation with highest merit is one with the smallest value of (100 times) the sum of expected length and level of the result. Ties are resolved in favor of lesser expected length (a 100/101 strategy would resolve ties in favor of lesser expected level). (It is assumed here that the expected length of a chain will never exceed 100.) The most important thing to note about the 101/100 strategy is that it is essentially the same as Kowalski and Kuehner's upper diagonal search strategy [5]. It is an admissable strategy [10] except for cases where the postprocessing operation removes B-literals by ground factorization or reduction. First proofs discovered by admissable strategies are guaranteed to be minimum level proofs.

The second strategy has a length multiplier of 501 and a level multiplier of 100 and is called the 501/100 strategy. In the 501/100 strategy, the projected inference operation with highest merit is one with the smallest value of (100 times) the sum of the expected level and 5 times the expected length of the result. Ties are again resolved in favor of lesser expected length. By multiplying length by 5 times as much as level, a strong length preference strategy is produced. The 501/100 strategy is, of course, inadmissable since it is clearly not always the case that it requires at least 5 inference operations to remove a single literal. (For a strategy to be admissable, the estimated additional cost to solution must always be less than or equal to the actual additional cost to solution.)

The 101/100 and 501/100 strategies were each tried with (bounded) and without (unbounded) length and level maxima. The length and level maxima used were those used by QA3.6 wherever possible.

Statistics. The performance of strategies will be primarily characterized by the "chains generated" statistic. Here, this information is represented by a 4-tuple: the first component is the number of chains retained; the second component is the number of acceptable chains generated; the third component is the total number of chains generated; the fourth component is the number of attempted inference operations. The number of retained chains is the number of acceptable chains minus the number of chains eliminated by subsumption, function depth tests, etc. No such processes were used to eliminate chains in this experiment, so the number of retained chains is always equal to the number of acceptable chains. The total number of chains generated is the number of acceptable chains plus the number of non-acceptable chains generated. These statistics and the time figures referred to below are automatically accumulated by PSTP and printed out when a proof is found.

Nearly comparable statistics are presented where available for QA3.6 and SLRTP (except QA3.6 statistics refer to clauses rather than chains).

Best and mean performance figures are presented for QA3.6 on each example. For QA3.6, the number of retained clauses is the number of retained clauses after subsumption and function depth tests; the number of acceptable clauses is computed as the number of successful resolutions and factorings; the number of attempted inference operations is computed as the number of attempted resolutions and factorings. The proportion of tested QA3.6 strategies which discovered a proof is given on the same line as the mean performance of QA3.6 strategies; unsuccessful strategies were excluded in computing the means.

Performance figures are presented for SLRTP where the set of support for the refutation was the negation of the theorem in the PSTP proof on each example. Due to the similarity of operations and terminology between the linear MESON procedure and the inference system for SLRTP, SL-resolution [5], we will here present a brief description of SL-resolution.

SL-resolution, a refutation procedure, can be viewed as a variant of model elimination without equality with the following features. (1) The capability for reordering B-literals at the end of a chain is formalized in the form of a literal selection function which designates the literal to be extended on in succeeding extension operations. (2) Factorization is a required operation for completeness in SL-resolution since the equivalent of the STRONG-SAVE postprocessing operation is employed. The model elimination factorization and reduction operations are combined into the SL-resolution reduction operation. (3) SL-resolution requires a fully factored input set of clauses, i.e., every non-tautologous factor of an input axiom must also be input (or, as in SLRTP, derived). A benefit of this is that SL-resolution reduction operations need never be performed with the leftmost involved literal being or following the last A-literal of the chain. (4) Any B-literal following the last A-literal of the chain is a candidate for removal by the reduction operation, not just the rightmost as in model elimination. (5) Upper diagonal search is the prescribed search strategy for SL-resolution.

For SLRTP, the number of retained chains is the number of retained chains after function depth tests and subsumption (subsumption is only used in eliminating redundant axiom chains or their factors during the process of generating a fully factored input set of axioms); the number of acceptable chains is computed as the number of successful extensions, reductions, and factorings (used only for generating a fully factored input set of axioms); the number of attempted inference operations is computed as the number of attempted extensions, reductions, and factorings. The GROUP1 and GROUP2 example statistics are taken from [2].

Time statistics. The "search time" statistic represents the time spent in searching for a proof by a compiled version of PSTP; it excludes time spent in inputting the problem, outputting of final statistics and proof, and garbage collection, although it does include time required for some trace output during the search. Search time is the only widely variable component of total time to solution with problem input and statistics and proof output time relatively constant and small. Although PSTP is conservative of storage (performing LISP CONS operations only when necessary when instantiating chains) and therefore ordinarily requires few garbage collections, garbage collection time is excluded because (1) time consuming garbage collections occurring at random times in the search for a proof tend to randomize the time statistics especially for short searches (this problem could be overcome by always starting a search for a proof immediately after a garbage collection) and (2) frequency of garbage collection is dependent on the amount of storage available (with infinite storage, there need not be any garbage collections). Nearly all the proofs presented here were found with about 25000 words available for storing chains and most were found with no garbage collections.

Time statistics should not be used for comparison among strategies used by different theorem provers without considerable caution and more information than is usually available. Such statistics are of course influenced by the machine and operating system used, language and coding of the theorem prover, whether compiled (SLRTP, PSTP) or interpreted (QA3.6), special conditions applying to the operation of the theorem prover (e.g., tracing), and some randomness in the times themselves (such randomness, attributable to variable load on the time sharing system, is visible in some anomalies in the statistics presented here).

<u>Results</u>. Four primary observations can be made from the results of the experiment presented here: (1) PSTP performs competitively with QA3.6 and SLRTP, (2) the 501/100 strategy performs better than the 101/100 strategy (for these examples), (3) the 501/100 strategy is relatively insensitive to length and level bounds, the 101/100 strategy is much more sensitive, and (4) elimination of some implicative forms of the axioms can result in improved performance.

The basis for comparison of the results of PSTP and QA3.6 is the number of acceptable chains generated (equals the number of chains retained) for PSTP versus the number of acceptable clauses generated (equals the number of successful resolutions and factorings) for QA3.6. This is a fairer comparison than one using the number of retained clauses for QA3.6 since QA3.6 eliminated clauses by function depth maxima and subsumption. Even this comparison is still somewhat unfair to PSTP since if function depth tests and subsumption had not been used in QA3.6, the number of generated clauses would presumably have been larger since eliminated clauses could now act as parent clauses in additional inferences.

Using this basis for comparison, the unbounded 501/100 strategy (the strategy we prefer for reasons given below) performed better than the average of QA3.6 strategies which found a proof in all the examples except PRIM, GROUP1, and GROUP2.

In the PRIM example, the unbounded 501/100 strategy performed only slightly worse than the average of QA3.6 strategies.

In the GROUP1 example, the absolute difference in performance is small even if the number of chains generated by PSTP is double the number of clauses generated by QA3.6. In view of the fact, for example, that by reversing the order of presentation of the axioms to PSTP can cause the performance of PSTP to exceed that of QA3.6, we tend to regard this difference as being relatively insignificant.

The difference in the case of GROUP2 is much more serious and has a rather different explanation. Where the formulation of the GROUP2 example has several unit axioms and two 4-literal associative axioms, the use of a length maximum value of 3 can be seen to be extremely restrictive. In resolution terms, this length maximum requires that only units be resolved against the associative axioms, and in the case of GROUP2 if the negation of the theorem is used as the set of support, the length maximum automatically restricts any tested strategy to a further refinement of unit resolution in which only the negation of the theorem can be directly resolved against the associative axioms. PSTP was tested with a variant of the GROUP2 example in which axiom 3 was reordered so that a proof meeting the length maximum value of 3 restriction existed. On this example, the unbounded 101/100 strategy generated 435 chains; all the other strategies generated 29. We therefore feel that the better performance of QA3.6 on this problem was more attributable to the restrictive length maximum than to an intrinsic inferiority of PSTP.

The comments about the restrictive length bound used by QA3.6 in the GROUP2 example can be extended to several other examples. The BURSTALL, SHORTBURST, HAS-PARTS1, HAS-PARTS2, PRIM, ANCES1, GROUP1, and GROUP2 examples all had very restrictive length maxima, in every case set at or below the minimum value required for PSTP to discover a proof. Level maxima were often similarly restrictive although we perceive this to be much less important in reducing the size of the search in the non-depth first search strategies tested. We feel that use of such restrictive length and level (especially length) bounds invalidates the results of [11] to a degree, since their use imposes severe limitations on the structure of the search space. In this restricted search space, tests of different strategies may fail to discriminate between strategies, or unfairly discriminate between them.

Given the similarity of SL-resolution and the linear MESON procedure, one would anticipate substantial similarity in the results for PSTP using the bounded 101/100 strategy (upper diagonal search) and SLRTP. For 6 of the examples (BURSTALL, SHORTBURST, HAS-PARTS1, PRIM, ANCES1, and NUM1), the results agree closely. Differences emerge for the remaining 3 examples. We don't know why PSTP did so much worse than SLRTP on the HAS-PARTS2 example. In the GROUP1 example, use of a fully factored input set of clauses was clearly beneficial to SLRTP since the very short proof could be shortened further by using a factored form of one of the associative axioms. PSTP with factorization could not match the SLRTP results, since extension by the associative axiom followed by factorization counted as 2 inference operations in computing the level of the resulting chain (whose value is used to compute the order function value) whereas extension by the factored associative axiom by SLRTP counts as only 1 inference operation. (PSTP could be made to equal SLRTP's performance on this example by inputting a fully factored set of axioms, a perfectly legal operation, although unnecessary for completeness.) In the case of the GROUP2 example, for which SLRTP failed to find a solution, both the bounded and unbounded 101/100 strategies in PSTP showed relative difficulty in discovering a solution. In SLRTP, this difficulty was exacerbated by the very feature which aided the quick solution of the GROUP1 example: mandatory factorization. PSTP with factorization and the STRONG-DELETE postprocessing operation (resulting in an inference system very similar to SL-resolution) failed to find a proof with the unbounded 101/100 strategy after 1507 chains were generated, discovered a proof while generating 410 chains with the unbounded 501/100 strategy, and discovered a proof while generating 458 chains with each of the bounded strategies. The proofs discovered were the same as those discovered without factorization.

We believe the detrimental effects of factorization as demonstrated in the GROUP2 example results are more typical than the beneficial effects illustrated in the GROUP1 example. In our experience, even in cases where factorization does shorten a proof (as it did not in the GROUP2 example), the proliferation of highly instantiated chains caused by the use of factorization still often outweighs the benefits. (These negative comments clearly refer only to general factorization where literals must be unified; factorization in the ground case is clearly beneficial and is included in the postprocessing operations we used here.) Should future experience prove this judgment about factorization wrong, the linear MESON procedure still permits factorization as a legal though optional operation.

Another point can be made here concerning SLRTP's efforts to discover a solution to the GROUP2 example. SLRTP uses a literal selection function to designate which literal of each derived chain is to be used in future extension operations. The only literal selection function tested was the function which always selects a literal which has the fewest matching literals among the axioms. This has the obviously desirable characteristic of reducing the branching rate of the search tree since the selected literal has the fewest matches among the axioms and, further, removal of the selected literal after some inference operations will usually instantiate the remaining literals and reduce the number of literals among the axioms matching them. However, this literal selection function is, in the case of problems with structure similar to the GROUP2 example, inconsistent with the use of length maxima. In GROUP2, for example, the literal selection function will show a preference for literals capable of being extended upon only by the associative axioms (since any positive literal matches the consequent of the associative axioms, any literal matching a unit axiom also matches the associative axioms). Thus, the effect of the use of this literal selection function is to increase the length of chains appearing in a deduction possibly requiring the increase of the length maximum used.

One final point remains about the comparison of results between PSTP and SLRTP. This concerns the very small number of attempted inference operations by SLRTP. This is due to the use of a literal classification tree which automatically selects out likely matches for literals to be extended upon from among the literals in the axioms. The extension operation is only attempted for axioms containing literals selected by the literal classification tree. This probably represents a fairly small (though real) saving in computational effort since one must count the cost of creating and accessing the literal classification tree and the cost saved is that of attempting unifications destined to fail, usually a fairly quick operation. The real benefit of use of the literal classification tree is the elimination of the multiple attempts at unifying literals that would ordinarily result from use of the literal selection function requiring discovery of the number of matches for a literal among literals in the axioms.

In comparing the four strategies tested by PSTP among themselves, one first discovers that the 501/100 strategy invariably performed as well as or better than the 101/100 strategy for the same choice of length and level maxima. This is especially true of the results for the BURSTALL, GROUP2, and PRIM examples in the absence of length and level maxima. A further demonstration of the superiority of the 501/100 strategy is its relative insensitivity to length and level bounds. Only in the BURSTALL example did the bounded 501/100 strategy perform significantly better than the unbounded strategy. Also, in the PRIM and GROUP2 examples, the addition of length and level bounds actually degraded the performance of the 501/100 strategy since the bounds excluded proofs discovered by the unbounded strategy. In contrast, performance of the 101/100 strategy was often improved by the addition of length and level bounds, but (as stated above) never improving upon the performance of the 501/100 strategy. The demonstrated insensitivity of the 501/100 strategy to the addition of length and level bounds seems especially significant in view of the often extreme restrictiveness of the bounds tested.

Due to its generally good performance and lack of improvement with the addition of bounds, we regard the unbounded 501/100 strategy as the best among those tested.

We feel generally that, provided it performs adequately, a complete (e.g., length preference) strategy like the unbounded 501/100 strategy is to be preferred to an incomplete (e.g., length bounded) strategy like the bounded 101/100 strategy, even if the latter, with appropriate choice of bounds, can often match the performance of the former.

Finally, we merely note that judicious elimination of various implicative forms of the axioms can result in significantly improved performance as demonstrated in the results for the HAS-PARTS1, HAS-PARTS2, and PRIM examples. Of course, this elimination of implicative forms of the axioms destroys the completeness property of the linear MESON procedure. However, this controlled incompleteness may be desirable in cases where significant improvement in performance results. Completeness could be preserved and nearly the same effect gained by presenting PSTP with all the implicative forms of the axioms, but (via the order function definition) giving PSTP a strong preference for using one instead of another.

One feature of the linear MESON procedure not previously discussed is the length of its proofs. It is characteristic of linear theorem proving strategies that they require longer proofs than some other strategies. This and past studies [1,4] indicate that linear strategies can overcome this increased proof length and perform competitively with other procedures. The linear resolution strategy tested in QA3.6 was less successful since the special chain rejection criteria of variants of the model elimination procedure were not used.

While the length of a proof is one measure of its complexity, we feel that the increased length of linear MESON procedure proofs is not a great disadvantage in terms of readability. The problem-reduction method oriented form of such proofs often makes them more comprehensible than ordinary resolution proofs relying on converging lines of deduction resulting in a refutation.

4. Summary

We have presented the linear MESON procedure, a procedure we feel to be one of the most natural available systems of complete inference in the first order predicate calculus due to its linear format and relationship to goal-subgoal trees. It also has advantages in the capability for inputting multiple implicative variants of individual axioms so that individual variants can be differentially treated (or ignored). Another advantage of the linear MESON procedure as compared with, for example, SL-resolution is the optional nature of the factorization operation. Although the point is not elaborated upon here, it is our observation (also made by Fleisig et al [4]) that factorization (except ground factorization) is usually harmful and results in instantiating chains too greatly.

The Programmable Strategy Theorem Prover (PSTP) is a theorem proving program using the linear MESON procedure as its inference system. Especially significant features of PSTP are the general capabilities for specifying information to be computed or retained about chains, for specifying which chains are to be retained or manipulated by a given command, and for specifying the order in which inference operations are to be performed in fully automatic searches for a proof.

We have presented the results for PSTP solutions of 9 examples previously tested in two other theorem proving studies. From these results we concluded that PSTP performed competitively as compared with the other tested theorem provers. We feel that the potential significance of PSTP is not that it perform spectacularly using simple search strategies such as those tested (it doesn't), but that it provides an inference system and system features which facilitate the user specification of more complex and effective search strategies and chain elimination criteria.

We also demonstrated empirically the inferiority of the diagonal search strategy to similar strategies which have a stronger length preference component and that such length preference oriented strategies all but eliminate the need for length maxima for problems of this level of complexity. We prefer search strategies that have, for example, length preference built in to the added imposition of length bounds: the use of preference strategies in the absence of bounds results in complete strategies guaranteed to find a solution if one exists.

We have also criticized some of the methodology used in previous theorem proving studies. Results in such studies are often heavily dependent on the length and level (especially length) bounds used in restricting the search for a solution. In consequence, success in finding a solution in reasonable time and space is often more attributable to the bounds used than the tested strategy. Thus, such results fail to adequately discriminate between different theorem proving procedures. We urge that future studies test theorem proving procedures in the absence of artificially imposed bounds.

Acknowledgments

I wish to thank Prof. Donald W. Loveland (Department of Computer Science, Duke University) for his guidance during the development of PSTP and for his continuing development of the model elimination and MESON procedures. I also wish to thank Profs. Jack Buchanan, Allen Newell (both at Department of Computer Science, CMU) and Peter B. Andrews (Department of Mathematics, CMU), and René Reboh (Datalogilaboratoriet, Uppsala University) who, in addition to Prof. Loveland, all reviewed an earlier version of this paper.

Bibliography

1. Aubin, R. Some experimental results on SL-resolution. Memo No 66, Department of Computational Logic, School of Artificial Intelligence, University of Edinburgh, Edinburgh, July 1973.

2. Aubin, R. Personal communication. Dec. 1973.

3. Bobrow, R. J., Burton, R. R., Jacobs, J. M. and Lewis, D. ***UCI LISP Manual.*** University of California, Irvine, Calif., 1973.

4. Fleisig, S., Loveland, D., Smiley, A. K. III and Yarmush, D. L. An implementation of the model elimination proof procedure. ***J. ACM 21***, 1 (Jan. 1974), 124-139.

5. Kowalski, R. and Kuehner, D. Linear resolution with selection function. ***Artificial Intelligence 2*** (1971), 227-260.

6. Loveland, D. W. A simplified format for the model elimination theorem-proving procedure. ***J. ACM 16***, 3 (July 1969), 349-363.

7. Loveland, D. W. A unifying view of some linear Herbrand procedures. ***J. ACM 19***, 2 (Apr. 1972), 366-384.

8. Loveland, D. W. Forthcoming book. North-Holland, Amsterdam.

9. Loveland, D. W. and Stickel, M. E. The hole in goal trees: some guidance from resolution theory. ***Advance Papers 3rd Int. Joint Conf. on Artificial Intelligence***, Stanford, Calif., 1973, pp. 153-161.

10. Nilsson, N. J. ***Problem-Solving Methods in Artificial Intelligence***. McGraw-Hill, New York, 1971.

11. Reboh, R., Raphael, B., Yates, R. A., Kling, R. E. and Velarde, C. Study of automatic theorem-proving programs. Technical Note 75, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, Calif., Nov. 1972.

1. BURSTALL Example

Axioms:

```
 1. has(p1,ass(j,n0))
 2. follows(p2,p1)
 3. has(p2,ass(k,n1))
 4. labels(loop,p3)
 5. follows(p3,p2)
 6. has(p3,ifthen(equal(j,n),p4))
 7. has(p4,goto(out))
 8. follows(p5,p4)
 9. follows(p6,p3)
10. has(p6,ass(k,times(n2,k)))
11. follows(p7,p6)
12. has(p7,ass(j,plus(j,n1)))
13. follows(p8,p7)
14. has(p8,goto(loop))
15. follows(xp,yp) → succeeds(xp,yp)
16. succeeds(xp,zp) ∧ succeeds(zp,yp) → succeeds(xp,yp)
17. has(xp,goto(zp)) ∧ labels(zp,yp) → succeeds(yp,xp)
18. has(xp,ifthen(ze,yp)) → succeeds(yp,xp)
```

Theorem:

```
19. succeeds(p3,p3)
```

2. SHORTBURST Example

Axioms:

```
 1. labels(loop,p3)
 2. has(p3,ifthen(equal(j,n),p4))
 3. has(p4,goto(out))
 4. follows(p5,p4)
 5. follows(p8,p3)
 6. has(p8,goto(loop))
 7. follows(xp,yp) → succeeds(xp,yp)
 8. succeeds(xp,zp) ∧ succeeds(zp,yp) → succeeds(xp,yp)
 9. has(xp,goto(zp)) ∧ labels(zp,yp) → succeeds(yp,xp)
10. has(xp,ifthen(ze,yp)) → succeeds(yp,xp)
```

Theorem:

```
11. succeeds(p3,p3)
```

3. HAS-PARTS Example 1

Axioms:

```
1. in(John,boy)
2. in(x,boy) → in(x,human)
3. hp(x,xm,y) → in(sk1(x,y,z,xm,xn),y) v hp(x,t(xm,xn),z)
4. hp(x,xm,y) → ¬hp(sk1(x,y,z,xm,xn),xn,z) v hp(x,t(xm,xn),z)
5. in(x,hand) → hp(x,n5,fingers)
6. in(x,human) → hp(x,n2,arm)
7. in(x,arm) → hp(x,n1,hand)
```

Theorem:

```
8. hp(John,t(n2,n1),hand)
```

4. HAS-PARTS Example 2

Axioms:

1. in(John,boy)
2. in(x,boy) → in(x,human)
3. hp(x,xm,y) → in(sk1(x,y,z,xm,xn),y) v hp(x,t(xm,xn),z)
4. hp(x,xm,y) → ¬hp(sk1(x,y,z,xm,xn),xn,z) v hp(x,t(xm,xn),z)
5. in(x,hand) → hp(x,n5,fingers)
6. in(x,human) → hp(x,n2,arm)
7. in(x,arm) → hp(x,n1,hand)

Theorem:

8. hp(John,t(t(n2,n1),n5),fingers)

5. PRIM Example

Axioms:

1. Dxx
2. Dxy ∧ Dyz → Dxz
3. Px v Dg(x)x
4. Px v Ln1g(x)
5. Px v Lg(x)x
6. Ln1x ∧ Lxa → Pf(x)
7. Ln1x ∧ Lxa → Df(x)x
8. Ln1a

Theorem:

(9. ¬Px v ¬Dxa   negation of theorem)
10. Px1 ∧ Dx1a

6. ANCES1 Example

Axioms:

1. ¬J v A v H
2. K v H v J
3. ¬K v H v J
4. ¬A v ¬B
5. ¬A v B
6. ¬H v ¬C

Theorem:

7. H ∧ ¬C

7. NUM1 Example

Axioms:

1. Px ∧ Myzw ∧ Dxw → Dxy v Dxz
2. Mxxs(x)
3. Mxyz → Myxz
4. Mxyz → Dxz
5. Mas(c)s(b)
6. Pa

Theorem:

7. Dab

8. GROUP1 Example

Axioms:

1. $Pxyu \wedge Pyzv \wedge Pxvw \rightarrow Puzw$
2. $Pxyu \wedge Pyzv \wedge Puzw \rightarrow Pxvw$
3. $Pg(xy)xy$
4. $Pxh(xy)y$
5. $Pxyf(xy)$

Theorem:

(6. $\neg Pj(x)xj(x)$ negation of theorem)
7. $Pj(x1)x1j(x1)$

9. GROUP2 Example

Axioms:

1. Pxex
2. Pexx
3. $Pxyu \wedge Pyzv \wedge Puzw \rightarrow Pxvw$
4. $Pxyu \wedge Pyzv \wedge Pxvw \rightarrow Puzw$
5. Pxxe
6. Pabc

Theorem:

7. Pbac

					Statistics	
Length Multi-plier	Level Multi-plier	Length Maximum	Level Maximum	Proof Code	Chains Generated (ret/acc/tot/att)	Search Time (sec)
1. BURSTALL Example						
101	100	-	-	A	191/191/215/3129	46.3
501	100	-	-	B	74/ 74/ 75/1229	16.1
101	100	2	12	A	45/ 45/ 45/741	9.8
501	100	2	12	A	45/ 45/ 45/741	9.2
QA3.6 best		2	12		38/ 42/ /1462	
QA3.6 mean 16/19		2	12		99/118/ /3222	
SLRTP		3	13		48/ 48/ /122	
2. SHORTBURST Example						
101	100	-	-	A	18/ 18/ 19/144	3.0
501	100	-	-	A	16/ 16/ 16/128	2.1
101	100	2	10	A	16/ 16/ 16/128	2.2
501	100	2	10	A	16/ 16/ 16/128	2.1
QA3.6 best		2	10		12/ 12/ /255	
QA3.6 mean 14/14		2	10		20/ 21/ /325	
SLRTP		3	10		16/ 16/ / 42	
3. HAS-PARTS Example 1						
Implicative form for axioms:						
101	100	-	-	A	7/ 7/ 7/ 47	0.7
501	100	-	-	A	7/ 7/ 7/ 47	0.7
101	100	3	10	A	7/ 7/ 7/ 47	1.0
501	100	3	10	A	7/ 7/ 7/ 47	0.8
Disjunctive form for axioms:						
101	100	-	-	A	12/ 12/ 12/124	1.8
501	100	-	-	A	12/ 12/ 12/124	1.7
101	100	3	10	A	12/ 12/ 12/124	1.5
501	100	3	10	A	12/ 12/ 12/124	1.5
QA3.6 best		2	10		8/ 10/ /112	
QA3.6 mean 6/6		2	10		20/ 24/ /343	
SLRTP		2	10		12/ 12/ / 29	
4. HAS-PARTS Example 2						
Implicative form for axioms:						
101	100	-	-	A	11/ 11/ 11/ 83	1.3
501	100	-	-	A	11/ 11/ 11/ 83	1.3
101	100	3	10	A	11/ 11/ 11/ 83	1.3
501	100	3	10	A	11/ 11/ 11/ 83	1.9
Disjunctive form for axioms:						
101	100	-	-	A	50/ 50/ 50/478	8.1
501	100	-	-	B	38/ 38/ 38/430	5.7
101	100	3	10	A	38/ 38/ 38/430	5.7
501	100	3	10	B	38/ 38/ 38/430	6.4
QA3.6 best		2	10		12/ 14/ /205	
QA3.6 mean 6/7		2	10		44/ 51/ /938	
SLRTP		3	13		20/ 20/ / 41	

Length Multiplier	Level Multiplier	Length Maximum	Level Maximum	Proof Code	Statistics: Chains Generated (ret/acc/tot/att)	Search Time (sec)
5. PRIM Example						
Implicative form for axioms:						
101	100	-	-	A	812/812/1052/8072	200.7
501	100	-	-	A	57/ 57/ 64/640	11.6
101	100	3	18	A	70/ 70/ 82/816	12.3
501	100	3	18	A	54/ 54/ 60/609	8.2
Disjunctive form for axioms:						
101	100	-	-	B	165/165/187/1883	28.5
501	100	-	-	C	101/101/113/1220	18.8
101	100	3	10	B	130/130/146/1532	20.2
501	100	3	10	B	130/130/146/1532	18.8
101	100	3	18	B	130/130/146/1532	21.7
501	100	3	18	C	101/101/113/1220	16.8
QA3.6 best		3	10		13/ 19/ /208	
QA3.6 mean 9/10		3	10		36/ 97/ /999	
SLRTP		3	11		122/134/ /243	
6. ANCES1 Example						
101	100	-	-	A	23/ 23/ 23/240	3.2
501	100	-	-	A	13/ 13/ 13/108	1.3
101	100	2	10	A	13/ 13/ 13/108	1.6
501	100	2	10	A	13/ 13/ 13/108	1.4
QA3.6 best		2	10		5/ 12/ /158	
QA3.6 mean 19/20		2	10		6/ 13/ /129	
SLRTP		3	10		14/ 14/ / 26	
7. NUM1 Example						
101	100	-	-	A	10/ 10/ 11/ 47	1.2
501	100	-	-	A	10/ 10/ 11/ 47	1.1
101	100	5	10	A	10/ 10/ 11/ 47	0.9
501	100	5	10	A	10/ 10/ 11/ 47	0.9
QA3.6 best		5	10		8/ 10/ / 68	
QA3.6 mean 11/11		5	10		9/ 11/ / 83	
SLRTP		5	10		9/ 9/ / 21	
8. GROUP1 Example						
101	100	-	-	A	14/ 14/ 14/ 54	2.0
501	100	-	-	A	14/ 14/ 14/ 54	1.5
101	100	3	10	A	14/ 14/ 14/ 54	1.5
501	100	3	10	A	14/ 14/ 14/ 54	1.4
QA3.6 best		3	10		7/ 7/ / 33	
QA3.6 mean 9/9		3	10		7/ 7/ / 34	
SLRTP		3	10		9/ 12/ / 35	

	Length Multiplier	Level Multiplier	Length Maximum	Level Maximum	Proof Code	Statistics: Chains Generated (ret/acc/tot/att)	Search Time (sec)
9. GROUP2 Example							
	101	100	-	-	A	576/576/752/2408	97.2
	501	100	-	-	B	119/119/149/500	17.0
	101	100	4	10	A	225/225/325/938	28.8
	501	100	4	10	A	225/225/325/938	36.7
	QA3.6 best		3	10		54/ 74/ /324	
	QA3.6 mean 8/8		3	10		60/ 82/ /517	
	SLRTP		?	?		no proof found	

Proofs

1. BURSTALL Example

Proof A

```
19. succeeds(p3,p3)                                                     theorem
20. [succeeds(p3,p3)] succeeds(p3,x1) succeeds(x1,p3)                  extend by 16
21. [succeeds(p3,p3)] succeeds(p3,x1) [succeeds(x1,p3)] follows(x1,p3) extend by 15
22. [succeeds(p3,p3)] succeeds(p3,p6)                                  extend by 9
23. [succeeds(p3,p3)] [succeeds(p3,p6)] succeeds(p3,x1) succeeds(x1,p6) extend by 16
24. [succeeds(p3,p3)] [succeeds(p3,p6)] succeeds(p3,x1) [succeeds(x1,p6)]
    follows(x1,p6)                                                     extend by 15
25. [succeeds(p3,p3)] [succeeds(p3,p6)] succeeds(p3,p7)                extend by 11
26. [succeeds(p3,p3)] [succeeds(p3,p6)] [succeeds(p3,p7)] succeeds(p3,x1)
    succeeds(x1,p7)                                                    extend by 16
27. [succeeds(p3,p3)] [succeeds(p3,p6)] [succeeds(p3,p7)] succeeds(p3,x1)
    [succeeds(x1,p7)] follows(x1,p7)                                   extend by 15
28. [succeeds(p3,p3)] [succeeds(p3,p6)] [succeeds(p3,p7)] succeeds(p3,p8) extend by 13
29. [succeeds(p3,p3)] [succeeds(p3,p6)] [succeeds(p3,p7)] [succeeds(p3,p8)]
    has(p8,goto(x1)) labels(x1,p3)                                     extend by 17
30. [succeeds(p3,p3)] [succeeds(p3,p6)] [succeeds(p3,p7)] [succeeds(p3,p8)]
    has(p8,goto(loop))                                                 extend by 4
31. empty                                                              extend by 14
```

Proof B

```
19. succeeds(p3,p3)                                                     theorem
20. [succeeds(p3,p3)] succeeds(p3,x1) succeeds(x1,p3)                  extend by 16
21. [succeeds(p3,p3)] succeeds(p3,x1) [succeeds(x1,p3)] succeeds(x1,x2)
    succeeds(x2,p3)                                                    extend by 16
22. [succeeds(p3,p3)] succeeds(p3,x1) [succeeds(x1,p3)] succeeds(x1,x2)
    [succeeds(x2,p3)] follows(x2,p3)                                   extend by 15
23. [succeeds(p3,p3)] succeeds(p3,x1) [succeeds(x1,p3)] succeeds(x1,p6) extend by 9
24. [succeeds(p3,p3)] succeeds(p3,x1) [succeeds(x1,p3)] [succeeds(x1,p6)]
    follows(x1,p6)                                                     extend by 15
25. [succeeds(p3,p3)] succeeds(p3,p7)                                  extend by 11
26. [succeeds(p3,p3)] [succeeds(p3,p7)] succeeds(p3,x1) succeeds(x1,p7) extend by 16
27. [succeeds(p3,p3)] [succeeds(p3,p7)] succeeds(p3,x1) [succeeds(x1,p7)]
    follows(x1,p7)                                                     extend by 15
28. [succeeds(p3,p3)] [succeeds(p3,p7)] succeeds(p3,p8)                extend by 13
29. [succeeds(p3,p3)] [succeeds(p3,p7)] [succeeds(p3,p8)] has(p8,goto(x1))
    labels(x1,p3)                                                      extend by 17
30. [succeeds(p3,p3)] [succeeds(p3,p7)] [succeeds(p3,p8)] has(p8,goto(loop)) extend by 4
31. empty                                                              extend by 14
```

2. SHORTBURST Example

Proof A

```
11. succeeds(p3,p3)                                                     theorem
12. [succeeds(p3,p3)] succeeds(p3,x1) succeeds(x1,p3)                  extend by 8
13. [succeeds(p3,p3)] succeeds(p3,x1) [succeeds(x1,p3)] follows(x1,p3) extend by 7
14. [succeeds(p3,p3)] succeeds(p3,p8)                                  extend by 5
15. [succeeds(p3,p3)] [succeeds(p3,p8)] has(p8,goto(x1)) labels(x1,p3) extend by 9
16. [succeeds(p3,p3)] [succeeds(p3,p8)] has(p8,goto(loop))             extend by 1
17. empty                                                              extend by 6
```

Proofs

3. HAS-PARTS Example 1

Proof A

```
 8. hp(John,t(n2,n1),hand)                                                        theorem
 9. [hp(John,t(n2,n1),hand)] hp(John,n2,x1) hp(sk1(John,x1,hand,n2,n1),n1,hand)   extend by 4a
10. [hp(John,t(n2,n1),hand)] hp(John,n2,x1)
    [hp(sk1(John,x1,hand,n2,n1),n1,hand)] in(sk1(John,x1,hand,n2,n1),arm)         extend by 7
11. [hp(John,t(n2,n1),hand)] hp(John,n2,arm)                                      extend by 3b
12. [hp(John,t(n2,n1),hand)] [hp(John,n2,arm)] in(John,human)                     extend by 6
13. [hp(John,t(n2,n1),hand)] [hp(John,n2,arm)] [in(John,human)] in(John,boy)      extend by 2
14. empty                                                                         extend by 1
```

4. HAS-PARTS Example 2

Proof A

```
 8. hp(John,t(t(n2,n1),n5),fingers)                                               theorem
 9. [hp(John,t(t(n2,n1),n5),fingers)] hp(John,t(n2,n1),x1)
    hp(sk1(John,x1,fingers,t(n2,n1),n5),n5,fingers)                               extend by 4a
10. [hp(John,t(t(n2,n1),n5),fingers)] hp(John,t(n2,n1),x1)
    [hp(sk1(John,x1,fingers,t(n2,n1),n5),n5,fingers)]
    in(sk1(John,x1,fingers,t(n2,n1),n5),hand)                                     extend by 5
11. [hp(John,t(t(n2,n1),n5),fingers)] hp(John,t(n2,n1),hand)                      extend by 3b
12. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)] hp(John,n2,x1)
    hp(sk1(John,x1,hand,n2,n1),n1,hand)                                           extend by 4a
13. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)] hp(John,n2,x1)
    [hp(sk1(John,x1,hand,n2,n1),n1,hand)] in(sk1(John,x1,hand,n2,n1),arm)         extend by 7
14. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)] hp(John,n2,arm)    extend by 3b
15. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)]
    [hp(John,n2,arm)] in(John,human)                                              extend by 6
16. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)]
    [hp(John,n2,arm)] [in(John,human)] in(John,boy)                               extend by 2
17. empty                                                                         extend by 1
```

Proof B

```
 8. hp(John,t(t(n2,n1),n5),fingers)                                               theorem
 9. [hp(John,t(t(n2,n1),n5),fingers)] hp(John,t(n2,n1),x1)
    ¬in(sk1(John,x1,fingers,t(n2,n1),n5),x1)                                      extend by 3a
10. [hp(John,t(t(n2,n1),n5),fingers)] hp(John,t(n2,n1),hand)
    [¬in(sk1(John,hand,fingers,t(n2,n1),n5),hand)]
    ¬hp(sk1(John,hand,fingers,t(n2,n1),n5),n5,fingers)                            extend by 5b
11. [hp(John,t(t(n2,n1),n5),fingers)] hp(John,t(n2,n1),hand)                      extend by 4b
12. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)] hp(John,n2,x1)
    hp(sk1(John,x1,hand,n2,n1),n1,hand)                                           extend by 4a
13. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)] hp(John,n2,x1)
    [hp(sk1(John,x1,hand,n2,n1),n1,hand)] in(sk1(John,x1,hand,n2,n1),arm)         extend by 7
14. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)] hp(John,n2,arm)    extend by 3b
15. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)]
    [hp(John,n2,arm)] in(John,human)                                              extend by 6
16. [hp(John,t(t(n2,n1),n5),fingers)] [hp(John,t(n2,n1),hand)]
    [hp(John,n2,arm)] [in(John,human)] in(John,boy)                               extend by 2
17. empty                                                                         extend by 1
```

5. PRIM Example

Proof A

10.	Px1 Dx1a	theorem
11.	Px1 [Dx1a] Dx1x2 Dx2a	extend by 2
12.	Px1 [Dx1a] Dx1g(a) [Dg(a)a] ¬Pa	extend by 3a
13.	Px1 [Dx1a] Dx1g(a) [Dg(a)a] [¬Pa] Daa	extend by 9b
14.	Px1 [Dx1a] Dx1g(a)	extend by 1
15.	Pf(g(a)) [Df(g(a))a] [Df(g(a))g(a)] Ln1g(a) Lg(a)a	extend by 7
16.	Pf(g(a)) [Df(g(a))a] [Df(g(a))g(a)] Ln1g(a) [Lg(a)a] ¬Pa	extend by 5a
17.	Pf(g(a)) [Df(g(a))a] [Df(g(a))g(a)] Ln1g(a) [Lg(a)a] [¬Pa] Daa	extend by 9b
18.	Pf(g(a)) [Df(g(a))a] [Df(g(a))g(a)] Ln1g(a)	extend by 1
19.	Pf(g(a)) [Df(g(a))a] [Df(g(a))g(a)] [Ln1g(a)] ¬Pa	extend by 4a
20.	Pf(g(a)) [Df(g(a))a] [Df(g(a))g(a)] [Ln1g(a)] [¬Pa] Daa	extend by 9b
21.	Pf(g(a))	extend by 1
22.	[Pf(g(a))] Ln1g(a) Lg(a)a	extend by 6
23.	[Pf(g(a))] Ln1g(a) [Lg(a)a] ¬Pa	extend by 5a
24.	[Pf(g(a))] Ln1g(a) [Lg(a)a] [¬Pa] Daa	extend by 9b
25.	[Pf(g(a))] Ln1g(a)	extend by 1
26.	[Pf(g(a))] [Ln1g(a)] ¬Pa	extend by 4a
27.	[Pf(g(a))] [Ln1g(a)] [¬Pa] Daa	extend by 9b
28.	empty	extend by 1

Proof B

10.	Px1 Dx1a	theorem
11.	Pa	extend by 1
12.	[Pa] ¬Ln1g(a)	extend by 4b
13.	[Pa] [¬Ln1g(a)] Lg(a)a ¬Pf(g(a))	extend by 6c
14.	[Pa] [¬Ln1g(a)] Lg(a)a [¬Pf(g(a))] Df(g(a))a	extend by 9b
15.	[Pa] [¬Ln1g(a)] Lg(a)a [¬Pf(g(a))] [Df(g(a))a] Df(g(a))x1 Dx1a	extend by 2
16.	[Pa] [¬Ln1g(a)] Lg(a)a [¬Pf(g(a))] [Df(g(a))a] Df(g(a))g(a)	extend by 3a
17.	[Pa] [¬Ln1g(a)] Lg(a)a	extend by 7
18.	empty	extend by 5a

Proof C

10.	Px1 Dx1a	theorem
11.	Pa	extend by 1
12.	[Pa] ¬Dg(a)a	extend by 3b
13.	[Pa] [¬Dg(a)a] Dx1g(a) ¬Dx1a	extend by 2b
14.	[Pa] [¬Dg(a)a] Dx1g(a) [¬Dx1a] Px1	extend by 9a
15.	[Pa] [¬Dg(a)a] Df(x1)g(a) [¬Df(x1)a] [Pf(x1)] Ln1x1 Lx1a	extend by 6
16.	[Pa] [¬Dg(a)a] Df(g(a))g(a) [¬Df(g(a))a] [Pf(g(a))] Ln1g(a)	extend by 5a
17.	[Pa] [¬Dg(a)a] Df(g(a))g(a)	extend by 4a
18.	[Pa] [¬Dg(a)a] [Df(g(a))g(a)] Ln1g(a) Lg(a)a	extend by 7
19.	[Pa] [¬Dg(a)a] [Df(g(a))g(a)] Ln1g(a)	extend by 5a
20.	empty	extend by 4a

6. ANCES1 Example

Proof A

7.	H ¬C	theorem
8.	H	extend by 6a
9.	[H] J ¬A	extend by 1a
10.	[H] J [¬A] B	extend by 4b
11.	[H] J	extend by 5a
12.	[H] [J] ¬K	extend by 2a
13.	empty	extend by 3c

7. NUM1 Example

Proof A

```
 7. Dab                                        theorem
 8. [Dab] Pa Mx1bx2 Dax2 ¬Dax1                 extend by 1a
 9. [Dab] Pa Mbbx1 Dax1                        reduce
10. [Dab] Pa Mbbx1 [Dax1] Max2x1               extend by 4
11. [Dab] Pa Mbbs(b)                           extend by 5
12. [Dab] Pa                                   extend by 2
13. empty                                      extend by 6
```

8. GROUP1 Example

Proof A

```
 7. Pj(x1)x1j(x1)                                           theorem
 8. [Pj(x1)x1j(x1)] Px2x3j(x1) Px3x1x4 Px2x4j(x1)           extend by 1
 9. [Pj(x1)x1j(x1)] Pg(x2j(x1))x3j(x1) Px3x1x2              extend by 3
10. [Pj(h(x1x2))h(x1x2)j(h(x1x2))] Pg(x2j(h(x1x2)))x1j(h(x1x2))   extend by 4
11. empty                                                   extend by 3
```

9. GROUP2 Example

Proof A

```
 7. Pbac                                       theorem
 8. [Pbac] Pbx1x2 Px1x3a Px2x3c                extend by 3
 9. [Pbac] Pbx1e Px1ca                         extend by 2
10. [Pbac] Pbx1e [Px1ca] Px2x3x1 Px3cx4 Px2x4a extend by 4
11. [Pbac] Pbx1e [Px1ca] Pax2x1 Px2ce          extend by 1
12. [Pbac] Pbx1e [Px1ca] Pacx1                 extend by 5
13. [Pbac] Pbx1e [Px1ca] [Pacx1] Pax2x3 Px2x4c Px3x4x1   extend by 3
14. [Pbac] Pbx1e [Px1ca] [Pacx1] Pax2e Px2x1c  extend by 2
15. [Pbac] Pbbe [Pbca] [Pacb] Paae             extend by 6
16. [Pbac] Pbbe                                extend by 5
17. empty                                      extend by 5
```

Proof B

```
 7. Pbac                                       theorem
 8. [Pbac] Pbx1x2 Px1x3a Px2x3c                extend by 3
 9. [Pbac] Pbx1a Px1ba                         extend by 6
10. [Pbac] Pbx1a [Px1ba] Px2x3x1 Px3bx4 Px2x4a extend by 4
11. [Pbac] Pbx1a [Px1ba] Pax2x1 Px2be          extend by 1
12. [Pbac] Pbx1a [Px1ba] Pabx1                 extend by 5
13. [Pbac] Pbca                                extend by 6
14. [Pbac] [Pbca] Px1x2b Px2cx3 Px1x3a         extend by 4
15. [Pbac] [Pbca] Pax1b Px1ce                  extend by 1
16. [Pbac] [Pbca] Pacb                         extend by 5
17. [Pbac] [Pbca] [Pacb] Pax1x2 Px1x3c Px2x3b  extend by 3
18. [Pbac] [Pbca] [Pacb] Pax1e Px1bc           extend by 2
19. [Pbac] [Pbca] [Pacb] Paae                  extend by 6
20. empty                                      extend by 5
```

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* THE PROGRAMMABLE STRATEGY THEOREM PROVER: AN IMPLEMENTATION OF THE LINEAR MESON PROCEDURE		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Mark E. Stickel		8. CONTRACT OR GRANT NUMBER(s) F44620-73-C-0074
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Department of Computer Science Pittsburgh, Pennsylvania 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE June, 1974
		13. NUMBER OF PAGES 64
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Air Force Office of Scientific Research (NM) Agency 1400 Wilson Boulevard Arlington, Virginia 22209		15. SECURITY CLASS. *(of this report)* unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Released for public distribution; unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

The Programmable Strategy Theorem Prover (PSTP) is a theorem proving program for the first order predicate calculus using the linear MESON procedure as the inference system. The linear MESON procedure is a new variant of the model elimination theorem proving procedure for theories with or wihout equality which is an affirmation rather than a refutation procedure. It is profitably viewed as an extension of the problem - reduction method. The fundamental element of a linear MESON proceudre deduction, the chain, is a representation of a set of goals to be solved and their supergoals. PSTP is designed to be used ineractively or in a fully automatic mode. Some featuress of PSTP are general mechanism for specifying which chains are to be retained and manipulated, an automatic procedure for storing and retrieving information about chains when this information is requested, the capability of specifying an ordering function which can be used for specifying search strategies, and a powerful set of commands. Results of an experiment testing some simple search strategies and comparisons with results from other theorem proving studies are presented.