David A. Plaisted
Yunshan Zhu

The Efficiency of Theorem Proving Strategies

A Comparative and Asymptotic Analysis

2nd Edition

vieweg

Computational Intelligence

David A. Plaisted
Yunshan Zhu

The Efficiency of Theorem Proving Strategies

David A. Plaisted
Yunshan Zhu

The Efficiency of Theorem Proving Strategies

A Comparative and Asymptotic Analysis

2nd, revised Edition

SPRINGER FACHMEDIEN WIESBADEN GMBH

1st Edition 1997
2nd Edition 1999

http://www.vieweg.de

Cover design: Ulrike Weigel, www.CorporateDesignGroup.de
Produced by: Lengericher Handelsdruckerei, Lengerich
Printed on acid-free paper

ISSN 0949-5665
ISBN 978-3-528-15574-2 ISBN 978-3-663-07847-0 (eBook)
DOI 10.1007/978-3-663-07847-0

Contents

Preface

This is the second and slightly revised edition of this book. A few errors have been fixed, and some references to related work have been added. I thank the readers for their comments on the first edition.

We analyze the search efficiency of a number of common refutational theorem proving strategies on propositional and near-propositional problems. Search efficiency is concerned with the total number of proofs and partial proofs generated, rather than with the sizes of the proofs. We show that most common strategies produce search spaces of exponential size even on simple sets of clauses, or else are not sensitive to the goal. However, clause linking, which uses a reduction to propositional calculus, has behavior that is more favorable in some respects, a property that it shares with methods that cache subgoals. A strategy which is of interest for term-rewriting based theorem proving is the A-ordering strategy, and we discuss it in some detail. We show some advantages of A-ordering over other strategies, which may help to explain its efficiency in practice. We also point out some of its combinatorial inefficiencies, especially in relation to goal-sensitivity and irrelevant clauses. In addition, SLD-resolution, which is of importance for Prolog implementation, has combinatorial inefficiencies; this may suggest basing Prolog implementations on a different theorem proving strategy.

We also develop techniques for studying the asymptotic complexity of first-order theorem provers. This permits an analytic comparison of the worst-case performances of various theorem proving methods. We show that the speeds of various well-known methods range in complexity from singe to quintuple exponential with respect to various natural measures of the hardness of the input. This study reveals some of the efficiencies and inefficiencies of known methods, and also helps to suggest new strategies and combinations of strategies with good complexity properties.

Chapters 1 and 2 are completely independent of one another. The work in Chapter 1 was done while the first author was on sabbatical and leave of absence at the Max Planck Institute in Saarbrücken, Germany. This work was partially supported by UNC Chapel Hill and by the National Science Foundation under grant number CCR-9108904.

A short version of Chapter 1 appeared at the Twelfth Conference on Automated Deduction in Nancy, France, June - July, 1994. A longer version appeared in [Pla94c]. Portions of Chapter 1 also appeared in [PA96]. The work in Chapter 2 was jointly done by both authors and was partially supported by UNC Chapel Hill and by the National Science Foundation under grant number CCR-9627316. This book was typeset by the first author using LaTeX and BibTeX. Thanks are due to Sid Chatterjee for explaining **xfig** and embedded postscript.

Chapter 1

The Propositional Complexity of First-Order Theorem Proving Strategies

Many shall run to and fro, and knowledge shall be increased.

Dan. 12:4

1.1 Introduction

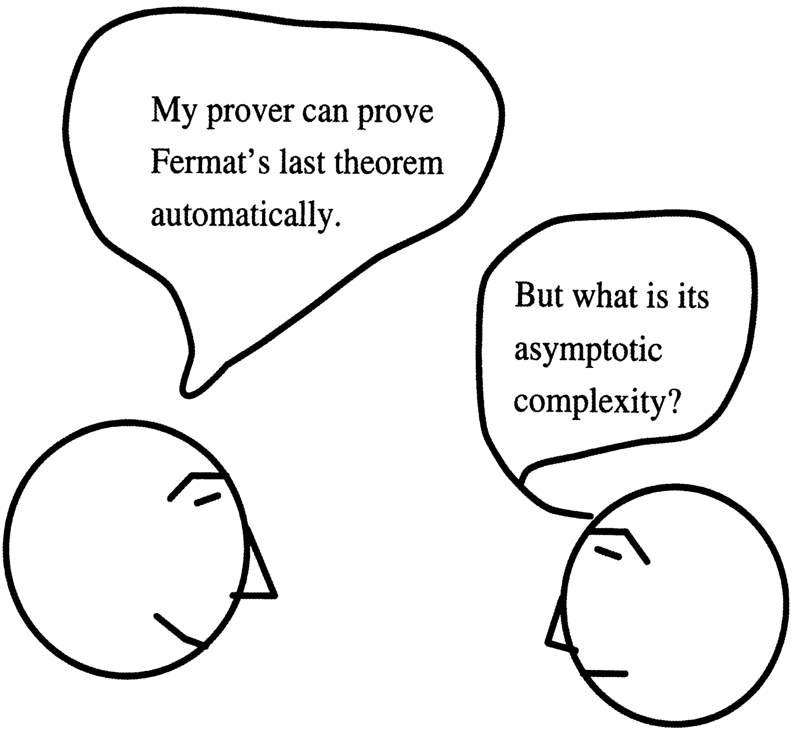
The efficiency of a theorem prover is more directly influenced by the total number of inferences performed before a proof is found than by the size of the final proof. In general, in the field of automated deduction for full first-order logic, there has been a great deal of attention devoted to the completeness of strategies but little to their efficiency, in the sense of the total work expended in the search for a proof. The main efficiency considerations to date have to do with the times needed by particular implementations to find proofs of particular example theorems, or with the efficiencies of decision procedures for specialized theories. Of course, there has also been work on the efficiencies of low-level operations employed by theorem provers (such as unification). It is informative (and fun) to evaluate a prover by running it on a series of examples, but this could well be supplemented by analytical results. To this end, a theoretical study would be useful. It would be nice to know something about the behaviors of proposed new strategies without having to read and understand papers about them or having to run them on examples. Theoretical measures of search space size would permit this. Such measures would also make it easier to weed out bad strategies early and would stimulate the development of good ones. There is more at issue than just a quantitative measure of performance – analytical measures reveal something about how a strategy works, and how it does subgoaling. This gives some insight into the strategy. A theoretical approach could also help to pinpoint problem areas and weaknesses in a method and lead to improvements. In general, theory does not replace experiment but it does supplement it, and provides insights that might otherwise be missed. Theory tends to make general statements and to be machine-independent, whereas experiment tends to deal in specifics and to be machine-dependent. This paper is an attempt to initiate (or further) a theory of the search efficiency of automated theorem proving.

In sum, we are interested in the sizes of the search spaces produced by clause form refutational theorem proving strategies for first-order logic. This interest is different from that of most logicians who are interested in provability or the length of proofs. For some examples of the latter, see [CR79, Hak85, Urq87, Ede92]. The paper [Let93] studies how accurately the length of a derivation reflects the actual complexity of a proof. By the search space size we mean the number of proofs and partial proofs. This latter measure is more relevant for the efficiency of theorem provers than the size of a minimal proof. There has been very little work on search space size. The paper [KBL93] shows that many refinements of resolution do not increase a certain measure of search space size by more than a factor of four, but does not compare refinements with one another. Their paper considers monotone refinements of resolution; these do not allow deletion operations such as deletion of subsumed clauses. However, the results are otherwise very general. We demonstrate some surprising and little appreciated inefficiencies of many common strategies, which may help to explain their poor performance on some kinds of problems. We also discuss the clause linking method [LP92] and methods that cache subgoals and show that they overcome some of these limitations. We present some examples where resolution has better performance. These analyses are interesting because they do not depend on particular machine architectures or data structures used to implement strategies, and are thus of a more universal nature. We only consider clause form refutational theorem proving methods for first-order logic; it would be interesting to extend this analysis to Hilbert-style, sequent-style, semantic tableau, and other methods. We emphasize Horn clauses, which are common in practice. We analyze the behavior of strategies on propositional Horn sets as well as giving some first-order clauses sets with a similar behavior.

We believe that our theoretical results are reflected in practice, both for strategies and their refinements. For example, we show that negative resolution on Horn sets is inefficient theoretically; this is also frequently true in practice. As a result, one can expect some practical benefit from this work. It may lead to the development of strategies that are more efficient in practice, as well as helping to reveal the comparative value of refinements to strategies. For example, under some circumstances, ordering predicate symbols improves the efficiency of resolution, and in other cases, it significantly degrades performance. This analysis also highlights the efficiency to be gained in model elimination and the MESON strategy [Lov78] by unit lemmas and caching, which reduce exponential behavior to polynomial behavior for Horn clauses. Also, we feel that an analytical approach will help to point out some underlying problems in the field, which need to be addressed before mechanical theorem provers can be reliable assistants to human mathematicians. A failure to address these issues can only be detrimental, as users become frustrated with the performance of their provers and don't understand the reasons for the inefficiencies. The kinds of problems where resolution and similar methods perform well are in many cases Horn clause problems, or problems of a similar nature; these are also the kinds of problems for which the theoretical analysis indicates good behavior (as long as goal-sensitivity is not important). It is on such Horn clause problems that many of the publicized successes of resolution in solving open problems, have occurred. This may give an impression of the power of existing theorem provers that does not correspond to their performance on the types of problems more likely to be encountered by a typical user.

Some of these results are particularly interesting because of their implications for neighboring areas of research. We discuss theorem proving methods based on term-rewriting, which correspond to the A-ordering refinement of resolution for propositional logic. Term-rewriting is of interest because it is often very efficient on pure equational problems. We show that from a theoretical standpoint, A-ordering has some significant advantages over other strategies, although it also has some severe problems, especially if the ordering is not chosen properly. Moreover, a good ordering can be hard to find: we give some evidence in section 1.5.10 that it is not always possible to choose an ordering that is natural, goal-sensitive, and efficient, even for unsatisfiable clause sets. This suggests that it may be difficult in general to obtain efficient goal-sensitive term-rewriting based theorem provers for first-order logic, and that other methods may have to be used. Giving up goal-sensitivity seems like a high price to pay, although it is conceivable that one could prove theorems efficiently without considering the goal. Also, we give a set of clauses for which A-ordering, even with a good ordering, generates an exponential number of clauses. Turning our attention now to logic programming, we show that SLD-resolution [Llo87] also has severe inefficiencies in some cases. Since SLD-resolution is the basis for logic programming implementations, this result may suggest the possibility of basing Horn-clause logic programming implementations on other theorem proving strategies.

Furthermore, this work highlights what we feel is a dilemma of theorem proving, namely, that most strategies are either inefficient on Horn clauses or are not sensitive to the theorem being proved. For hard problems, it seems essential to have a strategy that works backwards from the theorem to try to find a proof. Although some fairly hard theorems can be proved without backward reasoning, it seems unlikely that a strategy that simply combines general axioms will make much progress, in general. However, for Horn clauses, strategies that work backwards tend to be highly inefficient, and many problems consist largely of Horn clauses. The author has been aware of this problem for some time, and has developed some strategies to avoid this problem. But our impression is that few in the field appreciate this issue properly. Even the strategies that overcome this problem have additional problems of their own. The clause linking strategy of [LP92] is a back chaining strategy that is efficient on Horn clauses but sometimes needs to retain instances of more general clauses. Clause linking with semantics [CP94] is efficient on Horn clauses and makes use of semantics, but sometimes needs to enumerate ground terms, which we would also like to avoid. Also, the base method used for clause linking with semantics does not involve unification. Despite this, its notable successes on certain hard problems tends to confirm our theoretical considerations. We would like to find a back chaining strategy that is efficient on Horn clauses, based on unification, and still always permits instances to be deleted. Some such strategies exist; they are the MESON strategy, model elimination [Lov78], and the simple and modified problem reduction formats [Pla82, Pla88], all with caching. However, of these, either caching of unit lemmas and subgoals is not complete for first-order logic, as for the first two, or the strategies have propositional inefficiencies for non-Horn problems, as for the second two. For the MESON strategy and model elimination, if caching of non-unit lemmas and subgoals is done, then the efficiency on Horn clauses is lost.


My prover can prove Fermat's last theorem automatically.
But what is its asymptotic complexity?

1.2 First Order Logic and Refutational Theorem Proving

We assume the standard definitions of propositional and first-order logic. For a discussion of first-order logic and theorem proving strategies see [CL73, Lov78, Bun83, WOLB84]. We restrict our attention to clause form first-order refutational theorem proving. A *term* is a well-formed expression composed of variables and function and constant symbols, such as, $f(x, g(y, c))$. An *atom* is a predicate symbol, possibly followed by a list of terms. For example, P and $Q(a, f(x))$ are atoms. A *literal* is an atom or an atom preceded by a negation sign. For example, $\neg P(b)$ is a literal. A literal is called *positive* if it lacks a negation sign and *negative* if it contains a (single) negation sign. A *clause* is a set of literals, signifying their disjunction. Thus $\{P, \neg Q\}$ is a clause signifying $P \vee \neg Q$. Variables in a clause are assumed to be implicitly universally quantified. Thus the clause $\{P(x), \neg Q(x)\}$ means $(\forall x)(P(x) \vee \neg Q(x))$. A clause is *positive* if all of its literals are positive, and *negative* if all of its literals are negative; often we say *all-negative* for emphasis. A *Horn clause* is a clause having at most one positive literal. Thus $\{\neg P, \neg Q, R\}$ is a Horn clause. Such clauses are commonly used in Prolog programs. A set of clauses signifies the conjunction of the clauses in the set. For example, the set $\{\{\neg Q(x), P(x)\}, \{\neg P(y), Q(y)\}\}$ signifies the formula $(\forall x)(\forall y)((\neg Q(x) \vee P(x)) \wedge (\neg P(y) \vee Q(y)))$. Thus, a set of clauses represents a quantifier-free first-order formula in conjunctive normal form. It is known that any first-order formula can be converted to this form efficiently in a satisfiability-preserving manner. The theorem proving problem (in this refutational format) is to decide if such a set of clauses is unsatisfiable. The general problem is only partially decidable. A number of strategies have been developed to partially decide this property. We are interested in comparing their efficiency. We say a strategy is *complete* if it correctly reports whenever a set of clauses is unsatisfiable but may fail to terminate if the set of clauses is satisfiable.

A literal M is an *instance* of a literal L if M is obtained from L by replacing variables by terms in a systematic way, that is, all occurrences of the same variable are replaced by the same term. Thus $P(f(a), f(a))$ is an instance of $P(x, x)$. We similarly define what it means for a clause

D to be an instance of a clause C. We define the operation of *unit simplification* as follows: Suppose we have a unit clause $\{L\}$ and another clause $\{M_1, ..., M_n\}$, where L and M_1 are complementary. Then, the clause $\{M_1, ..., M_n\}$ can be deleted and replaced by $\{M_2, ..., M_n\}$. This extends to first-order logic; in that case, we require that M_1 be an instance of the negation of L.

We also define *pure literal clause deletion* as follows: Suppose S is a set of clauses and C is a clause in S. Suppose L is a literal in C and there is no literal M in any clause of S such that L and the complement of M are unifiable. Then L is said to be *pure*. Also, in this case, $S - \{C\}$ is unsatisfiable iff S is. So, pure literal clause deletion is the operation of deleting such clauses from S. This may cause other literals to become pure. Sometimes all of S can be deleted by repeated pure literal clause deletion. In this case, S is satisfiable.

We also define *subsumption*. In the propositional setting, a clause C subsumes D if C is a subset of D. We say C *properly* subsumes D if C is a proper subset of D. In the first-order setting, we say that C subsumes D if C has a (substitution) instance that is a subset of D. Note that if C subsumes D, then C logically implies D. If C is derived, then one can often simplify clause sets by removing subsumed clauses D without losing completeness.

1.3 Search Space Formalism

We formalize theorem proving strategies as directed graphs. Formally, a *theorem proving strategy* is a 5-tuple $< S, V, i, E, u >$ where S is a set of states, i maps the input clauses to a set of states, E is a set of *edges* (pairs of states), and u maps S to $\{True, False\}$. Each state s is labeled with a set $label(s)$ of elements from some underlying set V of structures (such as clauses or chains). If an edge (s, t) is in E, this means that t is a possible successor state to s. Thus, (S, E) is a directed graph. We require that no two distinct edges (s_1, t_1), (s_2, t_2) have $t_1 = t_2$. Thus the graph is a set of trees. Also, u is an unsatisfiability test; $u(s)$ is $True$ if the state s corresponds to a proof of unsatisfiability. We say such a strategy is *complete* if for all sets R of clauses, if R is unsatisfiable then there exists a path from some element of $i(R)$ to a state s such that $u(s)$

is *True*. We say such a strategy is *sound* if R is unsatisfiable whenever there is a path from some element of $i(R)$ to a state s such that $u(s)$ is *True*. A strategy is *linear* if for all s in S, there is a unique t in S such that there is an edge from s to t in E. The intention of this definition is that i and u are computable and of low complexity. Let F_M be the set of ordered pairs $\{(s, \{t : (s,t) \in E\}) : s \in S\}$ for a strategy M. Thus, $F_M(s)$ is the set of successors of a state s. We require that F_M be a function, in the sense that if the labels of s_1 and s_2 are the same then the sets of labels of their successors should also be the same. (Recall that each state is labeled with a set of elements of V.) Also, we intend that F_M should be computable and of low complexity. Often we omit V and write the strategy as a 4-tuple $< S, i, E, u >$.

As an example, we formalize resolution in this way. For this, we have the 5-tuple $< S, V, i, E, u >$ where each state in S is labeled with a finite set of clauses, V is the set of all clauses over some set of predicates and function symbols, $i(R) = \{R\}$ for all R, and (s,t) is in E if t is s together with all resolvents of clauses in s. Thus resolution is a linear strategy, in this formalism. Finally, $u(s) = True$ iff the empty clause is in $label(s)$. Now, resolution formalized in this way is complete, since if R is unsatisfiable, there is a resolution proof of the empty clause from R. Also, resolution is sound. In contrast, model elimination is not linear in this formalism. For model elimination, the labels of the states consist of single chains. Here $i(R)$ is a set of states, one for each clause in R, each state labeled with a singleton set containing a single chain. Also, (s,t) is in E if the chain in the label of t is obtained by a permissible operation (extension, reduction, or contraction) from the chain in the label of s. Thus, strategies that are conventionally thought of as linear, become non-linear in this framework, but strategies that are non-linear like resolution become linear in this framework.

1.4 Measures of Search Duplication

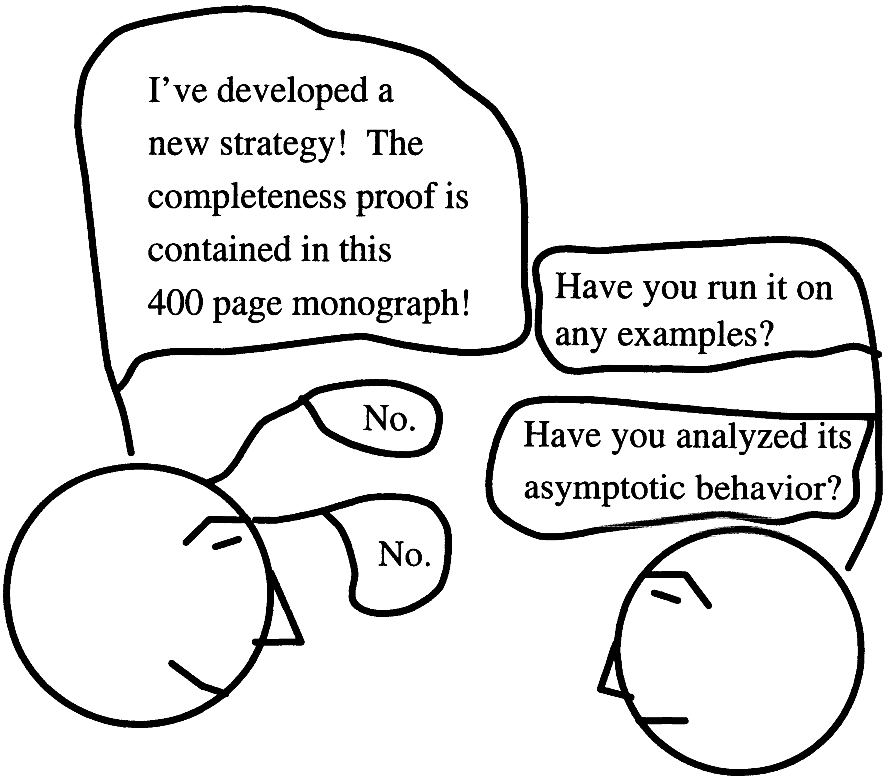
We now define some measures of search space duplication for such strategies. For this, we assume that R is a set of propositional clauses, for simplicity, although these ideas can be lifted to first-order logic. We can think of a search space $G =< S, V, i, E, u >$ as a function mapping a set R of clauses to a graph $G(R)$ representing the search space for R. For this, we define an *initial state* to be an element of $i(R)$ and a *final state* to be a state s such that $u(s) = True$. Thus the task of the theorem prover is to find a path from an initial to a final state. We say a state s is *reachable* from R if there is a path from some element of $i(R)$, to s. We are only interested in the nodes s that are reachable from R. Also, we are only interested in edges in E that occur on some such path. So, we define $S(R)$ to be the set of nodes reachable from R. We define $E(R)$ to be the set of edges in E that occur on some path of reachable states. Also, we define $G(R)$ to be the graph $< S(R), E(R) >$. Let $|T|$ be the number of elements in a set T. Then, we are interested to know how $|S(R)|$ depends on the length $c(R)$ of R, represented as a string of characters. For example, is $|S(R)|$ linear in $c(R)$, polynomial in $c(R)$, or exponential in $c(R)$? Also, we are interested in the structure of the states. Recall that S is a set of states, each labeled with a set of structures indicating lemmas or partial proofs. We are interested in how big these sets of structures can become, because this is a meaningful measure of search complexity. Thus, the most meaningful measure of search complexity is the sum, over all s in $S(R)$, of $|label(s)|$. Let us call this measure $||G(R)||$, and refer to it as the total duplication for R.

To further refine this measure, we consider three other measures: 1. The maximum length of a path in $G(R)$. 2. The maximum size of a subset of $S(R)$, no two elements of which are on the same path. 3. The maximum of $|label(s)|$ for all s in $S(R)$. We call the first, the duplication by *iteration*, the second, the duplication by *case analysis*, and the third, the duplication by *combination*. The intuition for this is that the length of a path represents the number of times that search must be iterated. Also, each path represents a case that must be considered in the search, so the second measure indicates the number of cases there are. The third measure concerns the sizes of the labels of the states. If the sizes of the labels are large, then there must be many elements of V in the same state label. However, in common propositional strategies, the elements of V are constructed from the predicates appearing in the input clauses. This means that there must be many combinations of these predicates, hence the term duplication by combination.

For each measure, we are interested in whether it is a constant, polynomial, or exponential in $c(R)$. We are also interested in the size of the total duplication $||G(R)||$. It is not difficult to show that $||G(R)||$ is bounded by the product of these three measures. To see this, we note that $G(R)$ is a tree. Each tree is a union of a set of paths from the root to a leaf. We can thus identify each state of $G(R)$ with a pair $(path, position)$ where the position tells the distance from the root. We thus have that the number of ordered pairs (s, v) such that $v \in label(s)$ is equal to the number of triples $(path, position, v)$ where $v \in label(s)$ for s the state corresponding to $(path, position)$. Thus the number of such ordered pairs (s, v) is bounded by the product of the number of paths, the length of the longest path, and the number of elements in the largest label. But the number of such ordered pairs is just $||G(R)||$ and the product is just the product of the three measures of duplication. This shows that the total duplication is bounded by the product of duplication by iteration, combination, and case analysis.

We say the duplication by iteration for R is *constant* if the duplication by iteration is bounded. We say the duplication by iteration for R is *linear* if the ratio of the duplication by iteration to $c(R)$ is bounded. We say it is *polynomial* if the duplication by iteration is polynomial in $c(R)$. We say it is *exponential* if the duplication by iteration is exponential in $c(R)$. Similarly, we can define what it means for duplication by combination and case analysis to be constant, linear, et cetera. We also define in this way what it means for the total duplication to be linear, et cetera. We say a strategy has *polynomial behavior* if all three kinds of duplication are polynomial, or equivalently, if the total duplication is polynomial. We say a strategy has *exponential behavior* if the total duplication is exponential, or equivalently, if one of the three kinds of duplication is exponential.

If a strategy is linear, then a *round* is an edge in $E(R)$. The rounds are ordered; the *first* round is the edge of the form $(i(R), s)$, the second round is the edge of the form (s, t), and so on, so the edges are ordered by their distance from $i(R)$. Sometimes we use a similar terminology for non-linear strategies. It is often useful to discuss the behavior of the rounds in order to analyze a strategy.


I've developed a new strategy! The completeness proof is contained in this 400 page monograph!
Have you run it on any examples?
No.
Have you analyzed its asymptotic behavior?
No.

1.5 Analysis of Duplication for Various Strategies

We are interested in determining the degree of duplication for various strategies and their refinements. In this way we obtain the following chart. This chart shows, in addition to the search space measures for each strategy, whether the strategy is goal sensitive. A strategy is *goal sensitive* for Horn clauses if each inference depends on a negative Horn clause; this means that some kind of backward chaining from the goal clauses is being done. In logic programming applications, one considers the negative clauses as goals or queries, and we adopt the same convention here. This seems to be true of many mathematical theorems as well as logic programs. Of course, there is no intrinsic reason why negative literals should be treated differently than positive literals in a more general context. If a strategy G is goal sensitive, then $G(R)$ will be empty for sets R of Horn clauses containing no all-negative clauses. We also indicate the search depth; this is the maximum length of a path in the search space from an initial to a final state. This indicates the depth at which a proof can be found. This differs from duplication by iteration; duplication by iteration considers essentially the maximum number of rounds of inference that can be done, whether or not a proof is found. This could conceivably be larger than the search depth (for example, if the set of input clauses is satisfiable or if we chose the wrong path to search). Presumably, the prover will not continue to search beyond nodes s for which $u(s) = True$. Thus, in some situations, search depth may give better information than duplication by iteration.

The chart is based on propositional Horn clauses. Horn clauses are interesting because they correspond to a derivation of a fact (atom) from a collection of facts (atoms), and such derivations are common. Horn clauses as a result appear frequently in sets of clauses seen by theorem provers. In addition, Horn clauses are useful for studying how a theorem prover performs subgoaling. Such clause sets are decidable in linear time [DG84]. However, it is conceivable that one could do even better than that. One may not even have to look at all the input if a goal-directed method is used; only the clauses that are in some sense relevant to the goal need to be considered. This could be relevant if there are thousands of input clauses and if many queries are given to the same database of clauses. In addition, our results are transferrable to certain first-order clause sets, as we will show. It is instructive at the beginning to give the simplest sets of clauses illustrating the various behaviors. Another reason for the interest in propositional Horn clauses is because of the dramatic differences they reveal between different strategies, often strategies that differ in fairly small and seemingly insignificant ways.

We would like to emphasize that the functions in this chart are upper bounds, valid for *all* propositional Horn sets. In addition, the bounds are tight, meaning that there are propositional Horn sets for which these bounds are achieved. Since we give several specific sets of clauses below, the reader may get the impression that we are only measuring the search behavior for these sets of clauses. This is an incorrect impression. These clause sets are only used to show that the bounds are tight.

Also, we are not considering which search method is used, whether depth-first, breadth-first, best-first, or some other search method. We only consider the total size of the search space. It's possible that a very good search method could lead to better bounds. However, we are not aware of any search method that can improve on the bounds given below. In particular, breadth-first search and depth-first iterative deepening [Kor85, ST85] should explore a portion of the search space having the same size as that indicated here. That is, if any of the bounds are exponential, these search methods will explore an exponential amount of the search space. Also, for theorem proving strategies having exponential search depth, any search method will (sometimes) explore an exponential amount of the search space.

The following abbreviations are used in this table: hyper-res means hyper-resolution, ord means ordering the literals, P_1-ded means P_1-deduction, 3-lit means 3-literal clauses, res means resolution, A-ord means A-ordering, neg means negative, g.o. means good ordering, b.o. means bad ordering, supp means support, ME means model elimination, lemm means lemmas, cach means caching, sprf means the simplified problem reduction format, mprf means the modified problem reduction format, clin means clause linking, f. means forward, b. means backward, and conn means a connection calculus.

Strategy	Search Depth	Combination	Iteration	Case Analysis	Goal Sensitive
hyper-res	linear	linear	linear	O(1)	no
hyper-res, ord	linear	linear	linear	O(1)	no
P_1-ded	linear	exp.	linear	O(1)	no
P_1-ded, 3 lit	linear	linear	linear	O(1)	no
P_1-ded, ord neg	linear	linear	linear	O(1)	no
res, A-ord	linear	exp.	linear	O(1)	no
all-neg res	linear	exp.	linear	O(1)	yes
all-neg res, g.o.	linear	exp.	?	O(1)	yes
all-neg res, b.o.	exp.	exp.	exp.	O(1)	yes
res, neg supp	linear	exp.	linear	O(1)	yes
ME	exp.	O(1)	exp.	exp.	yes
ME, unit lemm	linear	exp.	linear	O(1)	yes
ME, unit lemm, cach	linear	linear	linear	O(1)	yes
MESON	exp.	O(1)	exp.	exp.	yes
MESON, unit lemm, cach	linear	linear	linear	O(1)	yes
sprf, no cach	exp.	O(1)	exp.	exp.	yes
sprf, cach	linear	linear	linear	O(1)	yes
mprf, no cach	exp.	O(1)	exp.	exp.	yes
mprf, cach	linear	linear	linear	O(1)	yes
clin, f. supp	linear	linear	linear	O(1)	no
clin, b. supp,	linear	linear	linear	O(1)	yes
f. conn.	linear	linear	linear	O(1)	no
b. conn.	exp.	O(1)	exp.	exp.	yes

We can make some general observations about this table. The backward chaining strategies are goal-sensitive, but are mostly inefficient. Forward chaining strategies, though efficient for Horn clauses, are not goal-sensitive. All of the strategies that are goal sensitive have exponential duplication, except for the simplified problem reduction format with caching, the modified problem reduction format with caching, and clause linking with backward support. MESON and model elimination with caching and unit lemmas have this property, but these are not complete for general first-order clauses. A recent implementation of model elimination and unit lemmas with caching is described in [AS92]. Note that some refinements can be very damaging to a strategy. For example, ordering negative literals can severely degrade the performance of negative resolution.

These results are valid for sets of Horn clauses. We might consider a more general set of clauses, namely, those for which a renaming of predicate symbols produces a Horn set. For such clauses, none of the results given above are any better, and many of the results are much worse. For example, P_1 deduction and hyper-resolution can be made as bad as all-negative resolution, since we can choose to reverse the signs of the literals, making P_1 deduction and hyper-resolution simulate all-negative resolution. We believe that the behavior of the simplified and modified problem reduction formats degrades in a similar way. However, clause linking still has only linear duplication of search. This sets it apart from all other strategies considered, but the reason is that unit simplification is built in to this strategy. Without this, it might not have such good behavior either. And of course, other strategies with unit simplification added would have this good behavior also on unsatisfiable Horn sets. However, satisfiable Horn sets are more of a problem for the strategies other than clause linking since clause linking has a model-finding approach to detecting satisfiability that doesn't seem to fit into the other strategies given here. One can show that the model-finding part of Davis and Putnam's method will always succeed in polynomial time for satisfiable propositional Horn sets [GU89].

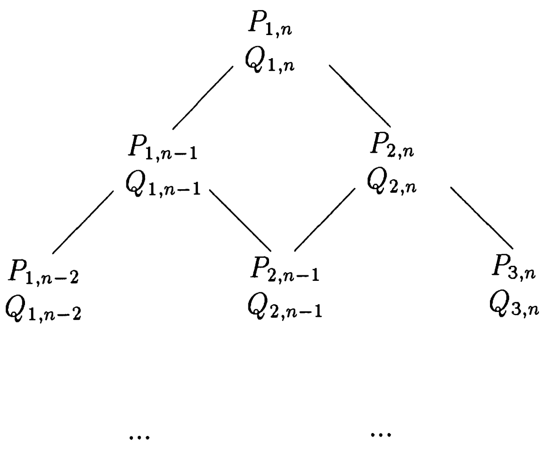
1.5.1 Hard sets of clauses for the strategies

We now indicate how the above results were derived. For this we consider the sets of clauses S_n^1, S_n^2, and S_n^3 defined as follows. Note that S_n^1 is unsatisfiable but S_n^2 and S_n^3 are satisfiable.

Let S_n^1 be the set of $n+2$ clauses $\{\{\neg P_1, \neg P_2, ..., \neg P_n, P\}$, $\{P_1\}$, $\{P_2\}$, ..., $\{P_n\}$, $\{\neg P\}\}$. We sometimes write clauses in Prolog format; a clause $\{P, \neg P_1, ..., \neg P_n\}$ is written as $P\ :-\ P_1, ..., P_n$. A clause $\{\neg P_1, ..., \neg P_n\}$ is written as $:-\ P_1, ..., P_n$. Let S_n^2 be the following clauses, written in Prolog format for readability:

$$
\begin{array}{lrll}
\textit{goal clause} & & :-\ P_{1,n} & \\
\textit{type 1 clauses} & P_{i,j} & :-\ P_{i+1,j}, P_{i,j-1}, & 1 \leq i < j \leq n \\
 & P_{i,j} & :-\ Q_{i+1,j}, Q_{i,j-1}, & 1 \leq i < j \leq n \\
 & Q_{i,j} & :-\ P_{i+1,j}, Q_{i,j-1}, & 1 \leq i < j \leq n \\
 & Q_{i,j} & :-\ Q_{i+1,j}, P_{i,j-1}, & 1 \leq i < j \leq n \\
\textit{type 2 clauses} & P_{i,i} & :-\ P_{i,i+n/2}, & i \leq n/2 \\
 & Q_{i,i} & :-\ Q_{i,i+n/2}, & i \leq n/2 \\
 & P_{i,i} & :-\ P_{i-n/2,i}, & i > n/2 \\
 & Q_{i,i} & :-\ Q_{i-n/2,i}, & i > n/2
\end{array}
$$

The following picture should help to illustrate the structure of S_n^2. The type 2 clauses are not shown.

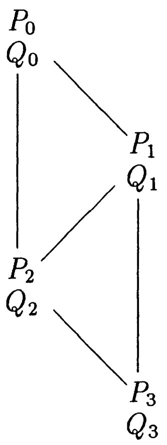


We can think of backward chaining theorem proving strategies on this set of clauses as ways of moving P-pebbles and Q-pebbles around on this graph. Initially, there is a P-pebble on the $(1, n)$ vertex. At each step, we are permitted to remove a pebble. If we remove a P pebble from vertex (i, j), we must either add two P pebbles or two Q pebbles to the two vertices $(i, j-1)$ and $(i+1, j)$ below. If we remove a Q pebble, we must add a P pebble and a Q pebble to these vertices. Note that the parity of the number of Q pebbles never changes unless some Q literal is generated in two or more ways. Later we will formalize this pebbling idea in a more general context.

Let S_n^3 be the following clauses, in Prolog format:

goal clause	$:- P_0, Q_0$	
type 1 *clauses*	$P_i :- P_{i+1}, P_{i+2},$	$0 \leq i < 2n-2$
	$P_i :- Q_{i+1}, Q_{i+2},$	$0 \leq i < 2n-2$
	$Q_i :- P_{i+1}, Q_{i+2},$	$0 \leq i < 2n-2$
	$Q_i :- Q_{i+1}, P_{i+2},$	$0 \leq i < 2n-2$
type 2 *clauses*	$P_{2n-1} :- P_{n-1}$	
	$P_{2n} :- P_n$	
	$Q_{2n-1} :- Q_{n-1}$	
	$Q_{2n} :- Q_n$	

The following picture should help to illustrate the structure of S_n^3. As before, the type 2 clauses are not shown.



...

Here we can think of backward chaining strategies as methods of pebbling this graph. Whenever a pebble is removed from vertex i, pebbles must be added to vertices $i+1$ and $i+2$. As before, there are P pebbles and Q pebbles and the parity of the number of Q pebbles is preserved unless a Q pebble is generated in two ways.

We also introduce the set T_n^2 of clauses which is S_n^2 together with the unit clauses $P_{i,i}$ and $Q_{i,i}$ for $1 \leq i \leq n$. We introduce T_n^3 which is S_n^3 together with the unit clauses P_{2n-2}, P_{2n-1}, Q_{2n-2}, and Q_{2n-1}. T_n^2 and T_n^3 are unsatisfiable, but easy if unit simplification is done.

We now give a sample backward chaining proof attempt. Consider S_n^2. We can resolve the initial goal clause $\neg P_{1,n}$ with $P_{1,n}$:− $P_{2,n}, P_{1,n-1}$ to obtain the clause $\{\neg P_{2,n}, \neg P_{1,n-1}\}$. Then we can resolve this with $P_{2,n}$:− $Q_{3,n}, Q_{2,n-1}$ to obtain the clause $\{\neg Q_{3,n}, \neg Q_{2,n-1}, \neg P_{1,n-1}\}$. Different choices for these two resolutions would have led to eight clauses in all (because we have a choice which literal to resolve on).

As the number of resolutions increases, the number of clauses generated increases exponentially. For S_n^3, the graph is narrower, but one can still get exponentially many such proofs by backward chaining.

We use these clause sets to show exponential behavior of some of the strategies. The strategies that have exponential behavior are often back chaining strategies that are similar to Prolog (SLD resolution [Llo87]) in their execution; thus, a clause $\{P, \neg Q, \neg R\}$ can be viewed as a Prolog clause $P\ :-\ Q, R$, which means that if P is a goal, then Q and R become subgoals that are solved in turn. An SLD-resolution between this clause and some all-negative clause C such that $\neg P \in C$ produces the clause $C - \{\neg P\} \cup \{\neg Q, \neg R\}$. This replaces the literal $\neg P$ by $\neg Q$ and $\neg R$, corresponding in Prolog terms to replacing the subgoal P by the subgoals Q and R. Initially, an all-negative clause is chosen to start the search. The literals of the all-negative clauses are considered as (sub)goals. The search proceeds by subgoaling.

We now indicate why these clauses sets are difficult for some strategies. In S_n^1, there are a large number of negative literals in the non-unit clause, and if an order for resolving them is not specified, many clauses can be generated with some of the negative literals deleted, since there are exponentially many orders for resolving the literals. This causes a problem for forward chaining methods that do not order the negative literals. In S_n^2, for each subgoal $P_{i,j}$ and $Q_{i,j}$, there is a choice of two clauses to resolve with it, each generating two more literals (subgoals). (This corresponds to the two ways of choosing P pebbles and Q pebbles.) These choices each generate more subgoals, each having two choices for a clause to solve it. Therefore, these choices can be made in many ways, generating many combinations of the $P_{i,j}$ and $Q_{i,j}$ for backward chaining methods. Also, for some methods, the same subgoal will be solved repeatedly. This set of clauses was chosen to neutralize the obvious methods of reducing the search space. The type 2 clauses were added so there would be no pure literals, to neutralize pure-literal-clause deletion. In S_n^3 and T_n^3, there are fewer clauses altogether and fewer subgoals at each level. The subgoals P_i and Q_i both depend on P_{i+1}, Q_{i+1}, P_{i+2}, and Q_{i+2}, for all $i < n-1$.

Let's adopt the Prolog subgoal calling formalism to describe the search space for all-negative resolution. In T_n^3, the top-level goal clause is $\{\neg P_0, \neg Q_0\}$. This corresponds to the two subgoals P_0 and Q_0. If we call subgoals in a depth-first manner, we will first attempt P_0, which will eventually succeed, and then we will call Q_0. During one attempt to solve P_0 (one resolution), the subgoals P_1 and P_2 will be generated, and in another attempt to solve P_0, the subgoals Q_1 and Q_2 will be generated. During one attempt to solve Q_0, the subgoals P_1 and Q_2 will be generated, and during the other attempt, the subgoals Q_1 and P_2 will be generated. This leads to a total of two occurrences of each of the subgoals P_1, P_2, Q_1, and Q_2. Each occurrence of P_1 and Q_1 will eventually be called, each call corresponds to two resolution operations, and each resolution generates two more subgoals. All subgoals attempted will eventually succeed, and later subgoals in the same clause will be called. Thus four more occurrences of the subgoals P_2 and Q_2 will be generated and eventually called. So the subgoals P_1 and Q_1 will both be solved twice, the subgoals P_2 and Q_2 will both be solved six times, et cetera, in an exponential sequence generated by a simple recurrence relation. Let $t(L)$ be the time required to generate a proof of a literal L using some strategy. With depth-first search as indicated here, we have $t(P_i) = 1 + t(P_{i+1}) + t(P_{i+2}) + t(Q_{i+1}) + t(Q_{i+2})$ and $t(Q_i) = 1 + t(P_{i+1}) + t(Q_{i+2}) + t(Q_{i+1}) + t(P_{i+2})$ for $i < 2n - 2$. The solution is exponential, and this leads to exponential behavior for most backward chaining methods, and often to exponential search depth.

In S_n^3 the behavior is a little better; all the attempts to solve a subgoal will fail and this will be detected within a linear number of rounds. Thus, for example, in the top-level clause $\{\neg P_0, \neg Q_0\}$, if the subgoal P_0 is attempted first, it will eventually fail and Q_0 will never be called. This considerably reduces the size of the search space. However, the duplication by combination will still be exponential, since there are an exponential number of paths of subgoal calls that are possible from each top-level goal. We believe that T_n^2 and T_n^3 will be quite a challenge for pure back-chaining methods without unit simplification, and S_n^2 and S_n^3 will be a challenge for pure back-chaining methods even with unit simplification. Of course, these clause sets can be solved fast by forward chaining methods.

Now, the strategies that have exponential behavior can often be made more efficient in simple ways, such as adding unit simplification. For example, T_n^2 and T_n^3 can be shown unsatisfiable in polynomial time in this way. However, unit simplification and subsumption do not help

S_n^2 and S_n^3 because there are no positive unit clauses in the input. Also, these examples could be made slightly more complicated or lifted to first order logic and would still reveal the same poor behavior, even with unit simplification. Later we give such simple modifications to these sets of clauses. We think it is most illuminating initially to give the simplest examples demonstrating bad behavior. Furthermore, we believe that the kinds of bad behavior illustrated here often occur in practice in the execution of theorem provers, but they are masked by the thousands and thousands of clauses generated. A straightforward implementation of various strategies can produce combinatorial problems of which the programmer is not aware. An awareness of these problems can lead to modest changes to the search procedure which can have dramatic (positive) effects on its performance.

We now discuss the strategies in turn, justifying the entries in the above chart. Often we identify a state with its label, and thus each state is considered as a set of elements of V, though this is not formally correct. First we consider hyper-resolution [Rob65a].

1.5.2 Hyper-resolution

Hyper-resolution is equivalent to a sequence of resolutions that eliminate all the negative literals in a clause, by resolving with clauses that are all-positive. The positive clauses are called *electrons* and the clause containing negative literals is called the *nucleus*. For example, by hyper-resolving the nucleus $\{\neg P, \neg Q, R\}$ and the two electrons P and Q we obtain the hyper-resolvent R. Now R (really $\{R\}$) subsumes the original nucleus $\{\neg P, \neg Q, R\}$. For theorem proving purposes, subsumed clauses can be deleted, so that we delete the original nucleus and only retain the simpler clause R. We assume that such subsumed clauses are deleted. In general, for propositional Horn clauses, each hyper-resolvent is a positive unit clause that subsumes the parent (nucleus) clause, causing the parent to be deleted. Therefore, each round of hyper-resolution reduces the size of the clause set. Therefore, after a linear number of rounds, either a proof is found or the search stops (for Horn sets). This means that the search depth and the duplication by iteration are linear. Also, there is constant duplication by case analysis (since this method is linear in our formulation) and linear duplication by combination (since literals from different clauses are never combined, and the number of clauses in each state is no greater than that in the previous state). Also, the same analysis applies to hyper-resolution enhanced with any literal ordering method (ordering of the positive literals), since in the Horn case each clause has at most one positive literal.

1.5.3 P_1 deduction

P_1-deduction is the strategy that resolves two clauses only if one of them is positive [Rob65a]. For Horn clauses, as in hyper-resolution, this produces a clause that subsumes its parent. For example, if we resolve P (which is positive) and $\{\neg P, \neg Q, R\}$ we obtain $\{\neg Q, R\}$, which subsumes $\{\neg P, \neg Q, R\}$. Thus the parent clause $\{\neg P, \neg Q, R\}$ can be deleted. In this way, if subsumption (at least parent subsumption) is tested after each resolution, then each P_1 resolution reduces the size of the set of clauses, so that the search depth and the duplication by iteration and combination are always linear.

Assuming that subsumption is only tested after each round of resolution, it is possible to obtain additional resolvents. We could have resolved Q with $\{\neg P, \neg Q, R\}$ to obtain $\{\neg P, R\}$, for example. In this way, we can obtain resolvents containing arbitrary subsets of the negative literals of a clause. Because any subset of the n subgoals (negative literals) can be generated, there are 2^n clauses that can be generated. Subsumption testing after each round will reduce this number to some extent, since sets need not be retained if proper subsets have been generated. To analyze this, we consider the order in which the clauses are generated. P_1 deduction on S_n^1 will first generate all subsets of size $n-1$, then all subsets of size $n-2$, deleting those of size $n-1$ by subsumption, then all subsets of size $n-3$, deleting all those of size $n-2$ by subsumption, and so on. Also, given two distinct ground clauses of size k, neither can subsume the other. Therefore, the number of clauses generated is at least the maximum over k of the number of subsets of n elements of size k. This implies that there will be at least $2^n/(n+1)$ clauses generated such that no such clause subsumes any other. This bound is actually not as good as possible, but it is still exponential. We can derive this bound by noting that for some k, the number of subsets of size k must be at least $2^n/(n+1)$, since there are at most $n+1$ sizes and 2^n subsets altogether. However, the search depth and the duplication by iteration are linear, and there is a constant duplication by case analysis.

If the input clauses are restricted to have 3 literals, then there are at most two subgoals per clause, and at most 4 subsets of these exist. Thus the duplication by combination is linear. Another refinement is to specify a total ordering on the negative literals of a clause. Only the negative literal that is smallest in this ordering can be resolved on. This is still complete. If an ordering on predicate symbols is specified, then the subgoals will be solved in order, reducing the number of subsets of the subgoals generated to a linear amount. This indicates the potential importance of limiting the number of literals per clause (which can be done by introducing new predicate symbols), and ordering the predicate symbols.

1.5.4 A-ordering

Resolution with A-ordering [Sla67] is the strategy in which an ordering is specified on predicate symbols, and literals with predicates that are maximal in the ordering, are resolved on first. For example, suppose we have clauses $\{\neg P, \neg Q, R\}$ and $\{\neg R, T\}$. Suppose $R > P$, $R > Q$, $R > T$ where $>$ is the ordering on predicates. Then R is the predicate that will be resolved on, and we can resolve these two clauses to produce the clause $\{\neg P, \neg Q, T\}$. We assume that the ordering used is a total ordering, although one could just as well define A-ordering with a partial ordering on the predicate symbols. We note that A-ordering proofs are regular, in the sense of [Tse68], so that the proof length is exponential for all clause sets for which regular resolution has exponential length proofs. But this does not settle its behavior on Horn sets. A-ordering does not behave exactly like P_1-deduction or like all-negative resolution with ordering. One would think that by choosing a suitable ordering we could simulate these, but it does not appear in general to be possible. We would have to make the unit clauses maximal in the ordering to simulate P_1 deduction, but because these unit clauses may also appear elsewhere, some non-P_1 deductions may occur. For example, we could have a clause set containing, among other clauses,

$\{\neg Q, P\}$, $\{P\}$, and $\{\neg P, R\}$. If we make P maximal in the ordering to simulate P_1 deduction, there will also be an unwanted resolution on the occurrence of the literal P in the clause $\{\neg Q, P\}$. However, we note that the clause $\{P\}$ subsumes $\{\neg Q, P\}$, so if subsumed clauses are deleted, the problem disappears. There is another problem, however, that prevents the simulation of P_1-deduction in some cases; we will present this additional problem later in section 1.5.10.

It is conceptually simpler to think of A-ordering with the search reordered a little: Suppose the predicate symbols are ordered $P_1 > P_2 > \ldots > P_n$. Then in the first round, all A-ordering resolutions on the literal P_1 are done, and in the second round, all A-ordering resolutions on the literal P_2 are done, and so on. Let us call this *uniform* A-ordering, in contrast to the usual search method in which all possible A-ordering resolutions are done at each round; we call the latter *breadth-first* A-ordering search. The uniform version is still a complete theorem proving strategy for propositional clauses, even non-Horn clauses. This uniform search method explores essentially the same search space as breadth-first A-ordering, but it can result in a search space larger or smaller than breadth-first search due to the different subsumption deletions that can occur. We can express one relationship between the two methods of search as follows:

Theorem 1.5.1 *Suppose S is a set of clauses and uniform A-ordering without subsumption deletion generates a search space of size s_1 from S. Suppose breadth-first A-ordering with subsumption deletion generates a search space of size s_2. Then $s_2 \leq s_1$.*

Proof. Breadth-first A-ordering does the same resolutions as uniform A-ordering, but some of them occur in earlier rounds. Therefore breadth-first A-ordering without subsumption deletion generates the same search space as uniform A-ordering without subsumption deletion. It follows that breadth-first A-ordering with subsumption deletion generates a search space that is the same size or possibly smaller.

□

It is clear that uniform search will stop after a linear number of rounds (when all the predicate symbols have been eliminated). Therefore the search depth and the duplication by iteration for uniform A-ordering are always linear, regardless of the ordering. After all predicates have been resolved on, the search stops. This holds even for non-Horn clauses, by the way. This shows that uniform A-ordering has a definite global notion of progress (elimination of predicates from the set of clauses). Because of the similarity of uniform A-ordering to the usual breadth-first A-ordering, it turns out that breadth-first A-ordering also has linear search depth and duplication by iteration, and a definite notion of progress. However, we give a formal proof of this as follows:

Theorem 1.5.2 *The search depth and duplication by iteration for A-ordering is linear, in fact, bounded by the number of predicates in S.*

Proof. Suppose C is an A-ordering resolvent of C_1 and C_2. Then the maximum predicate symbol in C is smaller than the maximum predicate symbols in C_1 and C_2. Then if C resolves against some other clause D, the resolvent of C and D will have a maximum predicate symbol that is yet smaller. A simple induction shows that no new resolvents can be produced beyond a depth of search equal to the number of predicate symbols.

□

For Horn sets, this is actually not better than the situation for all-negative resolution without ordering, where the search depth is also linear. But it is better than all-negative resolution with ordering, which can produce an exponential search depth for a bad ordering. However, duplication by combination for A-ordering can still be exponential, as shown by S_n^2. If we choose the A-ordering in S_n^2 so that the predicates $P_{i,j}$ are ordered by $j-i$, that is, $P_{i,j} > P_{k,l}$ if $j-i > l-k$, then exponentially many combinations of literals are generated. This is so because whenever we resolve on a literal $P_{i,j}$ we have two clauses to choose from, each generating a different combination of literals. The same is true for $Q_{i,j}$. Each such resolution produces literals whose $|j-i|$ value is one less. Thus it takes n stages until all such resolutions are exhausted, and a number of combinations exponential in n is generated. Also, A-ordering is not goal-sensitive. To verify this, it is only necessary to choose a set of Horn clauses with one goal clause $\neg P$ that is a negative unit clause, and to order the predicate symbols so that P is smallest. Then the goal clause $\neg P$ will not participate in any resolutions until the very end. One might ask whether we can make the A-ordering strategy goal sensitive by a proper choice of ordering. This is not always possible; if the goal is $\neg P, \neg Q$ then it can happen that P also appears in other clauses. If P is made maximal in the ordering, then resolutions involving P may occur that are not goal-sensitive (say, a resolution with the clause $\neg P, R$). However, later on we give special cases for which an ordering can be found that is goal-sensitive. Another question is whether there is always a good ordering for every set S of clauses, that is, an ordering that will produce polynomial behavior for A-ordering. Later we show that this is not always possible, but give some special cases where such a good ordering always exists.

1.5.5 Implications for term rewriting

Note that the first-order strategies based on term-rewriting techniques [HR91, BG90] generally reduce to A-ordering methods on clauses without equality. This shows that these methods also sometimes suffer from exponential search inefficiency and often lack goal sensitivity. Nor do they have smaller search depth than all-negative resolution. However, there is some advantage for A-ordering strategies over all-negative resolution in this framework, and that is that regardless of the ordering used, the search depth is still linear. (This holds even for non-Horn sets.) Also, term-rewriting methods are often very efficient on pure equality problems. An advantage of all-negative resolution is that it is goal-sensitive. Because of its importance for term-rewriting, we explore the behavior of A-ordering further, after introducing proof dags. This will reveal some additional advantages (and disadvantages) of resolution with A-ordering. It seems that methods based on conditional rewriting [ZK88] may order the search differently and may have different search behavior.

1.5.6 Proof dags

To facilitate the analysis of the remaining strategies, and even to clarify the analysis of the preceding ones, we introduce the concept of a *proof dag* (directed acyclic graph). This illustrates the dependencies between the literals in a minimal unsatisfiable set of Horn clauses. Let S be a minimal unsatisfiable set of Horn clauses. The proof dag $D(S)$ of S has as vertices the predicates appearing in S. Since S is minimal unsatisfiable, for every positive literal P in S there is a unique clause C in S containing P positively, and there will be one or more clauses containing $\neg P$. Suppose C is $P_1 \wedge P_2 \wedge \ldots \wedge P_n \supset P$. Then the proof dag $D(S)$ has edges from P_i to P for all such i. Given a predicate P in S, define its subgraph $D(P)$ to be the vertices v in $D(S)$ from which there is a path to P, together with edges between such vertices. These are the predicates that contribute to a proof of P and their dependencies. Note that a clause is a set of literals, and this can often be associated with a set of vertices of the proof dag. This can in turn be regarded as a pebbling of the graph, that is, we can think of some of the vertices as having pebbles on them. The inference procedure on a clause will often correspond to natural ways of moving these pebbles around the graph. This helps to explain and to understand the performance of the various strategies. For all-negative resolution, we can think of a clause C as a pebbling in which the predicates appearing in C are pebbled. Thus if C is $\{\neg P, \neg Q\}$, then the pebbling corresponding to C would include the vertices P and Q in the proof dag. An all-negative resolution corresponds to the removal of a pebble from a vertex and the addition of pebbles to its predecessors. For example, a resolution of an all-negative clause $\{\neg P, \neg Q\}$ with the clause $\{\neg P_1, \neg P_2, P\}$ corresponds to a removal of a pebble from P and the addition of pebbles to the predecessor vertices P_1 and P_2. For hyper-resolution, each resolvent is a positive unit clause, and we place a pebble on each vertex P for which the corresponding positive unit clause P has been derived. Though formally adding nothing new, this terminology sometimes helps to clarify the underlying ideas.

Define the proof complexity $cp(P)$ to be the number of vertices in $D(P)$. For example, suppose S is the set of clauses containing $\{P\}$, $\{\neg P, Q\}$, and $\{\neg Q\}$. Then the proof dag $D(S)$ has two vertices P and

Q and an edge from P to Q. Also, $cp(P)$ is 1 and $cp(Q)$ is 2. Now, if $Q_1, \ldots, Q_n$ are the vertices of $D(S)$ with no outgoing edges, then these are goal vertices and there must be a goal clause $\{\neg Q_1, \ldots, \neg Q_n\}$ in S. In our example, Q is a goal vertex. Also, a vertex with no incoming edges is a fact (a positive unit clause), like P in our example. It is convenient to use proof dags because it is often convenient to specify orderings on predicate symbols in terms of their structure and thereby derive bounds on the performance of resolution strategies. For example, if we choose an A-ordering in which literals with small cp values are resolved on first, then we can cause A-ordering to simulate forward reasoning, that is, P_1 deduction with ordering of negative literals. This yields polynomial behavior on minimal unsatisfiable sets of Horn clauses.

1.5.7 Other properties of clause sets

Proof dags are only defined for minimal unsatisfiable clause sets. However, we would like to use the machinery of proof dags even on satisfiable sets of clauses. We can do this as follows. We say that a set S of Horn clauses is *well-ordered* if there is a partial ordering $<$ on the predicate symbols such that if $P\ :-\ P_1, \ldots, P_n$ is a clause in S then $P_i < P$ for all i. Note that minimal unsatisfiable Horn sets are well-ordered. We call the minimal such ordering the *well-ordering* of the predicate symbols. We say that a set S of Horn clauses is *deterministic* if for every predicate symbol P there is at most one clause C in S such that P appears positively in C, that is, $P \in C$. We note that minimal unsatisfiable Horn sets are both well-ordered and deterministic. Also, many of the results that are stated for minimal unsatisfiable clause sets apply equally well to well-ordered, deterministic clause sets. In order to apply well-orderings to A-ordering and all-negative resolution, it is convenient to extend these orderings to total orderings on S, and we typically assume that this is done in some manner, especially for A-ordering.

1.5.8 All-negative resolution

All-negative resolution is like P_1-deduction with signs reversed: One of the parent clauses in a resolution must be all-negative. As explained earlier, this strategy does poorly on S_n^2 and S_n^3, generating exponentially many combinations of subgoals. However, the search depth for all-negative resolution is still linear, and there is no (i.e., constant) duplication by case analysis. To see that the search depth is linear, suppose that S is minimal unsatisfiable and consider a proof dag $D(S)$ for S. Each resolution involves a literal $\neg P$ from an all-negative clause C and a literal P from another clause D. Now, the effect of the resolution is to replace the literal $\neg P$ in C by the other literals in D. However, these other literals $\neg Q$ in D will satisfy $cp(Q) < cp(P)$. Therefore, if one always chooses the literal $\neg P$ such that $cp(P)$ is maximal, each resolution will reduce the maximum $cp(P)$ value of predicates P in the clause, and a proof will be found after a linear number of resolutions. Or it may be necessary to perform a sequence of resolutions to effect this, if there are more than one negative literal with the same maximum cp value. If S is unsatisfiable but not minimal unsatisfiable, this reasoning can still be applied to a minimal unsatisfiable subset of S. If S is satisfiable, then we can still show that the duplication by iteration is linear, but the argument is a little more complicated, as follows.

Theorem 1.5.3 *Suppose S is a propositional Horn set containing n different predicate symbols. Let C be some clause generated from S by all-negative resolution. Then there is some clause C' generated from S by not more than n all-negative resolutions such that C' is a subset of C.*

Proof. Suppose $C_1, C_2, ..., C_p$ is a minimal-length all-negative resolution proof of C. This means that C_p is C and every clause in the sequence is either in S or is an all-negative resolvent of two earlier clauses in the sequence. It is not hard to see that in this proof, at most one all-negative input clause is involved. Suppose without loss of generality that this clause is C_1. Now, from this proof, we construct a proof $D_1 D_2 ... D_k$ of length at most n of a clause C' such that C' subsumes C. We choose D_1 to be C_1. For each literal L that does not appear in C, let $last(L)$ be the maximum i such that $L \in C_i$. We obtain D_{i+1} from D_i by applying the following rule: Pick a literal $\neg P$ of D_i to resolve such that $last(\neg P)$ is as small as possible. Let j be $last(\neg P)$. This literal was removed from C_j by resolving with some clause D of S containing P to give C_{j+1}. We resolve D_i with D on the literals P and $\neg P$ to obtain D_{i+1}. This has the effect of replacing the literal $\neg P$ of D_i with other literals of D. We note that these other literals of D appear in C_{j+1}. Therefore their last appearance in the proof of C is later than the last appearance of $\neg P$. Each step of this proof increases the minimum value of $last(L)$ for literals L in the clause C_i. This means that the literal $\neg P$ will never be reintroduced into a clause D_m for $m > j$. Therefore, after n such resolutions, all literals that can be resolved on will have been, so that there are no resolutions left to do. This implies that a clause C_j has been derived containing only literals of C, so C_j subsumes C, and we can choose C' to be C_j.

□

Corollary 1.5.4 *After n rounds of all-negative resolution, all such C' will be generated, and so every clause C that can be generated by all-negative resolution will be subsumed by an already-generated clause. Then the search will stop, assuming that subsumed clauses are deleted. Thus the duplication by iteration is linear for all-negative resolution.*

1.5.9 All-negative resolution with ordering

We can specify an ordering on the predicate symbols for all-negative resolution. This means that in an all-negative clause C, the predicate symbol P of C that is maximal in the ordering is the only one that is resolved on. Typically a total ordering is used, but one can just as well use a partial ordering. One would expect that this use of an ordering would improve the behavior of the strategy, since fewer resolutions are possible. However, if an ordering on predicate symbols is specified, it can actually make the behavior much worse. It is only necessary to order the predicates so that the predicates P with smaller values of $cp(P)$ are resolved on first. This corresponds to moving the pebbles first that are farthest from the goal clause. For example, on T_n^3, this can cause each subgoal to be completely solved before working on the others, if we order the predicate symbols so that the P_j and Q_j with high j are resolved first. This can lead to exponential search depth (and duplication by iteration), and still allows exponential duplication by combination on T_n^2 and T_n^3. We give an example of a sequence of moves in a pebbling for T_n^3. We note that this only gives a portion of the search, since other clauses also can be used. First we remove a pebble from P_0 and place pebbles on P_1 and P_2, say. This corresponds to deriving the clause $\{\neg P_1, \neg P_2\}$. (We could also have chosen Q_1 and Q_2.) Then we remove the pebble from P_2, and place pebbles on P_3 and P_4. Then we remove the pebble from P_4 and place pebbles on P_5 and P_6. Eventually we solve P_6, removing the pebble from P_6 and all lower pebbles. We then remove the pebble from P_5 and add pebbles on P_6 and P_7. Eventually P_6 and P_7 are solved. Then pebbles are left on Q_0, P_1, and P_3. Next we remove the pebble from P_3 and place pebbles on P_4 and P_5, and so on. The fact that the lowest pebble is always moved means that subgoals are solved repeatedly.

A good ordering can lead to a linear search depth. We obtain a good ordering by resolving first on the predicate symbols P with a high value of $cp(P)$. For example, on S_n^3 and T_n^3, this means that we resolve the P_j and Q_j with low j first. Then we obtain linear search depth and polynomial behavior. For S_n^3 and T_n^3 this corresponds to a pebbling as follows: The pebble on P_0 is removed and pebbles are placed on P_1 and P_2 (say). Then the pebble on Q_0 is removed and replaced by pebbles on P_1 and Q_2, say. But there is only one pebble kept on P_1 even though it has been pebbled twice. Then the pebble is removed from P_1 and pebbles are added to P_2 and P_3. Then the pebble is removed from P_2 and replaced by pebbles on P_3 and P_4; at this stage there are pebbles on Q_2, P_3 and P_4. In a small number of steps we will reach the bottom and the search will end.

In general, if S is deterministic and well-ordered, and we use the well-ordering $<$ on S, then all-negative resolution with ordering may not have polynomial behavior. The reason for this is that the ordering $<$ may not be total, so an all-negative clause may have many maximal literals that are not ordered with respect to each other. Therefore many resolutions may be possible, an exponential number of clauses may be generated, and the duplication by combination may be exponential. However, if we extend the well-ordering $<$ to a total ordering $<'$, then the behavior will be polynomial for deterministic, well-ordered clause sets. This is because every resolution will replace a maximal literal

$\neg P$ in an all-negative clause by other literals $\neg Q$ such that $Q <' P$, and therefore the maximal negative literal in an all-negative clause will decrease in the ordering with every resolution operation. Note that this result applies to minimal unsatisfiable Horn sets, too, since they are deterministic and well-ordered.

However, a good (even total) ordering cannot always reduce the duplication by combination to a polynomial amount, if S is not deterministic. On S_n^2, there is exponential duplication by combination regardless of the ordering on predicate symbols chosen, but the proof is somewhat subtle. The reason for this is that we have to consider all possible orderings of predicate symbols and show that for all of them, the duplication by combination is exponential.

Theorem 1.5.5 *All-negative resolution with an ordering on the negative literals produces an exponential search space on S_n^2, regardless of the ordering used.*

Proof. We construct a set E of 2^{n-1} interpretations I of the set $\{P_{i,j},\ Q_{i,j} : 1 \leq i \leq j \leq n\}$. We consider an interpretation as a function from predicate symbols to truth values, such that I assigns P a value of *True* iff $I \models P$. For an arbitrary ordering $>$ on the predicate symbols, we show that there exists a set of 2^{n-1} *critical clauses* C_I, one for each interpretation $I \in E$, such that all the critical clauses will be generated by all-negative resolution with the ordering $>$. The set E of interpretations is defined as the set of interpretations that are models of the following set of formulae:

$$P_{1,n}$$

$$\neg(P_{i,j} \equiv Q_{i,j}), 1 \leq i \leq j \leq n$$

$$P_{i,j} \equiv (P_{i+1,j} \equiv P_{i,j-1}), 1 \leq i < j \leq n$$

These define $P_{i,j}$ and $Q_{i,j}$ in terms of literals with smaller values of $j - i$. The entire interpretation is therefore determined by the assignments to $P_{i,i}$ and $Q_{i,i}$. However, $Q_{i,j}$ is defined in terms of $P_{i,j}$, hence the entire interpretation is defined by the 2^n possible assignments to $P_{i,i}$ for $1 \leq i \leq n$. The formula $P_{1,n}$ also constrains these interpretations. However, one can show by a simple induction that if I and J

are two interpretations agreeing on the literals of form $P_{i,i}$ for $i > 1$ but disagreeing in their assignment to P_1, then I and J will assign different truth values to $P_{1,n}$. Thus half of the 2^n interpretations of the $P_{i,i}$ will result in $P_{1,n}$ being assigned true, so there are a total of 2^{n-1} interpretations in E.

Define the *weight* of a literal $P_{i,j}$ or $Q_{i,j}$ to be $j - i$, and the weight of $\neg L$ to be equal to the weight of L. Let $w(L)$ be the weight of L. Define the weight w(C) of a clause C to be $\Sigma_{L \in C} 3^{w(L)}$. We note that the type 1 clauses are all of the form $L \ :- \ M_1, M_2$ where $w(M_1) = w(M_2) < w(L)$. Thus, each all-negative resolution replaces a literal of weight w with two literals of weight $w - 1$. Now, $3^{w-1} + 3^{w-1} < 3^w$. It follows that each all-negative resolution produces a clause smaller than its all-negative parent in the weight ordering. Eventually a clause will be produced with two literals of weight zero. Call such a clause a *critical clause.* We show that for each interpretation $I \in E$, a critical clause C_I will be generated, all of whose literals are false in I. Also, we show that no two such critical clauses are identical, and none of them will be supersets of other clauses generated. This means that none of these clauses will be deleted by subsumption deletion. Therefore there will be at least 2^{n-1} clauses generated, regardless of the choice of the ordering.

We now show that if C is an all-negative clause and I is in E and C is false in I (that is, $(I \not\models C)$), and L is a literal in C, and $w(L) > 0$, then there is an input clause C' and a resolvent D of C and C' such that D is false in I and $w(D) < w(C)$. Note that this result is independent of which literal L is chosen, so it holds for an arbitrary ordering of literals. Suppose L is $\neg P_{i,j}$ for some i and j. Let D_1 be $(C - \{L\}) \cup \{\neg P_{i+1,j}, \neg P_{i,j-1}\}$ and let D_2 be $(C - \{L\}) \cup \{\neg Q_{i+1,j}, \neg Q_{i,j-1}\}$. These are the only two all-negative resolvents on L, using the structure of S_n^2. Since $(I \not\models L)$, we have that $I \models P_{i,j}$. Thus $I \models (P_{i+1,j} \equiv P_{i,j-1})$ since I was defined to satisfy the formulae $P_{i,j} \equiv (P_{i+1,j} \equiv P_{i,j-1}), 1 \le i < j \le n$. Therefore either $I \models P_{i+1,j} \wedge P_{i,j-1}$ or $I \models Q_{i+1,j} \wedge Q_{i,j-1}$. In the former case, we can let D be D_1 and in the latter case we can let D be D_2. The argument is similar if L is $\neg Q_{i,j}$.

For now, let's assume that the type 2 clauses are omitted. In the beginning, $\neg P_{1,n}$ is in S_n^2, and this clause is false in all $I \in E$. It follows that all-negative resolution will generate clauses C of smaller and smaller weight, for all $I \in E$, such that $(I \not\models C)$. Eventually, for all $I \in E$, there will be a critical clause C_I that is false in I. We need to show that none of these clauses will be identical. For this we consider S_n^2 as a graph, in the following way:

$$\begin{tikzcd}
 & & \begin{matrix}P_{1,n}\\ Q_{1,n}\end{matrix} \arrow[dl] \arrow[dr] & & \\
 & \begin{matrix}P_{1,n-1}\\ Q_{1,n-1}\end{matrix} \arrow[dl] \arrow[dr] & & \begin{matrix}P_{2,n}\\ Q_{2,n}\end{matrix} \arrow[dl] \arrow[dr] & \\
\begin{matrix}P_{1,n-2}\\ Q_{1,n-2}\end{matrix} & & \begin{matrix}P_{2,n-1}\\ Q_{2,n-1}\end{matrix} & & \begin{matrix}P_{3,n}\\ Q_{3,n}\end{matrix} \\
 & \cdots & & \cdots &
\end{tikzcd}$$

The nodes of this graph are ordered pairs of integers called *positions*, and the edges are the arrows. Each all-negative resolution replaces a literal by a literal in a position immediately below it. We call the $i+1, j$ and $i, j-1$ positions the *children* of the i, j position. The weight of a position i, j is $j - i$. Define a *complete path* in this graph structure to be a path starting at the root (the $1, n$ position) and extending down to some i, i position following the arrows; there will be exponentially many such paths. Note that each all-negative resolvent will contain a literal on each such path, since each all-negative resolution replaces a literal at the i, j position with literals at the $i+1, j$ and $i, j-1$ positions (its children), and any path that passes through the i, j position must also pass through either the $i+1, j$ position or the $i, j-1$ position. Therefore, the critical clauses will also have literals on all such paths.

Finally, we show that there is a function from critical clauses that maps C_I onto I. This implies that if I differs from J then C_I is different from C_J as desired. This is given by a kind of geometric argument. Namely, suppose that I is in E. Suppose that two of the three assignments of I for the literals $P_{i,j}$, $P_{i+1,j}$, and $P_{i,j-1}$ are given. We claim that the third is uniquely determined. We call this the *triangle property.* A simple way to see this is that an odd number of the statements $I \models P_{i,j}$, $I \models P_{i+1,j}$, and $I \models P_{i,j-1}$ must be true, by the way E is defined, since all interpretations in E satisfy the formulae $P_{i,j} \equiv (P_{i+1,j} \equiv P_{i,j-1}), 1 \leq i < j \leq n$. In general, we say that a position (a, b) is *determined* by a set $(a_1, b_1) \ldots, (a_n, b_n)$ of positions if for all pairs I, J of elements of E, $((I \models P_{a_1,b_1}) \equiv (J \models P_{a_1,b_1})) \wedge \ldots \wedge ((I \models P_{a_n,b_n}) \equiv (J \models P_{a_n,b_n}))$ implies $((I \models P_{a,b}) \equiv (J \models P_{a,b}))$. We say a clause C determines a position (a, b) if the positions of the literals in C determine (a, b).

Lemma 1.5.6 *If C is an all-negative clause generated from S_n^2 by all-negative resolution not using the type 2 clauses, then for every literal L in C there is a path from the root to L such that every position on this path is determined by C.*

Proof. To begin with, $\neg P_{1,n}$ is the only all-negative clause, and the lemma holds for this clause. Assume by induction that a negative clause C is generated by all-negative resolution and the lemma holds for it. Let D be a clause generated by one all-negative resolution from C. Then some literal L of C is replaced by literals L_1, L_2 at the two children positions, to obtain D. Now, the positions of L_i are in D, hence they are determined by D. Therefore, by the triangle property, the L position is also determined by D. Since the other literals of C are in D, D determines the positions of all the literals of C. By induction, C determines paths to all its literals. Hence D determines paths to all the literals of C. D has the two new literals L_i. We obtain paths to these literals by adding their positions to the end of the paths to L.

□

Lemma 1.5.7 *A critical clause determines every position.*

Proof. Suppose C is a critical clause. Then C has two literals L_1, L_2 of weight zero. By the preceding lemma, C determines paths L_1 and L_2. We show that these paths determine all the other positions. Suppose to the contrary that some position is not determined by these paths. We say two positions p and q are *neighbors* if there is some position having both p and q as children. Let w be the largest weight of an undetermined position. Now, the paths to L_1 and L_2 are complete paths and so will contain some position of weight w. So there must be some undetermined position p_1 which has a neighbor p_2 of weight w such that p_2 is determined. Thus p_1 and p_2 are children of some other position p. Now, p is determined since $w(p) < w$. By the triangle property, p_1 is also determined, since p_2 is. This contradicts our inference that p_1 is undetermined. Therefore we conclude that all positions are determined by C.

□

Corollary 1.5.8 *The critical clauses C_I and C_J for I different from J are different, hence there are 2^{n-1} critical clauses and the search space is exponential.*

Proof. Suppose C_I and C_J are critical clauses for I different from J. Both C_I and C_J determine all positions. Hence they determine the positions of weight 0. Hence they determine different truth values for some position of weight 0, since I and J are different. Hence C_I and C_J are different. Since E has 2^{n-1} elements, there are 2^{n-1} critical clauses generated at some time during the search, and all-negative resolution with ordering on S_n^2 has exponential behavior, regardless of the ordering chosen.

□

The preceding discussion has not considered deletion of subsumed clauses. For this we note that if C subsumes D and C and D are different then $w(C) < w(D)$. For each interpretation $I \in E$, we consider the clause C of minimal weight such that $I \not\models C$. Each round of all-negative resolution will reduce the weight of such a clause, until a critical clause is produced. Subsumption will not affect this, since if $I \not\models C$ and D subsumes C then $I \not\models D$. Therefore such minimal clauses will not be deleted by subsumption.

We now consider the type 2 clauses, though it is not strictly necessary to do so to demonstrate the exponential behavior of ordered all-negative resolution. A simple way to consider the type 2 clauses is just to restrict attention to the top part of the graph, that is, the clauses all of whose vertices are of weight larger than $n/2$. These clauses are unaffected by the type 2 clauses or their resolvents. The structure of this top part of the graph is similar to that of the whole graph, and the same arguments can be applied to it.

$\square$

We note that this exponential bound applies also to all-negative resolution without an ordering on negative literals, and provides a rigorous proof for that case. Many of our other exponential lower bounds are based on this one, so we have also established them. We recall that S_n^3 has polynomial behavior for all-negative resolution with a good ordering. This shows a significant difference between S_n^2 and S_n^3, since even with a good ordering we obtain exponential behavior for all-negative resolution on S_n^2.

We now state these results in a slightly different way, which is closer to the search space formalism of [KBL93].

Definition 1.5.9 *A* proof path from S *is a sequence $C_1C_2C_3\ldots$ of clauses such that each C_i is either in S or is a resolvent of previous clauses C_j and C_k, with $j, k < i$. We also require that if $i \neq j$ then $C_i \neq C_j$. A proof path is* maximal *if it cannot be extended, that is, there is no other proof path having it as a proper prefix.*

The question we would like to address is how long these proof paths are. For this we assume that the resolution strategy used is all-negative resolution with an ordering on the negative literals, and that subsumed clauses are deleted. This means that every clause C_k must have two parent clauses C_i and C_j such that neither one is properly subsumed by another clause among the first $k-1$ clauses in the sequence.

Theorem 1.5.10 *Let $C_1C_2C_3\ldots$ be a maximal proof path from S_n^2, in which the resolution strategy is all-negative resolution with an ordering on the negative literals. Then the length of this path is at least $2^{n/2}$, regardless of the ordering used.*

Proof. The proof is essentially the same as that given above. Namely, we construct the same set E of interpretations and show that for each one a distinct critical clause will eventually be produced.

$\square$

This result was first presented in [Pla94d]. We think that this latter version of the result is more striking, because the simplicity of the formulation eliminates the need to describe the search space formally. Note that we are considering all possible orderings and also all possible choices of sequences of resolution operations, and that for all these possibilities the search is still exponential. The question then arises, can such behavior also be produced for unsatisfiable clause sets? What if breadth-first search is specified? We do not have the answers at present.

Returning to our usual search space formalism, we do not know whether a good ordering can always lead to linear duplication by iteration for all-negative resolution with ordering. If S is deterministic and well-ordered, then totally ordering the literals according to proof complexity will cause all-negative resolution with ordering to have polynomial behavior (and therefore polynomial duplication by iteration). The problematic case is satisfiable clause sets that are not deterministic or not well-ordered. The reason that the proof of theorem 1.5.3 for the unordered case does not work is that it imposes an ordering that depends on the proof being considered. For ordered resolution, the order has to be global. It's interesting to see that for the set S_n^3, using unrestricted ordering and just limiting the size of the resolvents to four or fewer literals will lead to good behavior, though this is not a complete restriction in general. Another interesting open problem is whether all-negative resolution with a good ordering has exponential behavior on unsatisfiable Horn sets. We only showed this for S_n^2, which is satisfiable. We show later that A-ordering with a good ordering has exponential behavior on (some) satisfiable clause sets. These results indicate that sometimes even a good ordering cannot help the ordering strategies to perform well on easy problems.

1.5.10 A-ordering and proof dags

We now return to develop additional properties of A-orderings in relation to proof dags and minimal unsatisfiable sets of clauses. This will reveal some additional advantages of A-orderings and perhaps help to explain the success of term-rewriting based methods. On the other hand, the weaknesses of this strategy will also be more clearly delineated.

First, we note that if S is a minimal unsatisfiable Horn set, and if we use resolution with A-ordering where the ordering on literals L is according to their proof complexity $cp(L)$, that is, literals with larger proof complexity are resolved away first, then A-ordering is goal-sensitive and has polynomial behavior. For this we assume that the proof complexity ordering is extended to a total ordering in some way. The goal-sensitivity follows because A-ordering mimics all-negative resolution in this case and the polynomial behavior follows from the polynomial behavior of all-negative resolution on minimal unsatisfiable clause sets using the proof complexity ordering. This suggests that A-ordering is good when there are no irrelevant clauses. In fact, we can say even more: If S is a minimal unsatisfiable set of clauses, then A-ordering with an *arbitrary* literal ordering has polynomial behavior. This is easiest to see for uniform A-ordering resolution: The number of clauses that can participate in future resolutions never grows; clauses containing eliminated predicate symbols cannot again produce new resolvents. (A predicate symbol is considered as eliminated when it is the largest literal of resolution in a round.) The reason that the number of clauses does not grow, is that minimal unsatisfiable clause sets are deterministic, so that each negative literal can be eliminated from a clause in exactly one way. If we could resolve a negative literal $\neg P$ against two other clauses, then there would have to be two clauses containing P, and so S would not be deterministic; this property is preserved among the clauses that can participate in future resolutions. Also, the number of literals in each clause is bounded by the number of predicate symbols in S altogether. Finally, after all the predicate symbols have been eliminated, the search will stop. This result holds even if no subsumption deletion is done; therefore it follows by theorem 1.5.1 that the search space for breadth-first A-ordering is also polynomial, regardless of the ordering.

This is a positive result, indicating that if there are no irrelevant clauses then A-ordering performs well, regardless of the ordering. The behavior is even better (at least, goal-sensitive) if we choose the ordering so that literals with high proof complexity resolve first. Unfortunately, it does not always suffice to use the proof complexity ordering in this way, even for unsatisfiable clause sets. Consider the set T_n^2; if A-ordering resolution is applied to this set with the proof complexity ordering, then the behavior is exponential, as noted before. There does not seem to be any natural way to overcome this problem, because the symmetries in T_n^2 make it hard to justify a preference for one of P_{ij} and Q_{ij} over the other one. This shows that if there are enough redundancies in the input, then it can be impossible to find a natural ordering that is efficient and goal-sensitive, for resolution with A-ordering.

Note that minimal unsatisfiable clause sets are deterministic. The above result concerning minimal unsatisfiable clause sete can easily be extended to *all* deterministic clause sets. That is to say, A-ordering on deterministic clause sets has polynomial behavior regardless of the ordering.

We can give another positive result for A-ordering, without goal-sensitivity. Recall that minimal unsatisfiable sets of clauses are well-ordered. We now consider well-ordered sets in general, and show that A-ordering with a suitable ordering has polynomial behavior. Suppose S is well-ordered, and suppose we perform resolution with A-ordering where the literal of resolution is chosen as the *smallest* literal in the well-ordering, that is, the literal farthest away from the goal (or goals). Note that if a positive literal in a clause C is minimal in the well-ordering, then there must not be any negative literals in C, so C must be a positive unit clause. This means that every A-ordering resolution will involve a positive unit clause and another clause, and so we will simulate P_1-deduction, which has polynomial behavior. This result, in contrast to that for P_1-deduction, does not require that parent clauses be subsumed after each resolution, since A-ordering will impose an ordering on the negative literals automatically. This result applies both to satisfiable and unsatisfiable clause sets.

From the above result, it follows that if there is a clause set S for which A-ordering (with a good ordering) has exponential behavior, then S must not be well-ordered. We now exhibit such a set of clauses. In particular, we show that the following set A_n of clauses will produce exponential behavior for A-ordering resolution, regardless of the ordering:

$$\begin{array}{lll}
P_{i,j} & :- \; P_{i,j+1}, P_{i+1,j+1}, & 0 \le i,j < n \\
P_{i,j} & :- \; P_{i,j+1}, Q_{i+1,j+1}, & 0 \le i,j < n \\
P_{i,j} & :- \; Q_{i,j+1}, P_{i+1,j+1}, & 0 \le i,j < n \\
P_{i,j} & :- \; Q_{i,j+1}, Q_{i+1,j+1}, & 0 \le i,j < n \\
Q_{i,j} & :- \; P_{i,j+1}, P_{i+1,j+1}, & 0 \le i,j < n \\
Q_{i,j} & :- \; P_{i,j+1}, Q_{i+1,j+1}, & 0 \le i,j < n \\
Q_{i,j} & :- \; Q_{i,j+1}, P_{i+1,j+1}, & 0 \le i,j < n \\
Q_{i,j} & :- \; Q_{i,j+1}, Q_{i+1,j+1}, & 0 \le i,j < n
\end{array}$$

Here we assume that $P_{i,n}$ and $P_{i,0}$ are identified, for all i, and that $P_{n,j}$ and $P_{0,j}$ are identified, for all j, and similarly for Q. Thus we have a kind of a "torus" structure. This set of clauses is not deterministic or well-ordered, and is trivial for forward and backward chaining strategies, due to the lack of positive and negative clauses. However, for the A-ordering strategy, larger and larger clauses will be generated, and the larger the clauses become, the more combinations of P and Q are possible. Therefore, there is exponential behavior for A-ordering, regardless of the ordering. This clause set should be an interesting set to test A-ordering based theorem provers on. The structure is reminiscent of S_n^2 in some ways. One could obtain unsatisfiable clause sets hard for the A-ordering strategy by considering $A_n \cup T_{2n}^2$ (with the predicate symbols in T_{2n}^2 renamed) or by adding some other unsatisfiable clause set with a sufficiently long proof to A_n. We now prove the exponential bound on search space size (duplication by combination).

Definition 1.5.11 *A* position *of A_n is an ordered pair (i, j) of integers from the set $\{0, 1, ..., n\}$, where the positions $(i, 0)$ and (i, n) are identified for all i, and the positions $(0, j)$ and (n, j) are also identified, for all j. We consider these positions as nodes of a graph; there is an edge from (i, j) to $(i, j + 1)$ and an edge from (i, j) to $(i + 1, j + 1)$ for all i and j such that these ordered pairs are valid positions. We call the node $(i, j + 1)$ the* left child *of (i, j) and we call $(i + 1, j + 1)$ the* right child *of node (i, j). A* path *is a sequence of positions $\alpha_1, \ldots, \alpha_k$ of A_n such that for all a, α_{a+1} is either a left or a right child of α_a; that is, there is an edge from α_a to α_{a+1}. The* length *of this path is k. The* distance *from position α to position β is the length of the shortest path from α to β. If α is (i, j) then P_α denotes $P_{i,j}$ and Q_α denotes $Q_{i,j}$.*

Definition 1.5.12 *If $\mathcal{P}$ is a set of positions $(i_1, j_1), \ldots, (i_k, j_k)$ of A_n then a* cluster *of clauses for $\mathcal{P}$ is the set of clauses of the form $\{R^1_{i_1,j_1}, \ldots, R^k_{i_k,j_k}\}$ where each R^a is either $\neg P$ or $\neg Q$, except for one of the positions (i_b, j_b), for which R^a is either P or Q. We call this position (i_b, j_b) the* distinguished position *of the cluster. Thus all the clauses in a cluster are Horn clauses, the positive literals all appear at the same position, and if $\mathcal{P}$ has k elements, a cluster for $\mathcal{P}$ consists of 2^k clauses. The* area *of a cluster for $\mathcal{P}$ is the number of elements of $\mathcal{P}$. The* cluster height *of a cluster $\mathcal{A}$ for $\mathcal{P}$ is the maximum distance (of length n or less) from the distinguished position α of $\mathcal{A}$ to some other position β of $\mathcal{A}$. Note that A_n is the union of n^2 clusters, all of area 3 and height 2.*

Definition 1.5.13 *A cluster $\mathcal{A}$ for set $\mathcal{P}$ of positions has the* path property *if for every path of length n or less from the distinguished position α of $\mathcal{P}$ there is a non-distinguished position β in $\mathcal{P}$ on the path. A cluster $\mathcal{A}$ has the* clear path property *if it has the path property and if for every non-distinguished position β in $\mathcal{P}$, there is a path x from the distinguished position α to β such that none of the interior nodes of x are positions in $\mathcal{P}$. Only the first and last nodes in the path are in $\mathcal{P}$. Such a path is called a* clear path *for $\mathcal{A}$. We can also define what it means for a (definite) Horn clause to have the (clear) path property, in a similar way. Note that A_n is the union of n^2 clusters, all of which possess the path property and the clear path property.*

Definition 1.5.14 *Suppose $\mathcal{A}$ is a cluster for $\mathcal{P}$ of height h with distinguished position α. A position β of $\mathcal{A}$ is* exterior *if there is some path x of length h from α that passes through β such that no other positions on this path besides α and β are in $\mathcal{P}$. Such a path is called an* exterior path *for β. A position is* interior *if it is not exterior. The* frontier *of a cluster is the set of positions at maximal distance from the distinguished position. Note that the height of a cluster is the maximum distance from the distinguished position of the cluster, to a frontier position. Also, a frontier position is exterior.*

Lemma 1.5.15 *Suppose $\mathcal{A}$ is a cluster that has the clear path property, and has height h. Then $\mathcal{A}$ has area at least $h + 1$, and has at least h exterior positions.*

Proof. Suppose α is the distinguished position of $\mathcal{A}$ and β is some frontier position. Then there is a clear path x from α to β. Now, for each position γ in x, we construct two paths γ_1 and γ_2, one by taking left children repeatedly and the other by taking right children repeatedly. Both γ_1 and γ_2 must contain positions of $\mathcal{A}$, since their initial positions can be reached by a clear path. However, among all the paths γ_i that can be constructed in this way for various γ, at least h of them must be mutually disjoint, as a simple geometric argument shows. (The ones to choose depend on how x goes from positions to left or right children.) Since each of these h mutually disjoint paths contains a position of $\mathcal{A}$, there must be h distinct such positions altogether. In fact, each of these paths must contain an exterior position, since we can take the last position of $\mathcal{A}$ encountered on that path. Thus there are at least h exterior positions, as claimed. This does not count the distinguished position, giving a total of $h + 1$ positions and thus an area of $h + 1$ or more, as asserted in the theorem.

□

Lemma 1.5.16 *Suppose $\mathcal{A}$ is a cluster that has the clear path property, and has height h. Suppose $\mathcal{B}$ is another cluster that has the path property and "subsumes" $\mathcal{A}$, in the sense that its positions are a proper subset of those of $\mathcal{A}$. Then $\mathcal{B}$ also has height h and contains all the exterior positions of $\mathcal{A}$.*

Proof. Since $\mathcal{B}$ subsumes $\mathcal{A}$, it must have the same distinguished position α. Let β be a frontier position of $\mathcal{A}$, and let x be a clear path from α to β. Note that the length of x is h. Since $\mathcal{B}$ subsumes $\mathcal{A}$, $\mathcal{B}$ cannot have any positions anywhere else on the path x besides at α and β. Since $\mathcal{B}$ has the path property, it must have β as one of its positions. Since the distance from α to β is h, $\mathcal{B}$ has height h or more. However, the height of $\mathcal{B}$ cannot be greater than h, so it must be exactly h. Now, the same argument (except for height considerations) applies to any exterior position γ of $\mathcal{A}$ too. There must be an exterior path passing through γ. Any cluster having the path property must contain some position on this exterior path, and so $\mathcal{B}$ must also. But since $\mathcal{B}$ subsumes $\mathcal{A}$, it must contain the position γ.

□

The same argument applies to subsumptions between individual clauses, which shows that the clauses eliminated from consideration by a cluster resolution (defined below) cannot cause trouble.

Definition 1.5.17 *Suppose $\mathcal{A}_1$ for $\mathcal{P}_1$ and $\mathcal{A}_2$ for $\mathcal{P}_2$ are two clusters. Suppose that there is a non-distinguished position α of $\mathcal{A}_2$ which is also the distinguished position of $\mathcal{A}_1$. Suppose that P_α or Q_α is the maximal predicate symbol in both clusters, in an ordering $<$ on predicate symbols. We define a* cluster resolution *that resolves on this maximal predicate symbol (P_α or Q_α) in all possible ways among the clauses in these two clusters according to A-ordering with the ordering $<$. Afterwards, the clauses in the clusters $\mathcal{A}_1$ and $\mathcal{A}_2$ are deleted. It turns out that the clauses produced by a cluster resolution are themselves a cluster for the set of positions $\mathcal{P}_1 \cup \mathcal{P}_2 - \{\alpha\}$.*

The reason for the definition of cluster resolution and the associated deletion of clusters is that since P_α or Q_α has effectively been eliminated, due to the way that A-ordering resolution works, the clusters $\mathcal{A}_1$ and $\mathcal{A}_2$ no longer effectively have all their clauses; such "partial clusters" complicate the analysis. We show an exponential lower bound even with them deleted, which implies an exponential lower bound if they are retained. Clauses that have been deleted will still have the path property, by reasoning similar to that in the following lemma; this is enough to show an analogue of lemma 1.5.16 for individual clauses.

Lemma 1.5.18 *Suppose $\mathcal{A}_1$ for $\mathcal{P}_1$ and $\mathcal{A}_2$ for $\mathcal{P}_2$ are two clusters, of heights h_1 and h_2, respectively, where $h_1 + h_2 \leq n$. Suppose that there is a frontier position α of $\mathcal{A}_2$ which is also the distinguished position of $\mathcal{A}_1$. Suppose that P_α or Q_α is the maximal predicate symbol in both clusters, in an ordering $<$ on predicate symbols. Then in one round of breadth-first A-ordering resolution using this ordering, we can resolve on P_α or Q_α and generate a cluster $\mathcal{A}$ of height $h_1 + h_2 - 1$. Also, $\mathcal{A}$ satisfies the (clear) path property if $\mathcal{A}_1$ and $\mathcal{A}_2$ do. The set of positions for $\mathcal{A}$ is $\mathcal{P}_1 \cup \mathcal{P}_2 - \{\alpha\}$.*

Proof. The height will be $h_1 + h_2 - 1$ since a path of this length may be obtained by joining longest paths from $\mathcal{A}_1$ and $\mathcal{A}_2$. One of these paths will end at α and the other will begin there. This will create a cluster, because one can obtain all combinations of literals at the various positions by resolving on appropriate clauses from the respective clusters. The path property is easy to verify, by piecing together paths in $\mathcal{A}_1$ and $\mathcal{A}_2$. The clear path property can be verified with a little geometric insight by considering the fact that α is a frontier position of $\mathcal{A}_2$, so that the paths in $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint (except for position α).

□

For this result, even if α is not a frontier position of $\mathcal{A}_2$, predicates that have been eliminated by A-ordering resolution cannot again be introduced, which prevents existing clear paths of $\mathcal{A}_2$ from being blocked by new positions from $\mathcal{A}_1$. This means that $\mathcal{A}$ will have the clear path property in this case, too. This is necessary to know, because cluster resolutions involving non-frontier positions may also be performed, and it is necessary to know that the clear path property is preserved.

Lemma 1.5.19 *Suppose uniform A-ordering resolution is done, starting from the set A_n of clauses. Suppose cluster resolution is done, so that clusters are deleted when some predicate in them is resolved on. Then, if the maximum area of any cluster is less than $n/2$, after every resolution, every predicate symbol will appear positively in one cluster and negatively in another cluster. Thus additional resolutions will always be possible, as long as the areas of the clusters are small.*

Proof. If the property is true before a round of uniform A-ordering resolution, it will be true after the round, since the effect of a round is to remove some clauses and generate new clusters. All predicate symbols that occurred positively and negatively in different clusters before the round, will still do so, because only one predicate symbol is eliminated per round, and that symbol will no longer appear in the remaining clauses. The limitation on height insures that the cluster will not border on itself, which could happen if it were so large that it could intersect all rows or columns of A_n. If a cluster bordered on itself, it would contain tautologous clauses (clauses that contain both a predicate symbol and its negation, for some predicate symbol).

□

Theorem 1.5.20 *Uniform A-ordering resolution on A_n will produce an exponential number of clauses, regardless of the ordering.*

Proof. Each cluster resolution increases the maximum height of a cluster by a factor of less than two. Also, if the maximum height is less than $n/2$, then eventually a cluster resolution will be done that increases the height of a cluster, since eventually some frontier literal will be the largest literal and will be chosen for resolution. Thus eventually a cluster with height h will be produced, where $n/2 \leq h < n$. A cluster of height h has area at least $h + 1$ and therefore at least 2^{h+1} clauses; for $h \geq n/2$ this is exponential in n. Now, if subsumptions occur, they only replace clusters by others of the same height, and having the same exterior positions. Any cluster of height h has at least h exterior positions, enough to give the exponential bound, so the argument is not affected. A subsumed clause is replaced by a clause having exactly the same exterior literals, which is all that we are concerned with anyway.

□

We note that this result is not affected by tautology deletion, since the heights remain less than n.

Corollary 1.5.21 *Breadth-first A-ordering resolution on A_n with an arbitrary ordering, will generate an exponential number of clauses.*

Proof. We note that the search space for breadth-first and uniform A-ordering resolution are essentially the same, except for subsumption deletion. However, as long as the heights are less than n, we have shown that subsumption deletion will not affect the exponential bound.

There is a subtlety that has to be dealt with in order to carry this through, and we indicate it here. That is, there are resolvents that do not correspond to cluster resolutions; these may involve clauses deleted in cluster resolution. Now, these "spurious" resolvents might conceivably increase in height much faster than the cluster resolvents, and then create clauses that would subsume the cluster resolvents and thereby reduce the search space. We sketch how this possibility can be excluded. First, it is not difficult to show that even the large spurious resolvents have to satisfy an extended path property that applies to paths of arbitrary length, not just n or less. Now, we are only concerned about the case when the maximum size of a cluster resolvent is less than $n/2$, since otherwise we know that the search space is exponential. The only problem is if there is a cluster $\mathcal{A}$ and some spurious clause C generated that properly subsumes some of the elements of $\mathcal{A}$, in particular, lacks some of the exterior nodes of $\mathcal{A}$. This can only happen if the "height" of C is larger than n, that is, it has wrapped around the torus. However, if this happens, then one can without much trouble construct an infinite path starting at the distinguished position of C that contains no other position of C, contradicting the extended path property. Such a path is obtained by going through the missing exterior position of $\mathcal{A}$, then taking enough right children until one can take a cycle of left children and never again encounter $\mathcal{A}$. This is always possible because $\mathcal{A}$ can be at most $n/2$ "wide," and an infinite path in wrapping around the torus can move to the right far enough to avoid $\mathcal{A}$ altogether.

As a consequence of this, the A-ordering resolutions can be done in an arbitrary fashion, not necessarily breadth-first, and we still can guarantee an exponential search space. Thus we can obtain an analogue of theorem 1.5.10 for A-ordering resolution, too, even with a good ordering.

□

We now continue a line of investigation begun earlier in section 1.5.4 about simulating P_1-deduction by A-ordering for unsatisfiable sets of clauses. It can happen that S is unsatisfiable but not well-ordered. In this case, it may not be possible to simulate P_1-deduction by A-ordering, since some of the A-ordering resolutions may involve positive literals from non-unit clauses. (This can happen if a clause contains only literals that are not derivable by positive unit resolution.) This is additional evidence that irrelevant clauses can cause combinatorial problems for the A-ordering strategy.

1.5.11 SLD-resolution

SLD-resolution is essentially the same as all-negative resolution with ordering of the negative literals, and has similar complexity properties. We do not explicitly mention it in the chart for this reason. However, because of its importance for Prolog, we make some comments concerning it. There are actually some differences between SLD resolution and all-negative resolution. One difference is that for each all-negative clause C, one of the literals L of C is chosen in an arbitrary way and all resolutions of C must resolve on the literal L. This is more flexible than all-negative resolution with ordering of the negative literals. Also, the search is typically performed depth-first rather than breadth-first as in our formalism. This can make Prolog programs faster than our analysis would indicate, because the ordering of clauses can cause the proof to be found early in the search. However, the proof depth is still a bound for the worst-case execution time, even with depth-first search.

The actual execution of Prolog programs is more restrictive than SLD-resolution; there is less flexibility in which literals can be resolved. Literals must be chosen for resolution in a last-in first-out manner. Subject to this, a Prolog program specifies the ordering by the order of the (negative) literals in the body of the clauses. Another difference between SLD-resolution and Prolog is that duplicate subgoals will be deleted from a clause but not from the Prolog execution. Thus if we have the clause $\{\neg P, \neg Q\}$ and resolve with $\{Q, \neg P, \neg R\}$ we obtain $\{\neg P, \neg P, R\}$ and the two occurrences of $\neg P$ merge into one. However, in Prolog execution, the procedure P would be called twice.

The last-in first-out restriction for SLD-resolution means that the literal of an all-negative clause C chosen to resolve on must be one of the literals most recently added to C. It turns out that the complexity properties of this "last-in first-out" SLD-resolution are different than those of SLD-resolution in general on some clause sets. For example, any Prolog program for T_n^3 will have exponential proof depth, even though there is a polynomial length all-negative resolution proof. The same subgoals will be created and solved repeatedly. Another kind of exponential behavior occurs for the satisfiable Horn set S_n^3, even though none of the subgoals will be successfully solved. This suggests a possible deficiency in the current methods of logic programming implementation. One would expect Prolog programmers to choose good literal orderings, which should help the complexity in most cases. However, on unstructured problems, inefficiencies might occur. Some method of caching successes and failures is necessary to overcome these inefficiencies. It might make Prolog more convenient in some cases if it were automatic. One strategy that appears to overcome the problems with SLD resolution is mentioned in [Lyn94]. Actually, that paper considers a more general framework than just SLD-resolution and gives a fairly general mechanism for reducing the search space.

Since Prolog is efficient in practice, we look for special cases where SLD-resolution performs well. For deterministic, well-ordered clause sets, there is an ordering that causes breadth-first SLD-resolution to have polynomial search depth and duplication by iteration and also polynomial duplication by combination. For such clause sets, A-ordering and all-negative resolution with a good ordering of the predicate symbols also have polynomial search depth and duplication by iteration and polynomial duplication by combination. For all these strategies, the good behavior can be obtained by always resolving on the predicate symbols that are largest in the well-ordering (or some total extension of it). The polynomial duplication by combination occurs because there is always only one such resolution possible from a given all-negative clause, since the clause set is deterministic, and the linear duplication by iteration follows because the ordering prevents predicate symbols that have been resolved away, from being reintroduced. Recall that minimal unsatisfiable Horn sets are deterministic and well-ordered.

For arbitrary unsatisfiable Horn sets, these results continue to hold if depth-first search is specified and the proper ordering of clauses and literals is used. To see this, suppose S is an unsatisfiable Horn set, and let T be a minimal unsatisfiable subset of S. Note that T is deterministic and well-ordered. Suppose we order the clauses so that the clauses in T are used before those in S. Also, suppose we choose the ordering for SLD-resolution so that the literals with the highest proof complexity are resolved on first. Then each SLD resolution will replace a negative literal by negative literals of smaller proof complexity. Assuming that duplicates of a given literal are deleted, this will result in a proof in a linear number of steps. In general, of course, depth-first search can lead to infinite loops and sacrifices completeness. We note that Prolog cannot always achieve this good behavior (proofs in a linear number of steps) because last-in first-out SLD resolution does not always permit the desired ordering of literals and because a given subgoal may be solved repeatedly. If each literal occurs at most once in the body of a clause (that is, negatively) then even Prolog can achieve polynomial behavior since subgoals will be solved at most once.

1.5.12 Set of support

The set-of-support restriction [WRC65] initially chooses some subset of the input clauses as the *support set*. This should have the property that the remaining clauses (not chosen) are satisfiable. A clause is *supported* if it is in the support set, or if it is the resolvent of two clauses, at least one of which is supported. The support strategy restricts resolutions to those in which one of the parent clauses is supported. The behavior of resolution with the set-of-support restriction and a negative set of support is the same as all-negative resolution, for Horn clauses. For Horn clauses, a resolvent of an all-negative clause and another clause is always all-negative. Therefore all-negative resolution produces the same search space as set of support. If the set of support is chosen as the set of positive clauses, then the behavior is like P_1-deduction except that additional resolvents can be generated. If some other support set is chosen, it is conceivable that the behavior could be worse. We note that there is no way to get polynomial behavior and goal sensitivity with set of support, regardless of the choice of support set. In order to get goal sensitivity, the set of support has to consist only of the negative clauses. This means that the behavior is the same as all-negative resolution, which has exponential duplication by combination. We view this as evidence that set-of-support is also combinatorially inefficient, though in practice it is one of the better traditional strategies.

1.5.13 Model elimination

For Horn sets, assuming that a negative clause is chosen to start the search, model elimination [Lov69] and the MESON strategy [Lov78] behave essentially the same as SLD-resolution or all-negative resolution with an ordering on the negative literals. These strategies follow Prolog's execution model fairly closely for Horn clauses, and so they actually correspond to last-in first-out SLD-resolution. However, they permit duplicate literals to be deleted, unlike actual Prolog execution. Also, the search formalism is different because this strategy in the general (non-Horn) case is still an input strategy, that is, each inference involves a "chain" and an input clause. Therefore, what appears as duplication by combination in all-negative resolution with ordering, appears as duplication by case analysis in model elimination and the MESON strategy. There is another difference, namely, the MESON strategy and model elimination have a feature that prevents infinite loops. For example, we might have clauses $\neg P$, $P\ :-\ Q$, $Q\ :-\ R$, and $R\ :-\ Q$. Now, letting P be the starting subgoal, we successively generate Q, R, then Q again, and can get into an infinite loop in Prolog. This cannot happen with model elimination and the MESON strategy. The reason is that their data structures (chains) keep information corresponding to the stack of subgoals activated. Whenever it is detected that a subgoal has been attempted from within a call of the same subgoal, that particular chain can be deleted.

By considering these strategies in terms of SLD-resolution in this way, we can analyze them and see that they are goal-sensitive (if a negative clause is chosen to start the search), have no (i.e., constant) duplication by combination, since each node in the search space consists of a single chain, and have exponential duplication by case analysis. The exponential behaviors are from the clause sets S_n^2, S_n^3, T_n^2, and T_n^3, and the reason for this is essentially the same as that for all-negative resolution with a bad ordering. The search depth (and therefore the duplication by iteration) is exponential, because the last-in first-out restriction causes subgoals to be solved repeatedly in T_n^2. This is true even with a good ordering, since even with an ordering the strategy is subject to the last-in first-out restriction. These results hold even for minimal unsatisfiable clause sets, as witnessed by a minimal unsatisfiable subset of T_n^2. For satisfiable, deterministic, well-ordered clause sets with a good ordering, the behavior can be made polynomial. This is because there must be at least one subgoal that is not derivable, and by properly guessing which one it is one can fail quickly. For clause sets that are not well-ordered, it is not clear what happens in general, even with a good ordering.

1.5.14 Lemmas and caching

It is possible to use a lemma mechanism with model elimination and the MESON strategy. This makes use of the fact that when a negative literal $\neg P$ is eliminated by resolution, and all literals descending from $\neg P$ are also eliminated, then we have essentially derived a proof of P. This corresponds to a successful return from a call to the procedure P in Prolog. This means that any further occurrences of $\neg P$ can also be eliminated by the "lemma" P. That is, further calls to the procedure P will also return successfully, so the computation does not need to be repeated. When using lemmas, we have to modify the search space structure, since the chains interact. This makes the search linear, and so instead of duplication by case analysis we now have duplication by combination. Therefore the duplication by case analysis is $O(1)$. The search depth is now linear, because of the lemmas, as is the duplication by iteration. To see this, note that whenever a subgoal is called, it increases the length of the procedure stack by one. This stack is linearly bounded in length, because a chain can be deleted if some procedure appears twice on the stack. Whenever a procedure returns, the stack reduces in length by one but the fact that this procedure has successfully returned is added as a lemma. Therefore, each call to a non-lemma procedure and each return from a non-lemma procedure either increase the size of the stack or increase the number of lemmas Let N be the sum of the stack size and twice the number of lemmas. Then N increases by 1 whenever a non-lemma procedure is called or returns successfully. The maximum value of N is three times the number of predicates in the original set of clauses. This shows that the search depth and the duplication by iteration are linear. We are not counting the procedure calls that are already lemmas. Each such call only leads to a constant amount of additional search depth, and the number of such calls is bounded by the sum of the sizes of the clauses in the set of input clauses. Therefore the linear bounds are still valid. The duplication by combination is still exponential, as verified by S_n^2 and S_n^3, where no lemmas are generated.

We now consider caching. By this we mean that failures as well as successful returns from a procedure are remembered. If a procedure was called and failed before, then the computation does not have to be repeated when it is called again. This caching of failures only helps reduce the search space if the search is done in a depth-first manner as in Prolog, because then there are no two calls to the same procedure operating in parallel, or if parallel calls to a procedure are combined in some way. Depth-first search can be done because the loop-detecting feature of these strategies prevents infinite chains of procedure calls. However, this loop-detection makes the search space dependent not only on the current procedure being called but also on procedure calls earlier in the stack. This means that the caching is unsound unless this dependence on the stack is eliminated by removing loop-detection, and thereby losing first-order completeness. We assume that some kind of depth-first iterative deepening [Kor85, ST85] search is done to avoid infinite loops and help organize the caching. Then, with each subgoal, we cache not only whether it returns successfully, but how much depth of search it was permitted. Assuming the search is done in this way, each failing return from a procedure increases the size of the cache, and so reasoning as above we can show that the duplication by combination is linear, as well as the search depth and duplication by iteration. However, since there are a number of stages of depth-first iterative deepening, each one taking a linear amount of duplication, it may be more accurate to say that the duplication is quadratic. But our search formalism is not really adapted to consider depth-first iterative deepening in a natural way, so we just assume an optimal depth bound and state the duplication as linear.

1.5.15 The MESON strategy

The MESON strategy has behavior like model elimination for Horn clauses, so the bounds are the same. For the MESON strategy, unit lemmas result in behavior like that of model elimination with unit lemmas. The MESON strategy with unit lemmas and caching has behavior like model elimination with unit lemmas and caching.

1.5.16 Problem reduction formats

The simplified and modified problem reduction formats [Pla82, Pla88] simulate Prolog's back chaining mechanism, but are complete for first-order logic. The simplified problem reduction format [Pla82] without caching is much the same as model elimination, for Horn clauses. The simplified problem reduction format generates formulae called *decompositions*. For Horn problems, all the decompositions generated are input Horn clauses, so the number of them is linear (if caching is done). The inference mechanism essentially simulates hyper-resolution, once a sufficient number of decompositions are generated. Thus, the search depth and duplication by iteration are linear. The duplication by combination is linear as for hyper-resolution, if caching is done. As with model elimination and the MESON strategies, the organization of the search for depth-first iterative deepening may mean that the duplication should really be considered as quadratic. However, for simplicity we assume an optimal depth bound and thus have linear duplication. Actually, it is possible to obtain these good bounds without caching or lemmas; it is only necessary to eliminate duplicate decompositions. The reason for this is that the decompositions are "local" and contain all necessary information about the chain of procedure calls. This makes it possible to cache without losing first-order completeness or efficiency for Horn problems. This is an advantage over MESON and model elimination. The behavior of the modified problem reduction format [Pla88] for Horn clauses is the same as that of the simplified problem reduction format, both with and without caching, since the two methods differ only in how non-Horn clauses are treated.

1.5.17 Clause linking

The clause linking method [LP92] reduces first-order logic to propositional calculus, and then applies a Davis and Putnam-like procedure [DP60, DLL62]. This reduction to the propositional calculus is done by successively instantiating the clauses using unification with literals of other clauses. A propositional decision procedure is then periodically applied to the resulting clauses. We consider the application of this method to propositional Horn clauses. For the clause linking method with forward support, we note that this method behaves essentially the same as hyper-resolution, and the same bounds apply to it. For backward support, we note that no new clauses are generated, and it is only the support status that gets changed. If S is unsatisfiable, then eventually an unsatisfiable subset of S will be marked as backward supported. After a linear number of rounds, all the clauses that can be backward supported, will be, and the search will terminate. Then the Davis-and-Putnam-like decision procedure will terminate in polynomial time, since S is a Horn set. Thus the search depth and duplication by iteration are linear. The duplication by combination is linear, since no new clauses are generated (except by unit simplification). The duplication by case analysis is $O(1)$ because each state has one successor. Caching is not necessary, because clauses are never combined except briefly in the Davis-and-Putnam-like decision procedure. Thus we obtain goal-sensitivity and good behavior without requiring caching or losing first-order completeness. These are advantages over the simple and modified problem reduction formats as well as over MESON and model elimination.

1.5.18 Connection calculi

As for connection calculi [Bib87], there are many of them. The connection calculi make use of connections between literals in (possibly) different clauses to control the search. The chart is only intended to show that they can be implemented to simulate forward reasoning, like hyper-resolution, or the backward chaining resolution strategies. It is also of course possible that connection calculi with behavior like that of the clause linking method exist.

1.5.19 Caching

Several of these strategies perform similarly on Horn sets. They are model elimination and the MESON strategy with caching of unit subgoals and lemmas, the simplified and modified problem reduction formats with caching, and clause linking with backward support. We refer to these collectively as backward chaining methods with caching (or, as *caching strategies*). Though clause linking does not cache, we include it here because the behavior is similar and because the deletion of duplicate copies of a clause can be regarded as caching.

1.6 Preventing Unit Simplifications

We now give modifications of these clause sets that still display the same exponential behavior, even for strategies used together with unit simplification. For some of these clause sets, we do not know how well the various theorem proving strategies will perform. First we give a transformation within propositional calculus that defeats unit simplification by introducing non-Horn features. A second transformation prevents unit simplification by introducing first-order features but retains the Horn property.

The following transformation eliminates unit simplifications for back chaining methods: For each Horn clause $L\ :-\ L_1, ..., L_n$ where L and all L_i are positive literals, delete this clause and replace it by the clauses $L, P\ :-\ L_1, ..., L_n$ and $L\ :-\ P$, where P is a new predicate symbol. Note that the first of these is a non-Horn clause, since both L and P are positive literals. This eliminates the possible unit simplifications for back chaining methods, but produces a non-Horn set of clauses. For forward chaining methods, the unit L can be rederived and some unit simplification can still occur. Let $U(S)$ be S transformed in this way. Later we discuss the search space sizes of various strategies on clause sets produced by the U operator.

We now give alternative methods to avoid unit simplification efficiencies, which introduce first-order features but retain the Horn property. Also, these transformations often produce unsatisfiable sets of Horn clauses; this answers the question whether such exponential behavior can be produced in unsatisfiable Horn sets. Of course, such behavior cannot be produced in unsatisfiable propositional Horn sets if unit simplification is allowed, since that will in itself find a contradiction. However, by allowing limited first-order features, we can still obtain exponential behavior for Horn sets. Formally, if S is a set of propositional clauses, let $M(S)$ be a set of monadic first-order clauses with positive unit clauses P in S replaced by $P(a)$, and other clauses in S transformed by replacing positive literals P by $P(x)$ and negative literals $\neg P$ by $\neg P(x)$. Thus a clause $\{P, \neg Q, \neg R\}$ would be replaced by $\{P(x), \neg Q(x), \neg R(x)\}$, but $\{P\}$ would be replaced by $\{P(a)\}$. Then $M(T_n^2)$ and $M(T_n^3)$ are unsatisfiable Horn sets and have exponential behavior for back chaining strategies without caching, and unit simplification doesn't remove the exponential behavior. During the proof search, some generated clauses will have the variable x bound to a, but this only has the effect of replacing some of the unit resolutions on these clauses with unit simplifications, and does not significantly affect the search space.

1.7 Additional Hard Sets of Clauses

In addition to these transformations, we give some more hard clause sets. We give a set of clauses that is hard for forward chaining methods but easy for backward chaining methods. We also give an example that is hard for both forward and backward chaining methods, but for which clause linking still has polynomial behavior. For some of the clause sets, we do not know which strategies exhibit polynomial behavior. We give non-Horn propositional examples, and Horn non-propositional examples.

We first give an example of a first-order Horn set that is hard for forward-chaining methods but easy for back chaining methods. Consider the set containing the clause $P(x_1 ... x_n) \ :- \ Q_1(x_1), ..., Q_n(x_n)$, together with the unit clauses $Q_i(a)$ and $Q_i(b)$, for all i. Suppose the goal is $:- \ P(a, a, ..., a)$. Call this set S_n^4. Then there are exponentially many hyper-resolvents, but back chaining is linear. Note that back chaining methods will bind the variables to a and then limit the use of the Q_i, but forward chaining strategies will generate many bindings of variables to combinations of a and b.

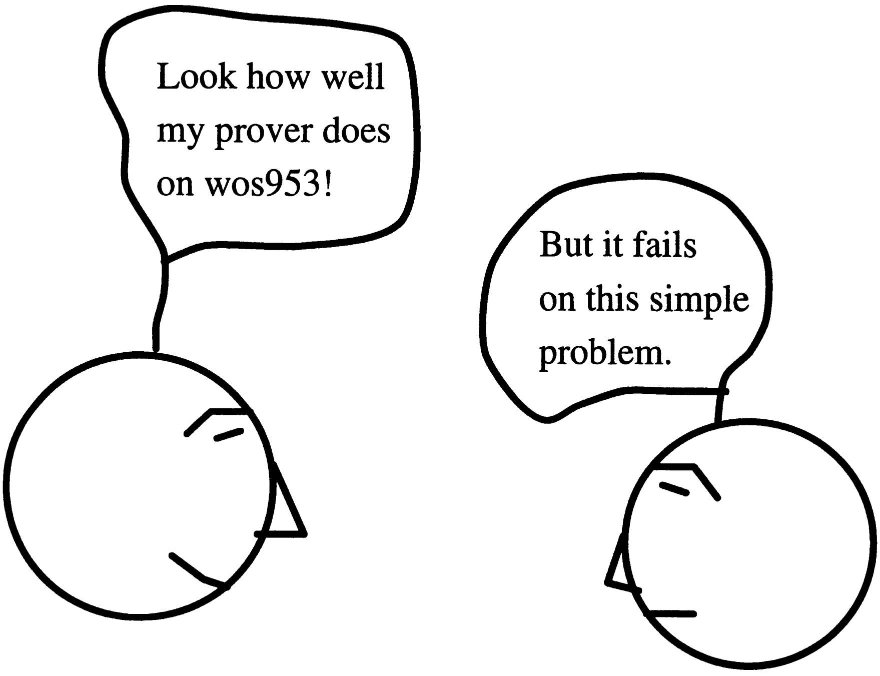
We now exhibit a Horn set that is hard for all the non-caching strategies considered here. Given a set S of propositional clauses, let $N(S, n)$ be S with clauses $P\ :-\ P_1, ..., P_m$ for $m \geq 1$ replaced by $P(x_1...x_n)\ :-\ P_1(x_1...x_n), ..., P_m(x_1...x_n)$. Also, a positive unit clause $\{P\}$ is replaced by $P(x_1...x_n)\ :-\ Q_1(x_1), ..., Q_n(x_n)$. In addition, negative clauses $:-\ P_1...P_m$ are replaced by $:-\ P_1(a, a, ..., a), ..., P_m(a, a, ..., a)$. Finally, the unit clauses $Q_i(a)$ and $Q_i(b)$ are added. Then the sets $N(T_n^2, n)$ and $N(T_n^3, n)$ are unsatisfiable first-order Horn sets that produce exponential behavior for backward chaining strategies (except the caching strategies) because T_n^2 and T_n^3 do. They also produce exponential behavior for forward chaining strategies because there are exponentially many combinations of a and b for the n variables. This includes clause linking with forward support. However, these can be solved in polynomial time by strategies with caching because the backward chaining from the goal binds variables to a, and eliminates the exponentially many combinations of a and b. This includes clause linking with backward support. In addition, unit simplification for these clause sets does not help; it only replaces some of the unit resolutions with unit simplifications.

We now consider the operation of reversing the signs of all the literals in a clause. Equivalently, we could consider leaving the signs unchanged but reversing the way the strategies treat the signs. This causes hyper-resolution to become negative hyper-resolution, which means that all the positive literals in a clause are resolved in one operation. For Horn clauses, each clause has only one positive literal, and so negative hyper-resolution is the same as all-negative resolution, with the same search space complexities.

If S is a set of clauses, let $\overline{S}$ be S with the signs of all predicate symbols changed, and predicate symbols systematically renamed to new predicate symbols. Note that this causes forward chaining strategies to behave like backward chaining strategies, and vice versa. Let $Sym(S)$ be $S \cup \overline{S}$. Note that $Sym(S_n^2)$ and $Sym(S_n^3)$ have exponential behavior for hyper-resolution, since negative hyper-resolution (all-negative resolution) is exponential for S_n^2 and S_n^3. These satisfiable propositional sets of clauses have exponential behavior for all strategies (and refinements) discussed except clause linking and possibly MESON, model elimination, and the two problem reduction formats, all with caching. The reason these clause sets are easy for clause linking is that the Davis and Putnam-like decision procedure will quickly find a model and detect satisfiability. This is because the Davis and Putnam method is polynomial time on Horn sets [GU89], and these clause sets are similar enough to Horn sets to exhibit the same behavior. These sets will probably be quite a challenge for most theorem provers. However, they are not Horn sets. To obtain unsatisfiable propositional sets of clauses with this property, we can use $Sym(U(T_n^2))$ and $Sym(U(T_n^3))$, discussed below. Even for the simplified and modified problem reduction formats, the behavior is probably exponential, because many combinations of literals will be generated. It is possible that clause linking has polynomial behavior, but that depends on how the Davis and Putnam-like decision procedure works.

Consider $U(T_n^2)$ and $U(T_n^3)$. These are unsatisfiable propositional non-Horn sets which have polynomial behavior for P_1-deductionand hyper-resolution, even though unit simplification will not decide these clause sets. To show this, recall that in these clause sets we have clauses of the form $L, P \; :- \; L_1, ..., L_n$ and $L \; :- \; P$, where P is a new predicate symbol. We can show that the literals L_i will eventually be derived, and then the clause L, P will be derived by a sequence of unit resolutions (unit simplifications). Finally, a resolution operation between the clauses L, P and $L \; :- \; P$ will produce the resolvent L, and the proof will proceed further. $U(T_n^2)$ has exponential behavior for all the back chaining methods even with unit simplification, except possibly the caching strategies, since T_n^2 does. $U(T_n^3)$ can be solved in polynomial time by all-negative resolution with a proper ordering of negative literals. It is possible that $U(T_n^2)$ and $U(T_n^3)$ can be decided in polynomial time by a Davis-Putnam-like method with lemmas, as described in [Pla90]. We don't know how fast they can be solved by the ordinary Davis-Putnam method.

To defeat forward chaining methods, consider the clause sets Sym $(U\ (T_n^2))$ and Sym $(U\ (T_n^3))$. These still have exponential behavior for back chaining methods except possibly the caching strategies. In addition, they have exponential behavior for forward chaining methods, since the Sym operation causes forward chaining strategies to behave like backward chaining strategies. However, it is possible that a modified Davis-Putnam method with lemmas, as described in [Pla90], will decide these in polynomial time. If so, clause linking will also have polynomial behavior on these clause sets.


Look how well my prover does on wos953!
But it fails on this simple problem.

1.8 Discussion

Since our chart only considers propositional Horn sets, the results are somewhat limited. Still, even here some unexpected behavior occurs. In addition, we present other clause sets that are non-Horn or first-order and discuss the behavior of strategies on them. However, a more formal analysis for non-Horn propositional logic and for first-order logic would be interesting and probably difficult. Also, these results do not say how a theorem prover will perform on real theorems. Despite this, we believe that strategies having a large amount of search duplication often perform badly. Since the successes of a prover are usually more widely publicized than the failures, the poor performance of some provers on certain problems may not be well known to those outside of the theorem proving community. Our experience has been that on simple set theory problems, logic puzzles, and even propositional calculus problems, most of the strategies listed in the chart perform very badly. Also, we have found that on typical theorems, back-chaining strategies that do not use caching perform badly on Horn sets, and even back-chaining strategies that do cache (except for clause linking) perform badly on non-Horn sets. Forward chaining strategies generally seem to do well on Horn sets, but badly on non-Horn sets. Of course, forward chaining strategies are typically not sensitive to the theorem being proved, and function somewhat like blind search.

The methods of this paper are in a way more discriminating than the results of Haken [Hak85], who showed that resolution is exponential for any refinement. This tends to suggest that all strategies are the same. Our methods discriminate between strategies more finely, and support the argument that some strategies are better than others. Eder [Ede92] analyzed the sizes of proofs in different strategies. In contrast, we analyze the size of the entire search space.

We now consider briefly the behavior of clause linking in first-order logic. For a description of this strategy see [LP92]. The question arises whether clause linking is exponential on any first-order clause set for which other methods are polynomial. This is unlikely for any clause set for which the term sizes are small, since the behavior approximates propositional logic in that case. We note that for typical theorems, if the term sizes get large, often the proofs are too difficult to obtain. So, if a small bound on term size is set, then clause linking will probably be efficient, even for first-order logic. However, [Zam72, Zam89, Tam90, Tam91] have some first-order examples where a particular refinement of resolution similar to A-ordering is a decision procedure, but clause linking may generate an infinite search space. On the other hand, it is possible to construct simple first-order examples where clause linking generates a finite search space but resolution generates an infinite search space. For example, clause linking has a finite search space on clause sets containing variables and constant symbols but no function symbols. Thus clause linking is a decision procedure for this class. However, resolution can generate an infinite search space on these clause sets. For example, consider the following clause set: $\{\neg X = Y, \neg Y = Z, X = Z\}, \{\neg a = b\}$. All-negative resolution generates an infinite search space on this clause set. For almost any two theorem proving strategies, it's possible to find examples where one performs better than the other. We would at least hope to have a good understanding of where this occurs and why. Also, we would like to find strategies that have polynomial behavior on the examples presented here and also decide the clause sets decidable by resolution with A-ordering.

1.8.1 Adequacy

We have given a theoretical analysis of a number of strategies on propositional Horn sets and some first-order clause sets. We do not suggest that all theorem proving methods should be analyzed theoretically in this manner, since that would require all researchers to be theoreticians to some extent. Another acceptable alternative is to run a theorem prover on the clause sets given above and estimate the growth rate of the time taken. We propose that all new strategies be analyzed this way, analytically where possible and also by running them on these clause sets for all n up to say 50 (subject to time and space limitations!). The clause sets that seem most significant for this are S_n^2, S_n^3, T_n^2, T_n^3, A_n, $Sym(S_n^2)$, $Sym(S_n^3)$, $U(T_n^2)$, $U(T_n^3)$, $M(T_n^2)$, $M(T_n^3)$, $Sym(U(T_n^2))$, $Sym(U(T_n^3))$, $Sym(M(T_n^2))$, $Sym(M(T_n^3))$, $N(T_n^2, n)$, and $N(T_n^3, n)$. Note that some of these clause sets are easy for standard methods; this helps to distinguish between methods that have exponential behavior on the harder clause sets. The performance of a strategy on these examples doesn't tell everything about the strategy, but does tell something. We say a strategy is *adequate* if it runs in polynomial time on all these clause sets, as well as propositional Horn sets, and satisfies the following additional requirements: It should be complete, goal-oriented, and natural. Natural means that the strategy is not specifically designed to do well on these clause sets. It is possible that a good theorem prover may behave poorly on these examples. However, we believe that a strategy that performs well on these example clause sets will also do well on typical theorems. We don't know if any existing strategy is adequate, but clause linking may be. We note that clause linking has polynomial behavior on all the above clause sets, because the time taken by the Davis and Putnam procedure is not reflected in the search space size. However, since adequacy is defined in terms of running time, clause linking could still fail to be adequate.

1.8.2 Extensions

It would also be possible to extend this analysis to renameable Horn sets, that is, sets of clauses that can be transformed into Horn sets by changing the signs of some predicate symbols. Such an analysis would treat forward chaining and backward chaining strategies the same, and both would have exponential behavior. However, clause linking and maybe some of the other caching strategies would still have polynomial behavior. Note that UR-resolution has polynomial behavior on renameable Horn sets. However, this is not a complete strategy in general. Still, its good performance in practice tends to confirm its favorable theoretical properties.

We now briefly discuss a general analysis for first-order logic. For this case, it is not possible to bound search space size in the same way, since there is no recursive bound on the length of proofs and hence on the search space size. However, what we can do is analyze how efficient a strategy is on the structures it generates. For resolution, if the same literals are generated over and over again and combined in many different ways, then this may indicate an inefficiency. We can analyze this behavior for a general strategy by associating with each generated clause a set of instances of the input clauses that were used to generate it. We can then consider how often a given input instance contributes to the different clauses in a state, on a path of states, or in a set of states no two of which are on the same path. In this way, we might learn something about the various types of search space duplication that occur in first-order logic theorem proving.

Chapter 2

The First-Order Complexity of First-Order Theorem Proving Strategies

2.1 Introduction

We would like to study the complexity of first-order theorem proving procedures. A first-order theorem prover typically takes as input a formula A of first-order logic and, if the formula is valid, eventually outputs a proof. Otherwise, the prover may run forever. Since first-order logic is only partially decidable, this is the best we can hope for. One can easily show that there can be no recursive bound on the length of the running time in terms of the length of the input, even for valid formulas. This makes it appear impossible to do any meaningful complexity analysis, since it is impossible to have a theorem prover with, say, exponential running time. However, we would like some kind of an asymptotic complexity analysis, because it would give valuable and machine-independent insight into theorem provers, insight that is difficult to obtain by running examples alone. A theoretical analysis, for example, can tell us something about the behavior of a prover on infinite classes of problems, which cannot be determined by running examples. Such an analysis would enable us to say in a rigorous way what it means for one theorem proving method to be better than another, at least on certain kinds of problems, and would enable us to pose interesting open questions about the existence of theorem provers of various complexities. It would also suggests new approaches to theorem proving, as we will show, approaches that might not have been considered otherwise. Of course, experimental studies of theorem provers provide additional information that should be combined with theoretical insights to obtain a complete picture.

Our approach to a theoretical analysis of theorem provers is to find some natural partial function F from first-order formulas into the natural numbers such that $F(A)$ is defined iff A is valid and such that theorem provers exist whose complexity on valid formulas is recursive in $F(A)$. Also, we require that the question whether $F(A) \leq n$ should be decidable, given A and n. We call such an F a *theorem proving complexity measure* because it measures how hard A is, from a theorem proving perspective. Of course, such a function F cannot be recursive. We can take for F the minimal length of a proof of A in some logical system, for example. Then, since one can enumerate all possible proofs, one might be able to do theorem proving in time exponential in $F(A)$. However, it seems better to find measures F that do not depend on a particular system of logical inference, since there are many plausible inference systems, and the choice of one is, after all, somewhat arbitrary. We exhibit a number of such theorem proving complexity functions F that are not inference-based, and use them to analyze the performance of theorem proving procedures. We also consider functions F that are inference based; these are defined in terms of the lengths or depths of resolution proofs.

In addition to studying the time needed to show that A is valid as a function of $F(A)$, we can also study the relation of the size of a minimal proof of A to $F(A)$. Since minimal proof size can be taken as another function $F'(A)$, this question is essentially that of studying the relationship between different theorem proving complexity functions.

Now, in addition to measuring the complexity of theorem proving procedures relative to F, we also want to study the inherent complexity of theorem proving relative to F, that is, what can we say about the best possible complexity of a theorem proving procedure relative to F. For this, we use a slightly different formalism. We consider the problem, given a pair (A, n), where A is a first-order formula and n is an integer in unary, to determine whether $F(A)$ is defined and $F(A) \leq n$. We are interested in the complexity of this problem. Note that if $F(A)$ is defined, A is valid, so that any algorithm to solve this problem can be used as a theorem prover for first-order logic. However, the problem as stated is a purely complexity theoretic problem, and one can ask whether it is NP-complete, exponential time complete, or in some other complexity class. This formulation leads to a number of interesting questions and results in complexity theory, and we hope that this will stimulate some work in computational complexity concerning the complexity of theorem proving in first-order logic and other logics.

The above discussion is concerned with the validity of first-order formulas, to bring out the essential ideas. However, in our study, we will consider clause-form formulas. It is known that for any first-order formula A, there is a set S of clauses such that A is valid iff S is unsatisfiable, and S can effectively be computed from A. Then in order to prove that A is valid, we can prove that S is unsatisfiable, and this is the approach used by many current theorem provers. In this context, we define a theorem proving complexity measure as a partial function F from sets S of clauses into the natural numbers such that $F(S)$ is defined iff S is unsatisfiable and such that there are theorem proving procedures that are recursive with respect to F on unsatisfiable clause sets. Also, we require that the question whether $F(S) \leq n$ should be decidable, given S and n. One advantage of this approach is that clauses are considerably simpler syntactically than full first-order logic, and therefore more amenable to machine proving. Clauses are essentially conjunctive normal form formulas in which existential quantifiers have been eliminated, so that all variables are implicitly universally quantified. Another advantage of considering clause form is that we can apply Herbrand's theorem, which states that S is unsatisfiable iff there is a finite unsatisfiable set T of ground (propositional) clauses such that each clause of T is an instance of some clause in S. Now, the length of a shortest such T, written as a character string, can be much longer than the length of S. In fact, there is no recursive bound on the length of such a set T in terms of the length of S. However, we can bound the complexity of theorem proving procedures in terms of the length of T. It is not hard to devise a theorem prover that is exponential in the length of T, for example. One only needs to enumerate all possible T of length n, of which there are a number exponential in n, and apply an exponential propositional decision procedure to each one. Therefore we can let $F(S)$ be the length of the shortest such T, and then F is a theorem proving complexity measure. There are other measures one can use, such as the minimal number of clauses in any such T, or the minimum over all such T of the maximum size of the literals in T, or the length of a minimal resolution proof of the empty clause from S. We will define the measures M_{pd}, M_{pl}, M_{dup}, M_{lin}, and M_{sub} in this paper. The first two are inference-based, but the last three are not. Relative to these (and other) measures, we can study the complexity of theorem proving strategies. We can also compare the various measures with each other, and comment on which measures seem more reasonable. This is the approach that will be taken in this paper.

In the past, there has been some work on the complexity of theorem proving strategies, but mostly with respect to the length of proofs, and not to the difficulty of finding a proof. For some examples of studies of proof lengths, see [CR79, Hak85, Urq87, Ede92]. Haken [Hak85] showed that for a set of propositional problems, known as the pigeonhole problems, resolution needs to generate an exponential number of clauses in order to find a proof. This means that the minimum proof length is exponential for resolution, in this case. This work was extended by Buss and Turán [BT88]. The difficulty of finding a proof, and the size of the search space, are more relevant for the efficiency of theorem provers than the size of a minimal proof. There has been very little work on search space size. The paper [KBL93] shows that many refinements of resolution do not increase a certain measure of search space size by more than a factor of four, but does not compare refinements with one another. Their paper considers monotone refinements of resolution; these do not allow deletion operations such as deletion of subsumed clauses. However, the results are otherwise very general. An exception is the work of Plaisted [Pla94b, Pla94c], which considers the size of the search space generated. However, this work is largely propositional in nature, and here we wish to extend this work to an inherently first-order context. The price for this is that we need to consider specific search strategies, which was not necessary in [Pla94b, Pla94c]. Plaisted [Pla84] and Goubault [Gou94] studied complete problems in first-order logic. In particular, [Pla84] studied the problem of determining the minimal depth of binary resolution proofs from a set of clauses. A series of completeness results was obtained, ranging from nondeterministic exponential time for general first-order clauses to exponential time for some restricted subsets. Many other completeness results related to first-order logic were also presented in [Pla84]. Goubault [Gou94] showed that the problem of determining the minimal number of copies of first-order clauses needed for a proof is Σ_2^p complete. The paper [Let93] studies how accurately the length of a derivation reflects the actual complexity of a proof. For some work concerning the effect of different translations to clause form on the proof complexity, and how they can make a non-elementary difference in proof length, see [BFL94, Egl96].

There has been a considerable amount of work on the complexity of unification problems (for example, see [KN92]), but we are more concerned with pure first-order strategies here. There have also been works showing how one strategy can simulate another, which is more relevant for proof length than search complexity. One such study is [BF93], in which the *consolution* calculus is modified in such a way that it can be instantiated to a number of strategies. The paper [Bib82] presents a method of embedding ME-like strategies into each other. The handbook article [BE93] presents a great variety of calculi and compares their properties, as well as giving some simulation results. The book [Ede92] gives precise simulation results between first-order calculi from a complexity point of view. Hsiang and Bonacina [BH96] present a formalism that facilitates the study of infinite search spaces. The paper [BH98] presents a model for representing search in theorem proving. This model captures the notion of *contraction*, which is basically deletion of a derived clause or formula. The ability of contraction to reduce search spaces is studied. The paper [Bon] analyzes parallel implementations of contraction-based strategies in a machine-independent manner.

Here we analyze the complexity of first-order theorem provers. This work reveals a number of surprising differences in efficiency among common theorem proving strategies, ranging from single to quintuple exponential. These differences are not at all obvious from a casual inspection of these strategies. This analysis also suggests some simple modifications of existing strategies that have a better asymptotic behavior. We have felt for a number of years that common strategies were very inefficient on some problems and that this was significantly hindering the field, and this current analysis helps to confirm this and to suggest the advantages of certain methods which avoid these inefficiencies. On the other hand, this analysis also reveals cases where these traditional strategies perform fairly well, possibly helping to explain their popularity.

Another outcome of this work is that we can begin to say something about which kinds of clause sets S are best for which strategies in terms of various measures $F_1(S), \cdots, F_n(S)$. Even though we cannot always compute the F_i measures, this does at least help us to understand why different methods are better on different kinds of problems. It does not seem reasonable to expect that one method will be optimal on all sets S of clauses. In fact, it may be possible to show that such an optimal method does not exist.

One interesting feature of the present work is it gives a theoretical justification for using a size bound with an inference method. For example, Otter [McC90] prefers small clauses. Our analysis shows how the use of a size bound can improve the asymptotic complexity of a method, and how the choice of the right size measure can also make a significant difference.

Some of the results we present are already known, and our contribution is to put them into a common perspective.

2.2 Proof Complexity measures

The *arity* of a function symbol is the number of arguments it takes. A *term* is a well-formed expression containing function symbols, constant symbols, and variables, as, $f(x, g(a))$ for f of arity 2, g of arity 1, and a of arity 0 (a constant symbol). An *atom* is a predicate symbol followed by a list of terms, as, $P(x, g(a, b))$. A *literal* is an atom or an atom preceded by a negation sign, as, $\neg Q(a, x)$. A literal preceded by a negation sign is *negative* and a literal without a negation sign is *positive*. The literals L and $\neg L$ are said to be *complementary*. A *clause* is a disjunction of literals, written as a set, as, $\{\neg P(x), Q(f(x))\}$. Free variablesin a clause are assumed to be universally quantified. Thus the clause $\{\neg P(x), Q(f(x))\}$ represents the formula $\forall x(\neg P(x) \vee Q(f(x)))$.

A term, literal, or clause not containing any variables is said to be *ground.* A clause containing only negative literals is *negative* and one containing only positive literals is *positive.* A clause containing only one literal is a *unit* clause. A set of clauses represents the conjunction of the clauses in the set. Thus the set $\{\{\neg P(x), Q(f(x))\}, \{\neg Q(y), R(g(y))\}, \{P(a)\}, \{\neg R(z)\}\}$ represents the formula $(\forall x(\neg P(x) \vee Q(f(x)))) \wedge (\forall y(\neg Q(y) \vee R(g(y)))) \wedge P(a) \wedge (\forall z \neg R(z))$.

Definition 2.2.1 *The* linear size s^{lin} *of a term, literal, clause, or set of clauses is its length, written as a character string (ignoring commas and parentheses within a term). Thus the linear size of the literal* $P(f(x), g(y))$ *is 5.*

Definition 2.2.2 *The subterm size* s^{sub} *of a term, literal, clause, or set of clauses is its number of distinct subterms, with duplicate occurrences of the same subterm counted only once. Thus the subterm size of the literal* $P(f(x), f(x))$ *is 3, since it has only the subterms* x, $f(x)$, *and* $P(f(x), f(x))$. *(We count the entire literal as a term, too.)*

Definition 2.2.3 *A* substitution *is a mapping from variables to terms which is the identity on all but finitely many variables. If* L *is a literal and* α *is a substitution, then* $L\alpha$ *is the result of replacing all variables in* L *by their image under* α. *We define the application of substitutions to terms, clauses, and sets of clauses similarly. A substitution* α *is a* unifier *of literals* L *and* M *if* $L\alpha = M\alpha$. *If such a substitution exists, we say that* L *and* M *are* unifiable. *A substitution* α *is a* most general unifier *of* L *and* M *if for any other unifier* β *of* L *and* M, *there is a substitution* γ *such that* $L\beta = L\alpha\gamma$ *and* $M\beta = M\alpha\gamma$.

Definition 2.2.4 *We say clause* $C\alpha$ *is a* factor *of clause* C *if* α *is a most general unifier of two literals of* C. *Also,* D *is a (simple)* resolvent *of* C_1 *and* C_2 *if* $D = ((C_1 - \{L_1\}) \cup (C_2 - \{L_2\}))\Theta$, *where* $L_1 \in C_1$ *and* $L_2 \in C_2$, *and* Θ *is a most general unifier of* L_1 *and* L_2. *The clause* D *is a* resolvent *of* C_1 *and* C_2 with factoring *if* $D = ((C_1 - B_1) \cup (C_2 - B_2))\Theta$, *where* $B_1 \subseteq C_1$ *and* $B_2 \subseteq C_2$, *and* Θ *is a most general substitution unifying* $\{L : L \in B_1\} \cup \{\neg L : L \in B_2\}$. *In both cases, the clauses* C_1 *and* C_2 *are* parents *of* D. *The resolution is said to be* positive *if one of the parents is a positive clause and* negative *if one of the parents is a negative clause. It is a* unit resolution *if one of the parents is a unit clause. It is an* A resolution *if the literals (or subsets) B_1 and B_2 of resolution are minimal in their respective clauses in a prespecified ordering on literals. For a discussion of these strategies, see [CL73, Fit90, Lov78, WOLB84].*

Definition 2.2.5 *A* resolution proof *from a set S of clauses is a sequence C_1, C_2, $\cdots$, C_n of clauses, where each C_i is either in S or is a resolvent of previous clauses in the sequence. The* length *of this proof is n. The* depth *of this proof is defined recursively, as follows: The depth of an input clause is 0. If C_i is a resolvent of C_j and C_k, then the depth of C_i is one plus the maximum depth of C_j and C_k. The depth of the proof is the maximum depth of any C_i in it. This proof is a* refutation *if it contains the empty clause. A* P_1-deduction proof *is a resolution proof in which every resolution is positive. A* negative resolution proof *is a resolution proof in which every resolution is negative. A* unit resolution proof *is a resolution proof in which every resolution is a unit resolution. An* A-resolution proof *is a resolution proof in which every resolution is an A-resolution.*

Definition 2.2.6 *A (refutational) theorem proving method is* complete *if for every unsatisfiable clause set S, there is a proof that S is unsatisfiable using the method.*

It is known that resolution is complete [Rob65b]. Many of the refinements of resolution are also complete, including P_1-deduction, negative resolution, and A-resolution. Unit resolution, however, is not complete.

For those who may not be familiar with the refutational style of theorem proving, we give an example. Suppose that we want to show that the first-order formula $(\forall x \exists y\ (P(x) \supset Q(y))) \wedge (\forall y \exists z\ (Q(y) \supset R(z))) \supset (\forall x \exists z\ (P(x) \supset R(z)))$ is valid. Here $\supset$ represents logical implication. In the refutational approach, we negate this formula to obtain $\neg[(\forall x \exists y(P(x) \supset Q(y))) \wedge (\forall y \exists z(Q(y) \supset R(z))) \supset (\forall x \exists z(P(x) \supset R(z)))]$, and show that this formula is unsatisfiable. This is translated into clause form by rearranging the Boolean connectives and replacing existential quantifiers by new function symbols, called *Skolem functions.* By this means, we obtain the formula $(\forall x \exists y(P(x) \supset Q(y))) \wedge$

$(\forall y \exists z(Q(y) \supset R(z))) \wedge (\exists x \forall z(P(x) \wedge \neg R(z)))$, that is, $(\forall x \exists y(P(x) \supset Q(y))) \wedge (\forall y \exists z(Q(y) \supset R(z))) \wedge (\exists x P(x)) \wedge \forall z \neg R(z)$. Inserting Skolem functions, we obtain $(\forall x(P(x) \supset Q(f(x)))) \wedge (\forall y(Q(y) \supset R(g(y)))) \wedge P(a) \wedge \forall z \neg R(z)$. This translation is satisfiability preserving. Translating this formula into a set S of clauses, we obtain $\{\{\neg P(x), Q(f(x))\}, \{\neg Q(y), R(g(y))\}, \{P(a)\}, \{\neg R(z)\}\}$. The variables are implicitly regarded as universally quantified. We then have the following resolution refutation:

1. $P(a)$	(input)
2. $\neg P(x), Q(f(x))$	(input)
3. $Q(f(a))$	(resolution, 1,2)
4. $\neg Q(y), R(g(y))$	(input)
5. $R(g(f(a)))$	(3,4, resolution)
6. $\neg R(z)$	(input)
7. **false**	(5,6, resolution)

The designation "input" means that a clause is in S. Since **false** has been derived from S by resolution, we have proven that S is unsatisfiable, and so the original first-order formula is valid.

We now resume our discussion of the complexity of theorem proving strategies.

Definition 2.2.7 *The* proof length complexity measure $M_{pl}(S)$ *for unsatisfiable clause sets S is the minimum length of any resolution proof of the empty clause (i.e,* **false***) from S. The* proof depth complexity measure $M_{pd}(S)$ *of S is the minimum depth of any resolution proof of the empty clause from S.*

We note that although the depth measure is inference based, it can be viewed in a non-inference based manner, as suggested by [PZ]. It is equivalent to the depth of the minimal closed binary semantic tree over S (for the definition of this, see for example [CL73]).

In general, the proof complexity measures that are not inference based are *instance based*, that is, they are based on Herbrand's theorem. Herbrand's theorem says that if a set S of clauses is unsatisfiable, then there is an unsatisfiable set T of ground instances of the clauses in S. We call such a set T a *Herbrand set* for S. We obtain instance based complexity measures uniformly in the following way:

Definition 2.2.8 *We use $|S|$ to refer to the number of elements in a set S, that is, its cardinality.*

Theorem 2.2.9 *Suppose that f is a computable (i.e., recursive) function of ground clause sets, and g is a computable function such that for all clause sets S and all Herbrand sets T for S, there is a a Herbrand set T' for S such that $f(T) = f(T')$ and $s^{lin}(T') \leq g(f(T), S)$. Let $F(S)$ for unsatisfiable clause sets S be the minimum value of $f(T)$ such that T is a Herbrand set for S. Then F is a theorem proving complexity measure.*

Proof. We first show that, given n, it is possible to test if S has a Herbrand set T such that $f(T) = n$. For this, it is only necessary to examine all Herbrand sets T' such that $s^{lin}(T') \leq g(f(T), S)$, that is, $s^{lin}(T') \leq g(n, S)$. Since g is recursive, this is decidable. It follows that we can find the minimal n such that S has a Herbrand set T with $f(T) = n$ by enumerating n and applying the preceding procedure. Such a minimal n is by definition $F(S)$. This procedure runs in time recursive in $F(S)$, since g is recursive and one can test unsatisfiability in recursive (exponential) time. Therefore, one can prove theorems in time recursive in $F(S)$. Similarly, testing if $F(S) \leq n$ is decidable, since we can compute $F(S)$ if S is unsatisfiable.

□

We will need to derive some properties of the subterm size measure s^{sub}.

Theorem 2.2.10 *Suppose L and M are unifiable atoms and Θ is a most general unifier of L and M. Then $s^{sub}(\{L\Theta, M\Theta\}) \leq s^{sub}(L) + s^{sub}(M)$.*

Proof. By considering the working of a unification algorithm, we can express Θ as a composition of substitutions $\alpha_1\alpha_2 \cdots \alpha_n$, where each α_i binds one variable x_i in $\{L\alpha_1\alpha_2 \cdots \alpha_{i-1}, M\alpha_1\alpha_2 \cdots \alpha_{i-i}\}$ to a term t_i in $\{L\alpha_1\alpha_2 \cdots \alpha_{i-1}, M\alpha_1\alpha_2 \cdots \alpha_{i-i}\}$ not containing x_i. Let Θ_i be $\alpha_1\alpha_2 \cdots \alpha_i$. We consider the effect that α_i has on the subterm size of $\{L\Theta_{i-1}, M\Theta_{i-1}\}$, that is, what is the subterm size of $\{L\Theta_i, M\Theta_i\}$.

Each subterm u of $\{L\Theta_{i-1}, M\Theta_{i-1}\}$ is replaced by $u\alpha_i$, so the number of subterms is not thereby increased. Also, the subterms in t_i are added to the set of subterms of $\{L\Theta_{i-1}, M\Theta_{i-1}\}$. However, since t_i is a subterm of $\{L\Theta_{i-1}, M\Theta_{i-1}\}$, and t_i does not contain x, t_i and all its subterms still occur in $\{L\Theta_i, M\Theta_i\}$. Thus $s^{sub}(\{L\Theta_i, M\Theta_i\}) \leq s^{sub}(\{L\Theta_{i-1}, M\Theta_{i-1}\})$.

□

Corollary 2.2.11 *Suppose C is a clause and D is a factor of C. Then $s^{sub}(D) \leq s^{sub}(C)$.*

Proof. A factoring operation is a unification between literals of C. Reasoning as in the theorem, the result follows.

□

Corollary 2.2.12 *Suppose D is a resolvent of C_1 and C_2. Then $s^{sub}(D) \leq s^{sub}(C_1) + s^{sub}(C_2)$.*

Proof. A resolution can be expressed as a sequence of factorings followed by a unification of two literals. Assuming that C_1 and C_2 have no common variables, we can regard a resolution of C_1 and C_2 as a sequence of factorings on $C_1 \cup C_2$ followed by a unification of complementary literals and the deletion of two literals. The result then follows by the preceding corollary and the fact that $s^{sub}(C_1 \cup C_2) \leq s^{sub}(C_1) + s^{sub}(C_2)$.

□

Lemma 2.2.13 *If t is a term, then $s^{lin}(t) \leq 1 + r + r^2 + \ldots + r^{s^{sub}(t)}$, where r is the maximum arity of any function symbol in t.*

Proof. Let d be the depth of t, and let r be the maximum arity of any function symbol in t. Then $s^{sub}(t) \geq d$, since there has to be at least one distinct subterm of each depth, and $s^{lin}(t) \leq 1 + r + r^2 + \ldots + r^d$, adding the numbers of symbols at different depths. Therefore $s^{lin}(t) \leq 1 + r + r^2 + \ldots + r^{s^{sub}(t)}$.

□

Lemma 2.2.14 *If S is a set of clauses, then $s^{lin}(S) \leq 5(1 + r + r^2 + \cdots + r^{s^{sub}(S)})$, where r is the maximum arity of any function or predicate in S. Note that $r \leq s^{lin}(S)$.*

Proof. For literals L, $s^{lin}(L) \leq 1 + r + r^2 + ... + r^{s^{sub}(L)}$ as for terms, treating a predicate symbol as a function symbol. For clauses C, $s^{lin}(\{L_1, L_2, \cdots, L_n\}) \leq (n+1) + \Sigma_i s^{lin}(L_i)$, since we only have to add commas and parentheses to express a clause as a set of literals. For sets S of clauses, $s^{lin}(\{C_1, C_2, \cdots, C_n\}) \leq (n+1) + \Sigma_i s^{lin}(C_i)$ for the same reason. By simple reasoning, one can then show that if S is a set of clauses, $s^{lin}(S) \leq 5(1 + r + r^2 + \cdots + r^{s^{sub}(S)})$ in general. This is because the worst case is when there is only one very large literal, that is, $S = \{\{L\}\}$ for some literal L.

□

Definition 2.2.15 *Suppose S is a set of clauses and S' is a set of copies of the clauses in S with variables renamed in each copy. We call S' an* amplification *of S.*

Definition 2.2.16 *We define the* duplication measure *$M_{dup}(S)$ for S to be the minimal number of clauses $|T|$ in any Herbrand set T for S. We define $M'_{dup}(S)$ to be the minimum of $s^{sub}(S')$ over all amplifications S' of S such that there exists a Θ such that $S'\Theta$ is ground and unsatisfiable.*

Theorem 2.2.17 *If set S of clauses is unsatisfiable and T is a Herbrand set for S, then there is an amplification S' of S and a substitution Θ such that $T = S'\Theta$. (This essentially means that S' has as many copies of each clause in S as there are instances of it in T.)*

Proof. A simple consequence of Herbrand's theorem.

□

Lemma 2.2.18 *Suppose S' is an amplification of S and Θ_1 and Θ_2 are substitutions such that $S'\Theta_1$ and $S'\Theta_2$ are ground. Suppose for all literals L_1 and L_2 in clauses of S', $L_1\Theta_1$ and $L_2\Theta_1$ are identical (complementary) iff $L_1\Theta_2$ and $L_2\Theta_2$ are identical (complementary). Then $S'\Theta_1$ is unsatisfiable iff $S'\Theta_2$ is unsatisfiable.*

Proof. Under these assumptions, one can show how a model for $S'\Theta_1$ may be obtained from a model for $S'\Theta_2$ and vice versa.

□

Theorem 2.2.19 *Suppose that S is an unsatisfiable set of clauses and T is a Herbrand set for S. Then there is a Herbrand set T' for S such that $s^{sub}(T') \leq |T|s^{sub}(S)$. Furthermore, if S' is an amplification of S and Θ is a substitution such that $T = S'\Theta$, then such a T' may be obtained by a number of unifications on S' that is linear in the number of literals in S', followed by a replacement of variables by a constant symbol.*

Proof. Let γ be a most general substitution such that for two literals L_1 and L_2 in S', $L_1\gamma$ and $L_2\gamma$ are identical (or complementary) iff the corresponding literals $L_1\Theta$ and $L_2\Theta$ in T are identical (or complementary). Such a γ may be obtained by a sequence of unifications on atoms of S', at most $m-1$ unifications in all, where m is the number of literals in S'. Each such unification does not increase the subterm size of S', as we showed in [PSK95] and above in theorem 2.2.10. Therefore $s^{sub}(S'\gamma) \leq s^{sub}(S') \leq |T|s^{sub}(S)$. Also, let γ' be $\gamma\delta$ where δ replaces all remaining variables by a fixed constant symbol. Then by lemma 2.2.18, $S'\gamma'$ is unsatisfiable. Therefore $S'\gamma'$ is a Herbrand set for S, and $s^{sub}(S'\gamma') = s^{sub}(S'\gamma)$.

□

The preceding theorem suggests that it would be reasonable to define the duplication measure to be $s^{sub}(S')$, that is, M'_{dup}, rather than $|T|$.

Theorem 2.2.20 *The duplication measure is a theorem proving complexity measure.*

Proof. This proof is not obvious, which makes this measure somewhat interesting. We need to show that a g as in theorem 2.2.9 exists. The idea is that if we know there is a Herbrand set having only n clauses, then we can bound the linear size of the clauses in some Herbrand set recursively in n and S. Suppose S is a set of clauses and T is a Herbrand set for S with n clauses. By theorem 2.2.19, there is a Herbrand set T' for S such that $s^{sub}(T') \leq |T|s^{sub}(S)$. We need to show that there is a computable g such that $s^{lin}(T') \leq g(|T|, S)$. By lemma 2.2.13 above, it suffices to show that there is a recursive g' such that $s^{sub}(T') \leq g'(|T|, S)$. However, this follows from the fact that $s^{sub}(T') \leq |T|s^{sub}(S)$, since we can let $g'(n, S)$ be $ns^{sub}(S)$.

□

We note that there are a number of other reasonable ways one could define this duplication complexity measure. One could also define it based on the maximum number of copies of any one clause in S', or (as noted) M'_{dup}. Each of these two additional measures would have slightly different implications for theorem proving complexity.

One reason we feel that the duplication measure is interesting (apart from the fact that it is not inference based) is the fact that it incorporates unification. By this we mean that the sizes of the instances in a Herbrand set can be much larger than the sizes of the clauses in S, but the duplication measure M_{dup} only counts the number of clauses in a Herbrand set. Most theorem provers generate the extra term structure in a Herbrand set using unification.

Definition 2.2.21 *The* linear complexity measure $M_{lin}(S)$ *of a set S of clauses is the minimum, over all Herbrand sets T for S, of the maximum linear size of a literal in T. The* subterm complexity measure $M_{sub}(S)$ *of a set S of clauses is the minimum, over all Herbrand sets T for S, of the maximum subterm size of a literal in T.*

It is straightforward to show that these are theorem proving complexity measures. Note that we could also define these based on the linear or subterm size of the largest clause in T, or the linear or subterm size of T itself. This gives us four more plausible theorem proving complexity measures.

Definition 2.2.22 *The* time complexity of a theorem proving strategy *with respect to a complexity measure F is the time taken by the strategy on an input S, as a function of $F(S)$. The* space complexity of a theorem proving strategy *with respect to a complexity measure F is the space used, as a function of $F(S)$.*

As noted above, the time (or space) taken by a prover to find a proof is another complexity measure (if the prover is complete), and so in this definition we are really considering a relationship between complexity measures.

Definition 2.2.23 *The* measure estimation problem *for a complexity measure F is the set of pairs (S, n), where n is in unary, such that $F(S) \leq n$.*

We can ask what is the time or space complexity of recognizing this set. This is a purely complexity theoretic problem, which in some sense tells us how hard it is to prove theorems with respect to this complexity measure.

Definition 2.2.24 *The* inherent (time or space) complexity *of a complexity measure F is the complexity of deciding the measure estimation problem for F.*

Definition 2.2.25 *Suppose that f and g are functions of clause sets S. We say that the function $g(S)$ is* (upper bounded by a) single exponential *in $f(S)$ (or g is exponential with respect to f) if $g(S)$ is upper bounded by a function of the form*

$$c * 2^{p(s^{lin}(S), f(S))},$$

where p is a polynomial in two variables, and we write the set of such functions as $exp(f)$. We say that $g(S)$ is lower bounded by a single exponential *in $f(S)$ if $g(S)$ is lower bounded by some function of the form*

$$c * 2^{d * f(S)^{\epsilon}}$$

or

$$c * 2^{d * s^{lin}(S)^{\epsilon}},$$

for some $c, d, \epsilon > 0$. We say a function $g(S)$ is (upper bounded by a) double exponential *in $f(S)$ (or g is double exponential with respect to f) if $g(S)$ is upper bounded by a function of the form*

$$c * 2^{2^{p(s^{lin}(S), f(S))}},$$

with similar assumptions on p. We write this set of functions as $exp^2(f)$ *or* $dexp(f)$*. We say that* $g(S)$ *is* lower bounded by a double exponential *in* $f(S)$ *if* $g(S)$ *is lower bounded by some function of the form*

$$c * 2^{2^{d*f(S)^\epsilon}}$$

or

$$c * 2^{2^{d*s^{lin}(S)^\epsilon}},$$

for some $c, d, \epsilon > 0$*. We define triple, quadruple, and quintuple exponential similarly. The idea is that polynomials in* $s^{lin}(S)$ *for a clause set* S *may be regarded as constants, with respect to the polynomial* p*.*

We now clarify to some extent some relationships between the inherent complexity of a complexity measure and the complexity of a particular theorem prover with respect to a measure:

Theorem 2.2.26 *Suppose a prover finds proofs from sets* S *of clauses in time* $t(F(S))$*, where* F *is a theorem proving complexity measure. If* F *is inference-based and the prover finds proofs that are minimal with respect to* F *and* F *has inherent time complexity bounded below by* t'*, that is,* $\Omega(t')$*, then there exists a constant* $c > 0$ *such that the prover requires* $\Omega(t'(|S| + F(S)) - c * F(S))$ *time to find proofs from* S*.*

Proof. Suppose solving the measure estimation problem for (S, n) takes time at least $t'(|S|+n)$, that is, $\Omega(|S|+n)$ time. Now, we can solve the measure estimation problem for (S, n) by calling the prover on S, computing $F(S)$, and testing if $F(S) \leq n$. This takes time $t(F(S))+cn$ (cn for the comparison). Thus $t(F(S)) + cn$ is $\Omega(t'(|S| + n))$. Letting n be $F(S)$ and rearranging the inequality, we obtain that $t(F(S))$ is $\Omega(t'(|S| + F(S)) - c * F(S))$.

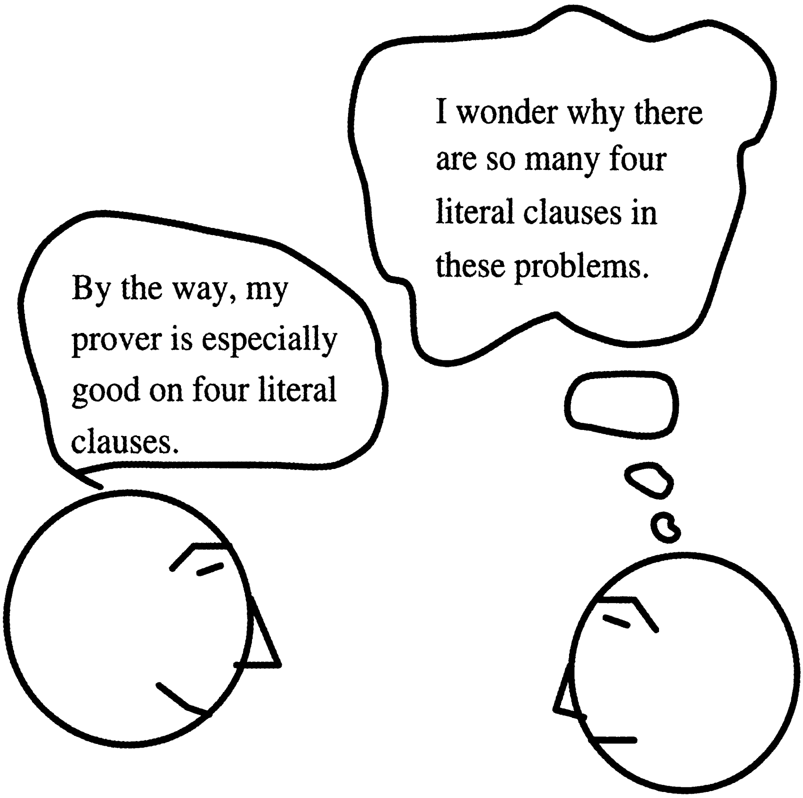
$\square$

For example, if F is the proof depth measure and a prover does breadth-first search, it will find a minimal depth proof and therefore the prover can calculate F. Thus we can lower bound the time for depth-bounded (breadth-first) theorem proving using the inherent complexity of the proof depth complexity measure.

Theorem 2.2.27 *If the measure estimation problem for F is solvable in time t, then there is a theorem prover with time complexity t' relative to F, where $t'(F(S)) = \Sigma_{i=1}^{F(S)} t(|S| + i)$.*

Proof. We can recognize in time $t(|S|+n)$ if $F(S) \leq n$, where n is in unary. We can do theorem proving by testing if $F(S) \leq 1$, $F(S) \leq 2$, ... If any of these tests succeed, we know that S is unsatisfiable. We know that the test will succeed when i becomes $F(S)$. The total time taken is then $t(|S| + 1) + ... + t(|S| + F(S))$, since t is a function of the length of the input.

□


By the way, my prover is especially good on four literal clauses.
I wonder why there are so many four literal clauses in these problems.

2.3 Inherent complexities

We now consider the inherent complexities of various complexity measures. Some previous studies of the complexities of various resource-bounded theorem proving problems [Pla84, Gou94] are relevant here. We mention some of these results, and state their relevance at the end of this section.

Theorem 2.3.1 *The problem of determining whether a depth d binary resolution proof exists from a set S of first-order clauses is NEXPTIME complete, where d is represented in unary [Pla84].*

Definition 2.3.2 *A* Horn clause *is a clause containing at most one positive literal. Thus $\{\neg P, \neg Q, R\}$ is a Horn clause. A* Horn set *is a set of Horn clauses.*

Theorem 2.3.3 *The problem of determining whether a binary resolution proof with length n or less exists from a set S of propositional Horn clauses is NP-complete, where n is represented in unary [Pla84].*

From theorem 2.3.3 we derive a corresponding NP-completeness result for first-order logic.

Theorem 2.3.4 *The problem of determining whether a binary resolution proof with length n or less and maximum clause size c or less exists from a set S of first-order clauses is NP-complete, where n and c are represented in unary.*

Proof. For propositional problems, we can choose c to be the length of the input, and we know that no clause longer than this will be generated by resolution. The NP-hardness result follows then from theorem 2.3.3. For first-order clause sets, we can show that this problem is still in NP, since we can nondeterministically generate n resolvents, each of which has size c or less. Using a polynomial time unification algorithm, the problem is solvable in nondeterministic polynomial time.

Note that we need to include the maximum clause size as an input, otherwise we cannot obtain the NP membership of the problem. This is due to the fact that resolution for first-order logic is not polynomially transparent [Let93], that is, clauses of size exponential in n are possibly needed for n step resolution proofs.

□

Theorem 2.3.5 *The problem of determining whether a set S of first-order clauses has a Herbrand set T with $|T| = d$ is Σ_2^p complete, where d is represented in unary [Gou94].*

Theorem 2.3.6 *The satisfiability problem for Schonfinkel-Bernays form formulas is NEXPTIME complete [Pla84].*

Theorem 2.3.7 *The problem of determining whether a set S of first-order clauses has a Herbrand set T such that every clause D in T has linear size $s^{lin}(D) \leq n$ is co-NEXPTIME hard, where n is represented in unary.*

Proof. This is a consequence of theorem 2.3.6 by observing that formulas in Schonfinkel-Bernays form do not have function symbols, and therefore the linear size of D for such clause sets is bounded by the linear size of clauses in S.

□

We now relate the completeness results of resource-bounded theorem proving to our study of complexity measures and complexity analysis. We draw a correspondence between theorem 2.3.1 and complexity analysis with respect to the proof depth measure. A theorem proving strategy is a *minimal strategy with respect to proof depth* if the depth of its proof always equals the proof depth measure, namely, the depth of the minimal binary resolution proof. To verify a proof depth measure of a set of clauses S, we can use a minimal strategy with respect to proof depth, and compute the depth of its proof. Theorem 2.3.1 states that the problem of verifying the proof depth measure of a set of clauses is NEXPTIME complete. Assuming that NP-hard problems require exponential time, all minimal strategies with respect to proof depth are likely to have no less than double exponential complexity with respect to the depth measure. A similar argument can be made for the other three complexity measures. Minimal strategies with respect to the proof length measure, duplication measure, and linear size measure are likely to have no less than exponential, exponential, and double exponential complexity with respect to the corresponding complexity measures. Note that the requirement of minimal strategy is needed. For example, to determine the proof length measure for propositional Horn clauses is NP-complete, but hyper-resolution solves the propositional Horn satisfiability problem in polynomial time. However, hyper-resolution is not a minimal strategy with respect to proof length. Note that the proof length measure is defined based on binary resolution.

These results give bounds on what is the best possible, at least for a minimal strategy. We now analyze the complexity of various strategies with respect to various measures and can compare them to the above bounds to get an idea of how close to optimal they are. Independent of that, we can compare the strategies with each other to get another measure of how good they are.

2.4 Inference Based Measures

First we give some interesting sets of clauses that cause various theorem proving strategies to exhibit various kinds of exponential behavior. These will be used to give lower bound on their complexity with respect to various measures. The upper bounds will be given based on general arguments.

Definition 2.4.1 *Clause C is a* hyper-resolvent *of clauses $B_1, \ldots, B_k$ if C is positive and may be obtained by a sequence of positive resolutions from $B_1, \ldots, B_k$ in which no earlier resolvent is positive. Thus $\{R, T\}$ is a hyper-resolvent of the clauses $\{P, T\}$, $\{Q\}$, and $\{\neg P, \neg Q, R\}$. A* hyper-resolution proof *from S is a sequence $C_1, C_2, \ldots, C_n$ of clauses in which each C_i is either in S or is a hyper-resolvent of earlier clauses in the sequence. Clause C is a* UR resolvent *of clauses $B_1, \ldots, B_k$ if C is a unit and may be obtained by a sequence of unit resolutions from $B_1, \ldots, B_k$ in which no earlier resolvent is a unit. Thus $\{R\}$ is a UR-resolvent of the clauses $\{P\}$, $\{Q\}$, and $\{\neg P, \neg Q, R\}$. A* UR-resolution proof *from S is a sequence $C_1, C_2, \ldots, C_n$ of clauses in which each C_i is either in S or is a UR-resolvent of earlier clauses in the sequence. A set S of clauses is* unit-resolvable *or* UR-resolvable *if it has a UR-resolution refutation.*

Definition 2.4.2 *The clause set S_{fg} consists of the two clauses $\{\neg P(x), P(f(x))\}$ and $\{\neg P(x), P(g(x))\}$.*

Theorem 2.4.3 *It is possible to generate 2^{2^n} clauses whose literals are of linear size $2^n + 2$ using depth n binary resolution proofs from S_{fg}.*

Proof. By induction. For $n = 0$ we observe that S_{fg} has two clauses whose literals are of size 3. For $n = 1$ we obtain the following clauses: $\{\neg P(x), P(f(f(x)))\}$, $\{\neg P(x), P(f(g(x)))\}$, $\{\neg P(x), P(g(f(x)))\}$, $\{\neg P(x), P(g(g(x)))\}$. This gives four clauses of literal size 4. In general, the number of function symbols at depth n is 2^n, so the literal size is $2^n + 2$ and the number of clauses possible is 2^{2^n}. Actually, even more clauses are derivable by resolving clauses at depths i and j, for $i \neq j$, but the given bounds are asymptotically still optimal.

□

Definition 2.4.4 *The clause set T_{fg} consists of the three clauses $\{\neg T(x,y), \neg T(y,z), T(x,z)\}$, $\{T(x, f(x))\}$, and $\{T(x, g(x))\}$.*

Theorem 2.4.5 *It is possible to generate 2^{2^n} unit clauses having literals with 2^n function symbols using depth n hyper-resolution proofs from T_{fg}.*

Proof. By induction. For $n = 0$ we have two unit clauses, each with one function symbol. With one hyper-resolution step, we can generate the four units $\{T(x, f(f(x)))\}$, $\{T(x, f(g(x)))\}$, $\{T(x, g(f(x)))\}$, and $\{T(x, g(g(x)))\}$. With depth two proofs, we generate units having four function symbols, sixteen in all. In general, at depth n we generate all possible units having 2^n function symbols, 2^{2^n} in all. As before, even more clauses are derivable by resolving clauses at depths i and j, for $i \neq j$, but the stated bounds are asymptotically still optimal.

□

Similar bounds are derivable for P_1-deduction (positive resolution), since a hyper-resolution corresponds to two P_1-deductions. We now consider how one may derive many clauses having large numbers of small literals, rather than a few large literals.

Definition 2.4.6 *Let T_{trans} be the clause set consisting of the eight clauses $\{\neg T_i(x,y),\ \neg T_j(y,z),\ T_k(x,z)\}$ for i,j,k in $\{1,2\}$.*

Theorem 2.4.7 *One can derive $2^{2^{n-1}+2}$ clauses having $2^{n-1}+2$ literals with binary resolution proofs of depth n from T_{trans}.*

Proof. By induction. For $n = 1$, we derive eight clauses with three literals. For length two, we resolve two clauses having three literals and obtain the following four-literal clauses:
$\{\neg T_i(x,y), \neg T_j(y,z), \neg T_k(z,w), T_l(x,w)\}$.

For length three, we can resolve these clauses to obtain the following 64 six-literal clauses:
$\{\neg T_i(x,y), \neg T_j(y,z), \neg T_k(z,w), \neg T_l(w,v), \neg T_m(v,u), T_n(x,u)\}$

In general, for length $n+1$, we resolve two clauses having $2^{n-1}+2$ literals and obtain a clause having 2^n+2 literals. (Two of the literals are deleted by the resolution operation.)

□

Theorem 2.4.8 *It is possible to derive an exponential number of one-literal clauses from the input set $S_{fg} \cup \{P(a)\}$ using length n P_1 deductions.*

Proof. A simple induction. In a length n proof, one can derive clauses having a linear number of function symbols, of which there are an exponential number (that is, exponential in n).

□

Theorem 2.4.9 *If there is a P_1 deduction proof of length n from S, then there is a hyper-resolution proof of length between $n/(s^{lin}(S))$ and n. If there is a hyper-resolution proof of length n, then there is a P_1 deduction proof of length between n and $ns^{lin}(S)$.*

Proof. Each hyper-resolution step corresponds to k P_1 steps, where k is bounded by the maximum number of negative literals in a clause in S, and k is in turn bounded by $s^{lin}(S)$ (the length of S in characters). Also, the number of negative literals in a clause never increases during P_1 deduction.

□

It will turn out that P_1 deduction is simpler to analyze than hyper-resolution. However, using the above theorem, many results about P_1-deduction apply immediately to hyper-resolution also, especially asymptotic complexity results. This equivalence between the two will often be taken for granted. We now introduce a set of clauses that encodes a binary counter; this clause set will be useful in the following discussion.

Definition 2.4.10 *The* binary counter *clause set B_n consists of the following clauses:*

$\{p(X_1, X_2, \ldots, X_{n-1}, 1), \neg p(X_1, X_2, \ldots, X_{n-1}, 0)\}$
$\{p(X_1, X_2, \ldots, X_{n-2}, 1, 0), \neg p(X_1, X_2, \ldots, X_{n-2}, 0, 1)\}$
...
$\{p(1, 0, 0, \ldots, 0), \neg p(0, 1, 1, \ldots, 1)\}$
$\{p(0, 0, 0, \ldots, 0)\}$
$\{\neg p(1, 1, 1, \ldots, 1)\}$

Theorem 2.4.11 *Any binary resolution refutation for the binary counter B_n has a proof depth (and length) at least $n+1$. Also, there is a binary resolution proof of depth $2n$ and length $2n+1$.*

Proof. We first consider Herbrand sets T for B_n. Such a Herbrand set T must have $2^n - 1$ clauses in it of the form

$$\{p(a_1, a_2, \ldots, a_i, 1, 0, \ldots, 0), \neg p(a_1, a_2, \ldots, a_i, 0, 1, \ldots, 1)\}$$

where the a_i are either 0 or 1. We can show that all these ground clauses are needed by the fact that if any of them are missing we can construct a model of the remaining ground instances of B_n over its Herbrand universe. We also have two unit clauses in the proof, namely,

$\{p(0,0,0,\ldots,0)\}$ and $\{\neg p(1,1,1,\ldots,1)\}$, for 2^n+1 ground clauses in all.

Now, any binary resolution proof from T has to have depth $n+1$ in order to include all of these 2^n+1 ground clauses, since a binary tree of depth n can only have 2^n leaves. Lifting to the first-order case, any binary resolution proof from B_n also has to have depth at least $n+1$. This proof therefore has length at least $n+2$.

To obtain a proof of depth $2n$, we successively derive the clauses of the form

$$\{\neg p(X_1,X_2,\ldots,X_i,0,0,\ldots,0), p(X_1,X_2,\ldots,X_i,1,1,\ldots,1)\}$$

for $i=n,n-1,\ldots,1$. This can be done in two resolutions from the clauses

$$\{\neg p(X_1,X_2,\ldots,X_{i+1},0,0,\ldots,0), p(X_1,X_2,\ldots,X_{i+1},1,1,\ldots,1)\}$$

and

$$\{\neg p(X_1,X_2,\ldots,X_i,0,1,\ldots,1), p(X_1,X_2,\ldots,X_i,1,0,\ldots,0)\},$$

the latter of which is in B_n. Therefore the clause

$$\{\neg p(0,0,\ldots,0), p(1,1,\ldots,1)\}$$

can be derived at depth $2n-2$, and a contradiction can be found at depth $2n$. We observe that this proof also has length $2n+1$.

□

Theorem 2.4.12 *Any P_1 deduction refutation from B_n has depth at least 2^n and length at least 2^n+1. Also, there is a proof at depth 2^n with length 2^n+1.*

Proof. By examining the clauses that may be generated. One starts with $p(0,0,0,\ldots,0)$ and generates the binary sequences in order, with one more resolution at the end to generate the empty clause. This is 2^n resolution steps in all, for a length of 2^n+1 and a depth of 2^n. No other steps are possible, so no shorter proof exists.

□

Theorem 2.4.13 *The number of atoms over k predicate / function / constant symbols, and having subterm size n, is bounded by $(2n+k)^{nr}$, where r is the maximum arity of any predicate or function symbol. We are assuming the variables are chosen from the set $x_1, x_2, \ldots, x_n$.*

Proof. The subterms of an atom can be given integer labels by their first occurrence in the atom, reading left to right. Thus for $P(f(g(a,b)))$, the term $f(g(a,b))$ would have index 1, the term $g(a,b)$ would have index 2, the term a would have 3, and the term b would have index 4. Later occurrences of a term can then be replaced by the integer index, so we can represent $P(g(f(a), f(a)))$ as $P(g(f(a), 2))$ where the 2 indicates an occurrence of the term $f(a)$ having index 2. The number of occurrences of predicate, function, and constant, symbols in this representation is then the subterm size of the atom. If the maximum arity is r, then the number of occurrences of function symbols and integer indices is at most nr. Each one of these nr positions can be either a function symbol, a constant, or an integer between 1 and n, $n+k$ possibilities in all. Thus there are at most $(n+k)^{nr}$ ground terms of subterm size n. If one also considers variables, there can be at most n variables since each variable is a term, and we get at most $(2n+k)^{nr}$ terms. These bounds are all of the form $c^{n\log(n)}$, and are therefore still single exponential.

□

Corollary 2.4.14 *The number of clauses over k function/constant and predicate symbols, and having subterm size n, is bounded by $(2n+k+1)^{2nr}$, where r is the maximum of 2 and the arities of all function and predicate symbols. We are assuming as before that variables are chosen from the set $x_1, x_2, \ldots, x_n$. We are also only counting the subterms appearing in the literals, as well as the literals themselves, in the subterm size. The number of sets of clauses over k function/constant and predicate symbols, and having subterm size n, is bounded by $(2n+k+2)^{4nr}$, under the same assumptions.*

Proof. We can consider a clause $\{L_1, L_2, \ldots, L_p\}$ as having a top-level binary "or" connective, that is, $or(L_1, or(L_2, \ldots, or(L_{n-1}, L_n)\ldots))$ and we can consider the predicate symbols as function symbols as before. We can assume that all literals are distinct, since a clause is a set. Therefore, the number of distinct subterms in the literals is at least p. There are at most $p - 1$ extra subterms added by the binary "or" connective, and $p - 1 < n$. Thus the clause may have $2nr$ symbols, and each symbol can be one of the $2n + k$ symbols for an atom, or the "or" connective. This gives $(2n + k + 1)^{2nr}$ altogether. For sets S of clauses, we can add an "and" connective between the clauses. This can double the length again, and adds one more symbol. Therefore, we obtain the stated bound of $(2n + k + 2)^{4nr}$ clauses having subterm size n.

□

In general, we need ways of describing various search strategies for a given theorem proving strategy.

Definition 2.4.15 *We say that a search strategy is* depth-bounded *if it generates proofs in order of increasing depth,* length-bounded *if it generates proofs in order of increasing length,* subterm-size bounded *if it generates clauses in order of increasing subterm size, and* linear-size bounded *if it generates clauses in order of increasing linear size. To clarify this further, a subterm-size (linear-size) bounded search strategy will do the following: For all i from 1 to infinity until a proof is found, generate all proofs in which all clauses have subterm size (linear size) bounded by i.*

2.4.1 Resolution Based Methods

Theorem 2.4.16 *Depth-bounded binary resolution has a complexity that is at worst double exponential with respect to the proof depth complexity measure M_{pd}.*

Proof. We note that if D is a resolvent of C_1 and C_2, then $|D| \leq |C_1| + |C_2|$ where $|C|$ is the number of literals in a clause C. Therefore the number of literals in a clause can at worst double in each round of resolution. Therefore in a depth n binary resolution proof, the number of literals in clauses can be at most exponential in n. This implies at most double exponentially many resolvents between any two clauses (with factoring). Furthermore, we note by corollary 2.2.12 that if D is a resolvent of C_1 and C_2, with or without factoring, then the subterm size of D is at most the sum of the subterm sizes of C_1 and C_2. Therefore the subterm sizes of clauses occurring in depth n proofs is at worst exponential in n. By corollary 2.4.14 above, this means that there are only double exponentially many clauses that can occur in such proofs. Thus the total search can be done in double exponential time.

□

Theorem 2.4.17 *Depth-bounded binary resolution has a search complexity that is double exponential with respect to the proof depth complexity measure M_{pd}.*

Proof. The upper bound was just given. We now give a lower bound. Consider the set of clauses $S_{fg} \cup B_n$. We showed in theorem 2.4.11 that B_n requires a depth $n+1$ binary resolution proof to find a refutation, and there is a refutation at depth $2n$. Doing depth-bounded (breadth-first) resolution on the clause set S_{fg} will generate a number of clauses that is double exponential in n, as shown in theorem 2.4.3.

□

Theorem 2.4.18 *Length bounded binary resolution (without factoring) has single exponential complexity with respect to the proof length complexity measure M_{pl}, if resolution and factoring are separated.*

Proof. By resolution without factoring we mean that resolution and factoring are considered as separate operations in the proof. We want to consider the total number of proofs of length n and show that this is exponential in n. We also want to show that we can generate each such proof in time exponential in n. In addition, we need to give a corresponding lower bound.

Now, suppose that the input clause set S has a maximum of c literals per clause. We want to count the number of possible resolutions at step i when we have already generated the clauses $C_1, C_2, \ldots, C_{i-1}$. We have to choose two clauses to resolve, which can be one of the clauses in the proof or a clause in S. There are $|S| + i - 1$ possibilities at most. Then we have to choose two literals to resolve. The number of literals can at most double, so the number in any clause is bounded by $2^{i-1}c$. Therefore there are at most $(|S| + i - 1)^2 * (2^{i-1}c)^2$ possible resolvents, bounded by $(|S| + n)^2 * (2^n c)^2$. We must resolve n times, so this must be raised to the n^{th} power, giving $(|S| + n)^{2n} * (2^n c)^{2n}$, or $(|S| + n)^{2n} * (4^{n^2})c^{2n}$. This has a quadratic in the exponent, but is still single exponential.

We also need to show that each resolution can be done in exponential time. The subterm size of a resolvent can at most double at each resolution, and so it is at worst exponential in n. Factoring does not increase the subterm size. Therefore each resolution can be done in exponential time, since resolution may be done in time polynomial in the subterm size.

For the lower bound, we can take the clause set $S_{fg} \cup \{P(a)\} \cup B_n$. The shortest proof from B_n has linear length, as we showed in theorem 2.4.11. Also, from $S_{fg} \cup \{P(a)\}$, one can derive an exponential number of proofs of linear length, as we showed in theorem 2.4.8.

□

We note that this result does not appear to hold if we allow many factoring operations to be included in a single resolution operation. However, even in that case, if we consider the sum of the sizes of the clauses in the proof instead of the length n of the proof, we can still obtain a complexity exponential in the proof length.

Lemma 2.4.19 *Suppose that S is a set of clauses with a depth n binary resolution refutation. Then there is a depth $2^n - 1$ P_1 deduction refutation from S (with factoring) and a length $|S|(2^n - 1)$ P_1 deduction refutation (with factoring). Also, on the ground level, clauses of size at most n are needed. If resolution and factoring are considered as separate operations, then there is a depth $n(2^n - 1) + (m - 1)$ P_1 refutation and a length $n|S|(2^n - 1) + M - |S|$ P_1 refutation, where m is the maximum number of literals in a clause of S and M is the sum of the number of literals in the clauses of S. All of these bounds are single exponential in n.*

Proof. We show this for propositional logic, and then lift to first-order logic. For propositional logic, the case $n = 0$ is immediate. Now, we show that from any set S' of propositional clauses having a depth k binary resolution proof, $k > 0$, there is a P_1-deduction proof of depth $2^k - 1$. For this, we consider the last predicate symbol P that gets resolved away to produce the empty clause. Then there must be a depth $k - 1$ binary resolution proof of P (with clauses of size at most k) and a depth $k-1$ binary resolution proof of $\neg P$ with clauses of size at most k. From the first proof, we can remove occurrences of P from S', and then we obtain a depth $k-1$ binary resolution refutation from this clause set with clauses of size at most $k - 1$. By induction, we get a P_1 proof of the empty clause of depth $2^{k-1} - 1$ from S' with P occurrences removed and with clauses of size at most $k-1$. Putting P back in, we get a P_1 proof of depth $2^{k-1} - 1$ of P from S' with clauses of size at most k. Resolving this P with S', we remove all occurrences of $\neg P$. This takes one more depth of proof. Then there is a depth $k-1$ proof of the empty clause from S' with all $\neg P$ removed. By induction, this can be made into a depth $2^{k-1} - 1$ P_1-deduction proof with clauses of size at most $k - 1$. The total depth is then $2^{k-1} - 1 + 1 + 2^{k-1} - 1$, or, $2^k - 1$. This completes the induction.

Lifting this result to first-order logic, we obtain that there is a depth $2^n - 1$ P_1 deduction proof from S. Also, since each clause in the lifted proof corresponds to a ground instance with at most n literals, it turns out that each resolvent corresponds to a ground instance with at most $n-1$ literals, since one literal was removed. Therefore we can perform at most $n - 1$ factoring operations after each resolution to insure that the non-ground clauses have the same number of literals as their ground instances. Thus for resolution and factoring as separate operations, the depth may be $n(2^n - 1)$. We may also have to factor the input clauses so that they have the same number of literals as their ground instances. This may require a maximum depth of $m - 1$ factorings, where m is the maximum number of literals of a clause in S. Thus the total depth may be $n(2^n - 1) + (m - 1)$.

To get the length $|S|(2^n - 1)$ result, we notice that the step of resolving P with S' will require at most $|S|$ resolutions in the lifted clause set. This step has to be done once for each resolution in the depth n proof. Since there will be at most 2^n-1 resolutions in a depth n proof, the result follows. If factoring and resolution are separate operations, then we may need $n-1$ factoring operations after each resolution, giving a length of at most $n|S|(2^n - 1)$. Considering factorings of the input clauses as before, this can be at most $n|S|(2^n - 1) + M - |S|$, where M is the sum of the number of literals in the clauses of S.

□

Theorem 2.4.20 *Depth-bounded hyper-resolution has triple exponential complexity with respect to the proof depth complexity measure M_{pd}.*

Proof. Consider the clause set $B_n \cup T_{fg}$. To refute B_n, hyper-resolution needs an exponential number (2^n) of steps. But hyper-resolution from the clause set T_{fg} can generate a triple exponential number of clauses with proofs of depth 2^n, by theorem 2.4.5. This gives the lower bound.

Now, suppose that S is a set of clauses having a depth n binary resolution proof. By lemma 2.4.19, S has an exponential depth P_1-deduction proof. Now, by general arguments, in exponential depth, the subterm size and number of literals of the clauses generated can be at most double exponential, and there can then be at most a triple exponential number of clauses generated, by corollary 2.4.14. This makes the total work at most triple exponential.

□

Theorem 2.4.21 *Length-bounded P_1-deduction has double exponential complexity with respect to the proof length complexity measure M_{pl}, if resolution and factoring are considered as separate operations.*

Proof. To get the upper bound, if there is a length n binary resolution proof, then the proof has depth at most n. Therefore by lemma 2.4.19, there is a P_1 deduction proof of length at most single exponential in n. Then, reasoning as for binary resolution with respect to the proof length measure, we obtain a double exponential upper bound for the complexity. That is, the subterm size measure and number of literals can at most double at each resolution, so they have a double exponential bound. Therefore we get a possibly double exponential number of choices at each step of a resolution to perform, and in a single exponential number of choices, this results altogether in at most a double exponential complexity.

For the lower bound, consider the clause set $B_n \cup S_{fg} \cup \{P(a)\}$. B_n is a binary counter, which P_1 deduction needs exponentially many resolutions to refute, but binary resolution requires only linear depth. Then the clauses $S_{fg} \cup \{P(a)\}$ give a double exponential number of single literal resolvents of exponential length using P_1 deduction, by theorem 2.4.8.

□

The following result is interesting because it has to do with the complexity of a length-bounded strategy with respect to a proof depth measure:

Theorem 2.4.22 *Length-bounded binary resolution has double exponential complexity with respect to the proof depth complexity measure M_{pd}, if resolution and factoring are separated.*

Proof. Note that if there is a depth n binary resolution proof, then there is an exponential length binary resolution proof, since each clause has at most two parents. The number of proofs of exponential length is at most double exponential. For the lower bound, consider the clause set $P_n \cup S_{fg}$, where P_n is the pigeonhole problem of size n. This is a propositional problem for which Haken [Hak85] showed that any resolution proof has exponential length. Since any propositional problem has a linear depth proof, P_n does too. However, it requires an exponential length proof, and there are double exponentially many clauses that may be generated from S_{fg} in exponential length.

□

2.4.2 Model Elimination

We now consider the complexity of model elimination [Lov69] with respect to the proof length and proof depth complexity measures.

Theorem 2.4.23 *With respect to the proof depth complexity measure M_{pd}, the complexity of model elimination is at least double exponential and at most triple exponential.*

Proof. For the lower bound, consider the clause set $B_n \cup S_{fg} \cup \{P(a)\}$. This has linear complexity with respect to the proof depth bound, due to B_n. However, the minimal length proof for model elimination is single exponential, analogous to the situation for P_1-deduction. The clause set $S_{fg} \cup \{P(a)\}$ in single exponential length proofs can generate a double exponential number of proofs, leading to the lower bound.

For the upper bound, if there is a depth n proof from S, then on the ground level, at most 2^n literals are involved in the proof (that is, in a Herbrand set for S). Now, we consider the tableau formulation of model elimination, for simplicity; in this formulation, we construct trees in which, at each node, there are outgoing edges labeled with $L_1, \ldots, L_m$ for some clause $\{L_1, \ldots, L_m\}$ in S (or an instance of a clause in S). The tree can be stopped when there are complementary edges on a path, and one never needs paths with identical literals appearing more than once. For 2^n propositions, the depth of the tree can therefore be at most single exponential, the number of nodes in the tree can be at most double exponential, and the number of choices altogether is at most triple exponential, since model elimination has at most a bounded number of choices at each step involving the addition of a new clause. There are also a number of choices proportional to the length of the chain, involving applying a substitution and deleting literals, but the overall result is still triple exponential.

□

Theorem 2.4.24 *With respect to the proof length complexity measure M_{pl}, the complexity of model elimination is at least double exponential and at most triple exponential.*

Proof. The upper bound follows because a proof of length n has depth no larger than n. It is surprising that we cannot derive a better result than this. The lower bound is from $B_n \cup S_{fg} \cup \{P(a)\}$. We know that there is a linear length refutation from B_n by theorem 2.4.11.

However, any model elimination refutation from B_n requires an exponential number of steps. In this many steps, the clause set $S_{fg} \cup \{P(a)\}$ permits a double exponential number of choices.

□

2.5 The Duplication Measure

We now consider the instance-based theorem proving complexity measures M_{dup}, M_{sub}, and M_{lin}, as opposed to the inference-based measures M_{pd} and M_{pl} considered above. The first such measure to be considered is the duplication complexity M_{dup}.

2.5.1 Clause Linking

We begin by analyzing the time complexity of methods based on clause linking with respect to the duplication complexity measure, first proving some results about the lengths of clause linking proofs. For this, we define $Gr(S)$ to be S with all variables replaced by a fixed constant symbol, as in [LP92], and similarly for literals and clauses.

Definition 2.5.1 *A* linking operation *between two clauses C and D is a unification between literals $L \in C$ and $M \in D$ such that L and M have opposite sign, generating instances $C\Theta$ and $D\Theta$ such that $L\Theta$ and $M\Theta$ are complementary. This is the basic operation in the clause linking strategy of [LP92], although that paper uses a slightly different notion of* hyper-linking *that links all the literals of a clause at once instead of only one. We say that the clauses $C\Theta$ and $D\Theta$ are* obtained from *C and D* by a linking operation.

Definition 2.5.2 *A* clause linking proof *from S is a sequence C_1, C_2, ..., C_n of clauses where each C_i is either in S or is obtained by a linking operation from two earlier clauses in the sequence. We assume that variables in the C_i are renamed so that no two clauses share any variables. We call n the* length *of the proof. A* clause linking refutation *is a clause linking proof in which the set $Gr(\{C_1, C_2, \ldots, C_n\})$ is propositionally unsatisfiable.*

Definition 2.5.3 *The* depth *of a clause C in a clause linking proof from S is 0, if $C \in S$, else it is $1 + max(d_1, d_2)$, where d_1 and d_2 are the depths of the clauses from which C was obtained by a linking operation. The* depth *of a clause linking proof is the maximum depth of any clause in the proof.*

We comment on how depth-bounded and size-bounded clause linking is implemented, for purposes of this discussion. For depth-bounded clause linking from a set S of input clauses, for each depth d from 1 until a proof is found, we do the following:

1. Generate all clauses of depth d or less that may be obtained by linking previously generated clauses of depth d or less. Call this set of clauses S_d.
2. Test $Gr(S_d)$ for satisfiability using some efficient propositional decision procedure such as Davis and Putnam's method.

Note that there is no nondeterminism in clause linking, implemented in this way, since the sets S_d are generated exhaustively in order of increasing d. That is, S_d is a function of S_{d-1} and S. The time required, then, to find a proof, is the time to generate the sets S_d for d up to the required depth, plus the time to test each set $Gr(S_d)$ for satisfiability. Subterm-size bounded and linear-size bounded clause linking are analogous, except that instead of the depth d, we use the subterm size or linear size of the clauses generated as the bound.

Recall that a theorem proving method is *complete* if it permits a proof of unsatisfiability for every unsatisfiable clause set. We recall from [LP92] that clause linking is complete, that is, if S is unsatisfiable, then there is a clause linking refutation from S.

Definition 2.5.4 *A* rigid clause linking proof *from S is a sequence S_1, S_2, ..., S_n of sets of clauses where S_1 is an amplification of S, each S_{i+1} may be expressed as $S_i\Theta_i$, and Θ_i is a unifier of two complementary literals in different clauses of S_i. We call n the* length *of the proof. A* rigid clause linking refutation *is a clause linking refutation in which the set $Gr(S_n)$ is propositionally unsatisfiable.*

Definition 2.5.5 *A clause $C\Theta$ is a* variant *of C if Θ is a 1-1 mapping of variables to variables. Thus $C\Theta$ is C with the variables renamed.*

Definition 2.5.6 *If $\{C_1, \ldots, C_n\}$ is a set of clauses, then a* separation *of $\{C_1, \ldots, C_n\}$ is a set $\{C_1', \ldots, C_n'\}$ of clauses where each C_i' is a variant of C_i and C_i' and C_j' do not share variables for distinct i and j. We use $Sep(S)$ to refer to a separation of S chosen in some way.*

Definition 2.5.7 *An* amplified clause linking proof *from S is a sequence S_1, S_2, ..., S_n of sets of clauses where S_1 is an amplification of S, each S_{i+1} may be expressed as $Sep(S_i\Theta_i)$, and Θ_i is a unifier of two complementary literals in different clauses of S_i. We call n the* length *of the proof. An* amplified clause linking refutation *is a clause linking refutation in which the set $Gr(S_n)$ is propositionally unsatisfiable.*

We develop some technical results to relate the length of clause linking proofs, amplified clause linking proofs, and rigid clause linking proofs. The problem is that variables are held rigid in the latter but not in the former two.

Proposition 2.5.8 *If there is an amplified clause linking refutation from S of length n, then there is a clause linking refutationfrom S of length* $4(n-1)$ *and depth $n-1$.*

Proof. Suppose S_1, S_2, ..., S_n is an amplified clause linking refutation from S. One obtains the clause linking refutation by listing for each S_i, $i > 1$, the two clauses that were modified by the substitution Θ_{i-1} and possibly their two parents, if they were not listed yet. Therefore each amplified step S_i for $i > 1$ may generate four clauses in the clause linking proof. Thus the $4(n-1)$ bound. Each amplified step S_i only increases the depth by one, whence the $n-1$ bound.

□

The following result gives a basis for bounding the length of amplified clause linking proofs.

Lemma 2.5.9 *Suppose that L and M are unifiable literals and that $Gr(L)$ and $Gr(M)$ are different. Let Θ be a most general unifier of L and M. Then Θ binds at least one variable of L or M to a non-variable term, and hence the number of variables in $\{L, M\}\Theta$ is less than the number of variables in $\{L, M\}$. Also, the number of non-variable symbol occurrences in $\{L, M\}\Theta$ is larger than the number in $\{L, M\}$.*

Proof. Consider the first subterms r and s of L and M, respectively, such that $Gr(r)$ and $Gr(s)$ differ. Since L and M are unifiable, both of these terms cannot start with a function symbol. Since $Gr(r)$ and $Gr(s)$ differ, both of these terms cannot be variables. Therefore one of these terms is a variable x and the other is a non-variable term t. On unification, x will be bound to t, so the number of variables will decrease and the number of non-variable symbol occurrences will increase.

□

Definition 2.5.10 *If T is a set of ground clauses and L is a literal such that for some clause C in T, $L \in C$, we say that L is* pure *in T if there is no other clause D and literal $M \in D$ such that M is complementary to L.*

We note that if T is minimal unsatisfiable, then T contains no pure literals.

Theorem 2.5.11 *Suppose S is unsatisfiable and T is a Herbrand set for S. Then there is an amplified clause linking refutation from S of length not greater than $s^{lin}(T)$.*

Proof. There is an amplification S_1 of S such that for some α_1, $T = S_1\alpha_1$. We can assume that T is minimal unsatisfiable, implying that T has no pure literals.

We know from lemma 2.2.18 that if $T = S_i\alpha_i$ and $Gr(S_i)$ has all literals identical/complementary when the corresponding literals in T are identical/complementary, then $Gr(S_i)$ will be unsatisfiable. Otherwise, there must be two literals L_1 and L_2 of S_i such that $Gr(L_1)$ and $Gr(L_2)$ are not identical/complementary but the corresponding literals

$L_1\alpha_i$ and $L_2\alpha_i$ of T are identical/complementary. If L_1 and L_2 are of the same sign, then there must be some literal M in S_i such that $L_i\alpha_i$ and $M\alpha_i$ are of opposite sign in T (since T has no pure literals). Then since $Gr(L_1)$ and $Gr(L_2)$ are not identical, it must be that some $Gr(L_i)$ is not identical to $Gr(M)$, too. Thus there must be two literals of $Gr(S_i)$ that are not complementary when the corresponding literals of T are. Thus there is a clause linking substitution Θ_i unifying these two complementary literals, and the number of non-variable symbol occurrences in $S_i\Theta_i$ will be larger than the number in S_i, by lemma 2.5.9. Since clause linking substitutions are most general unifiers, there will still be a substitution α_{i+1} such that $Sep(S_i\Theta_i)\alpha_{i+1} = T$. We can let S_{i+1} be $Sep(S_i\Theta_i)$. Thus this process can be continued. The number of such steps is then bounded by $s^{lin}(T)$, since this is a bound on the number of non-variable symbol occurrences in T. (We have, by the way, just proved the completeness of amplified clause linking.)

□

Corollary 2.5.12 *If S is unsatisfiable and T is a Herbrand set for S, then there is a clause linking refutation from S having length at most $4s^{lin}(T)$.*

Proof. This follows from theorem 2.5.11 and proposition 2.5.8.

□

We have now bounded the lengths of amplified clause linking proofs and clause linking proofs. We now bound the lengths of rigid clause linking proofs, and then give some examples to show that the former can sometimes be much longer than the latter.

Definition 2.5.13 *If S is a set of clauses, then $s^{sub}_{max}(S)$ is $max_{C\in S}s^{sub}(C)$ and $s^{lin}_{max}(S)$ is $max_{C\in S}s^{lin}(C)$.*

We note that $M'_{dup}(S) \le M_{dup}(S) * s^{sub}_{max}(S)$. Some of the following results could more accurately be stated in terms of $M'_{dup}(S)$.

Theorem 2.5.14 *Suppose that S is unsatisfiable. Then there is a rigid clause linking refutation of S of length at most $1 + M_{dup}(S)s^{sub}_{max}(S)$.*

Proof. Similar to the proof of theorem 2.5.11. We know that there is a Herbrand set T for S such that $|T| = M_{dup}(S)$. Thus there is an amplification S_1 of S such that for some α_1, $T = S_1\alpha_1$. Thus $|S_1| = |T| = M_{dup}(S)$. We can assume that T is minimal unsatisfiable, implying that T has no pure literals. We show that there is a rigid clause linking refutation from S_1 of length at most $1 + M_{dup}(S)s^{sub}_{max}(S)$.

We know from lemma 2.2.18 that if $T = S_i\alpha_i$ and $Gr(S_i)$ has all literals identical/complementary when the corresponding literals in T are identical/complementary, then $Gr(S_i)$ will be unsatisfiable. Otherwise, there must be two literals L_1 and L_2 of S_i such that $Gr(L_1)$ and $Gr(L_2)$ are not identical/complementary but the corresponding literals $L_1\alpha_i$ and $L_2\alpha_i$ of T are identical/complementary. If L_1 and L_2 are of the same sign, then there must be some literal M in S_i such that $L_i\alpha_i$ and $M\alpha_i$ are of opposite sign in T (since T has no pure literals). Then since $Gr(L_1)$ and $Gr(L_2)$ are not identical, it must be that some $Gr(L_i)$ is not identical to $Gr(M)$, too. Thus there must be two literals of $Gr(S_i)$ that are not complementary when the corresponding literals of T are. This means that one can obtain S_{i+1} by applying a clause linking substitution to S_i, and this will reduce the number of variables in S_i, by lemma 2.5.9. Since clause linking substitutions are most general unifiers, there will still be a substitution α_{i+1} such that $S_{i+1}\alpha_{i+1} = T$. Thus this process can be continued. The number of such steps is then bounded by $|S_1|s^{sub}_{max}(S)$, since this is a bound on the number of variables in S_1. (We have, by the way, just proved the completeness of rigid clause linking.) But $|S_1|s^{sub}_{max}(S) = M_{dup}(S)s^{sub}_{max}(S)$, and one more step may be needed in the proof to list S_1 itself.

□

This result implies that the bound for amplified clause linking can be exponentially larger than the bound for rigid clause linking. The question arises whether this can actually happen or whether the bound can be improved. We now give some surprising clause sets showing that this bound cannot be improved, and that clause linking and amplified clause linking can take much longer to find proofs than resolution and rigid clause linking. First we give a simple example to illustrate the idea, and then a more complicated example.

Consider the following clause set S_{cyc}:

$$\begin{array}{ll}
\{P(x_1,x_2,\ldots,x_n), & Q(x_2,x_3,\ldots,x_n,x_1)\} \\
\{\neg P(x_1,x_2,\ldots,x_n), & Q(g(a),x_2,\ldots,x_n)\} \\
\{\neg\, R(x_1,x_2,\ldots,x_n), & \neg\, Q(x_2,x_3,\ldots,x_n,x_1)\} \\
\{R(x_1,x_2,\ldots,x_n), & \neg\, Q(g(a),x_2,\ldots,x_n)\}
\end{array}$$

We develop some lemmas and results to show that this clause set requires exponential length clause linking refutations. In ths discussion, the term *input clause* refers to an element of S_{cyc}.

Definition 2.5.15 *A* g-clause *is an instance of a clause in S_{cyc} in which either*

- *All variables are bound to $g(a)$, or*
- *the variables $x_2,\ldots,x_i$ have been bound to the term $g(a)$, for some i, and the other variables are bound to distinct variables.*

A ground g-literal *is a ground literal in which either*

- *the predicate symbol is Q, and for some i, the first i arguments to Q are bound to $g(a)$, and the remainder to constant symbols, or*
- *the predicate symbol is P or R, and for some i, the second through i^{th} arguments of this predicate symbol are bound to $g(a)$, and the others are bound to constants, or*
- *the predicate symbol is P or R, and all the arguments of this predicate symbol are bound to $g(a)$*

A ground g-clause *is a ground clause all of whose literals are ground g-literals.*

We observe that all clauses in S_{cyc} are g-clauses. Also, a hyper-link operation between two g-clauses produces another g-clause.

Definition 2.5.16 *The* degree *of a g-clause is n, if x_1 is bound to $g(a)$, and otherwise $i-1$, where i is the highest integer not equal to one such that x_i is bound to $g(a)$. The degrees of the clauses in S_{cyc} are zero.*

Definition 2.5.17 *If L is a ground g-literal of the form $P(s_1, \ldots, s_n)$, $Q(s_1, \ldots, s_n)$, or $R(s_1, \ldots, s_n)$, or their negations, then the* degree *of L is the maximum i such that s_i is the term $g(a)$.*

Recall that $Gr(C)$ is the clause C with all variables replaced by a fixed constant symbol.

Lemma 2.5.18 *If C is a g-clause, then $Gr(C)$ is a ground g-clause.*

Lemma 2.5.19 *We can relate the parity of clauses and their ground instances in the following way:*

1. *If $\{ P(r_1, r_2, \ldots, r_n), Q(r_2, r_3, \ldots, r_n, r_1)\}$ is a g-clause of positive degree less than n, then the degrees of the literals $Gr(P(r_1, r_2, \ldots, r_n))$ and $Gr(Q(r_2, r_3, \ldots, r_n, r_1))$ have opposite parity, that is, one is odd and the other is even.*

2. *If $\{\neg P(r_1, r_2, \ldots, r_n), Q(g(a), r_2, \ldots, r_n)\}$ is a g-clause of positive degree less than n, then the degrees of the literals $Gr(\neg P(r_1, r_2, \ldots, r_n))$ and $Gr(Q(g(a), r_2, \ldots, r_n))$ have the same parity.*

3. *If $\{\neg R(r_1, r_2, \ldots, r_n), \neg Q(r_2, r_3, \ldots, r_n, r_1)\}$ is a ground g-clause of positive degree less than n, then the degrees of the literals $Gr(\neg R(r_1, r_2, \ldots, r_n))$ and $Gr(\neg Q(r_2, r_3, \ldots, r_n, r_1))$ have opposite parity.*

4. *If $\{R(r_1, r_2, \ldots, r_n), \neg Q(g(a), r_2, \ldots, r_n)\}$ is a ground g-clause of positive degree less than n, then the degrees of the literals $Gr(R(r_1, r_2, \ldots, r_n))$ and $Gr(\neg Q(g(a), r_2, \ldots, r_n))$ have the same parity.*

Theorem 2.5.20 *Any clause linking refutation of this clause set has length at least $2n+4$, but there is a resolution proof (with factoring) involving 3 resolutions, and a proof without factoring involving 3 resolutions and two factoring steps. We note that this clause set has duplication complexity 4.*

Proof. We show then that if T is a set of instances of S and every element of T is a ground g-clause of degree less than n, then T is satisfiable. This can be shown by considering the model M such that $M \models L$ iff L has positive even degree. The model M' such that $M' \models L$ iff L has odd or zero degree will also work. We then show that it takes over $2n$ links to create elements of large degree. First, each link operation can increase the degree by at most one. This gives a lower bound of n link operations. To get a $2n - 1$ lower bound, we observe that only every other link operation can increase the degree, due to the structure of S_{cyc}. We can increase the bound to $2n$ by noting that just one clause of degree n is not enough to get a contradiction, since we have two models M and M', one of which will satisfy any given clause. Thus we need at least $2n$ link operations. With the four input clauses, we obtain a lower bound of $2n + 4$ on the length of a clause linking refutation. In fact, with two more link operations, we can get a refutation, so the optimal bound is somewhere between $2n + 4$ and $2n + 6$. The resolution proofs mentioned in the theorem are easy to find.

□

We illustrate by giving a (not necessarily optimal) clause linking refutation for the case $n = 2$:

1.	$P(x_1, x_2), Q(x_2, x_1)$	(input)	(degree 0)
2.	$\neg P(x_1, x_2), Q(g(a), x_2)$	(input)	(degree 0)
3.	$\neg R(x_1, x_2), \neg Q(x_2, x_1)$	(input)	(degree 0)
4.	$R(x_1, x_2), \neg Q(g(a), x_2)$	(input)	(degree 0)
5.	$\neg R(x_1, g(a)), \neg Q(g(a), x_1)$	(2,3)	(degree 1)
6.	$R(x_1, g(a)), \neg Q(g(a), g(a))$	(5,4)	(degree 1)
7.	$P(g(a), g(a)), Q(g(a), g(a))$	(6,1)	(degree 2)
8.	$\neg P(g(a), g(a)), Q(g(a), g(a))$	(7,2)	(degree 2)
9.	$\neg R(g(a), g(a)), \neg Q(g(a), g(a))$	(8,3)	(degree 2)
10.	$R(g(a), g(a)), \neg Q(g(a), g(a))$	(9,4)	(degree 2)

The final set {7,8,9,10} is ground and unsatisfiable. We note that the clauses S_{cyc} essentially express a cyclic permutation between the two literals. To get more extreme examples, we need to encode clauses in which a permutation of exponentially many variables is expressed. The following set of four two-literal clauses accomplishes this:

Definition 2.5.21 S^n_{PQ} is the following set of four two-literal clauses:

$$\{P(f(f(...f(f(x_1,y_1),y_2),...y_{n-1}),y_n)),$$
$$Q(f(y_n,f(y_{n-1},f(y_{n-2},...,f(y_2,f(y_1,x_1))..)))))\}$$
$$\{\neg P(f(f(...f(f(x_1,y_1),y_2),...y_{n-1}),y_n)),$$
$$Q(f(f(...f(f(g(a),y_1),y_2),...y_{n-1}),y_n))\}$$
$$\{\neg R(f(f(...f(f(x_1,y_1),y_2),...y_{n-1}),y_n)),$$
$$\neg Q(f(y_n,f(y_{n-1},f(y_{n-2},...,f(y_2,f(y_1,x_1))..)))))\}$$
$$\{R(f(f(...f(f(x_1,y_1),y_2),...y_{n-1}),y_n)),$$
$$\neg Q(f(f(...f(f(g(a),y_1),y_2),...y_{n-1}),y_n))\}$$

Theorem 2.5.22 *Any clause linking refutation of S^n_{PQ} must have length at least $2 * 2^n + 4$. There is a resolution proof (with factoring) involving 3 resolutions, and a proof without factoring involving 3 resolutions and 2 factoring steps. We note that S^n_{PQ} has duplication complexity 4, and that $n < s^{lin}(S)$.*

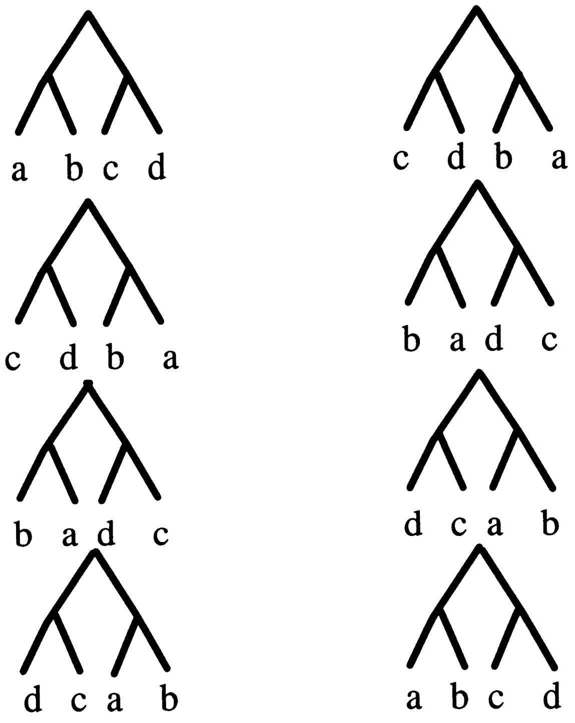
Proof. The idea is the same as the preceding theorem. Each clause expresses a permutation on a binary tree having 2^n leaves. To see this, consider $f(x,y)$ as a binary tree with x as the left subtree and y as the right subtree. Thus $f(f(...f(f(x_1,y_1),y_2),\dots y_{n-1}),y_n)$ is a binary tree in which y_n is the right subtree, $f(...f(f(x_1,y_1),y_2),...y_{n-1})$ is the left subtree, and x_1 is a leaf. Then the term $f(y_n,f(y_{n-1},f(y_{n-2},\dots, f(y_2,f(y_1,x_1))..)))$ is a binary subtree with the leaves permuted; the positions of the left and right subtrees down the leftmost path have all been reversed. This permutes the positions of the 2^n leaves, and one can show without much trouble that repeating this permutation 2^n times will bring every element into every position.

To obtain a clause linking refutation, each of these 2^n leaves must be bound to $g(a)$, as in the preceding theorem. The bound is obtained in a way similar to that for the set S_{cyc}, but here the degrees can be as high as 2^n, instead of n. One sees also that the depth of this refutation must be 2^n, since each linking operation can increase the degree by at most one, and we need clauses of degree 2^n for a refutation. The resolution proofs, as before, are easy to construct.

□

Permutations Induced by a Pair of Terms

$f(f(x_1,y_1),y_2)$ and $f(y_2,f(y_1,x_1))$



We note that the preceding clause set has a very short (length 7) rigid clause linking refutation. Therefore clause linking is much worse than rigid clause linking on this example. It is interesting that for some sets of Horn clauses, one obtains a much better result for clause linking, which may help to explain the good performance of clause linking in practice.

Theorem 2.5.23 *Suppose S is a Horn set having duplication complexity n. Suppose that there is a hyper-resolution refutation from S in which each derived unit is used only once as a lemma. Then there is a clause linking refutation from S of length $3n - 2$.*

Proof. Let Θ be a substitution such that $S'\Theta$ is a Herbrand set for S, where S' is an amplification of S. Consider the set (L, M) of pairs of literals of S' such that $L\Theta$ and $M\Theta$ are complementary. We order the clauses of S' so that C_i is before C_j if there is such a link (L, M) with $L \in C_i$ and $M \in C_j$ and L is positive and M is negative. Since S is a Horn set, such an ordering is possible. Then we order the links (L, M) by the ordering of C_i, where L is positive and $L \in C_i$. We obtain the clause linking proof by performing clause linking operations corresponding to these links once in order and then once in reverse order. This will give us $2n$ operations, and with n input clauses, the total proof length is $3n$. With a more careful analysis, one can show that $3n - 2$ suffices.

We give an example to make the construction clear. Suppose $S = S'$ $= \{\{P(a,x)\}, \{\neg P(u,v), Q(u,v)\}, \{\neg Q(w,b)\}\}$. Then we order these clauses left to right and have the following clause linking refutation:

1. $P(a,x)$	(input)	
2. $\neg P(u,v), Q(u,v)$	(input)	
3. $\neg Q(w,b)$	(input)	
4. $\neg P(a,v), Q(a,v)$	(1,2)	(beginning the forward phase)
5. $Q(a,b)$	(4,3)	
6. $\neg P(a,b), Q(a,b)$	(4,5)	(beginning the backwards phase)
7. $P(a,b)$	(1,6)	

We note that the last three clauses are ground and unsatisfiable and that the length of this proof is 7 which is $3 * 3 - 2$.

□

However, even for Horn clauses, clause linking (and resolution) can generate terms of subterm size exponential in the duplication complexity M_{dup}, as the following example shows:

$\{P_2(f(x,y)), \quad \neg P_1(x), \neg P_1(y)\}.$
$\{P_3(f(x,y)), \quad \neg P_2(x), \neg P_2(y)\}.$
$\ldots$
$\{P_n(f(x,y)), \quad \neg P_{n-1}(x), \neg P_{n-1}(y)\}.$
$\{P_1(g(z))\}$
$\{\neg P_n(x)\}$

This set of clauses has duplication complexity $n + 1$. By the above theorem, there is a clause linking proof of length proportional to n. However, this proof generates clauses whose subterm size is exponential in n, and this is unavoidable. We illustrate a few steps to show how this occurs:

1. $P_1(g(z))$ (input)
2. $P_2(f(x,y)), \neg P_1(x), \neg P_1(y)$ (input)
3. $P_2(f(g(z_1), g(z_2))), \neg P_1(g(z_1)), \neg P_1(g(z_2))$ (1,2, two steps)
4. $P_3(f(x,y)), \neg P_2(x), \neg P_2(y)$ (input)
5. $P_3(f(f(g(z_1), g(z_2)), f(g(z_3), g(z_4)))),$
   $\neg P_2(f(g(z_1), g(z_2))), \neg P_2(f(g(z_3), g(z_4)))$ (3,4,two steps)

$\ldots$

This shows that sometimes clause linking generates instances having much larger subterm size than necessary (by an exponential) and impairs the efficiency of clause linking. Rigid clause linking overcomes this problem, since by the proof of theorem 2.5.14, the subterm size needed is bounded by $M'_{dup}(S)$. Thus there is a rigid clause linking refutation for this set of clauses in which the subterm size remains small. Another way to overcome the problem is to allow a *subterm factoring* operation as follows:

Definition 2.5.24 *The* subterm factoring *operation generates a clause $C\Theta$ from C, where Θ is a most general unifier of two terms in C.*

With subterm factoring, clause linking can always find proofs by generating instances having a subterm size that is bounded by $2 * M'_{dup}(S)$. This may be a useful operation to add for this reason. We have the following general lifting theorem for resolution with subterm factoring, and similar theorems may be proven for clause linking and other inference systems:

Theorem 2.5.25 *Suppose S is a set of clauses and T is a Herbrand set for S. Suppose $D_1, D_2, \ldots, D_n$ is a resolution refutation from T. Then there is a resolution refutation $C_1, C_2, \ldots, C_p$ from S with subterm factoring such that for all i, $s^{sub}(C_i) \leq 2 * s^{sub}(T)$.*

Proof. We lift the proof from T as usual and insert subterm factoring operations whenever two subterms of D_i are identical but the corresponding subterms of C_j are not. After performing these factoring operations, we obtain that $s^{sub}(C_j)$ is bounded by $s^{sub}(D_i)$ for the corresponding clause D_i in the ground proof. If we resolve two clauses C_j and C_k, we produce a clause C such that $s^{sub}(C) \leq s^{sub}(C_j) + s^{sub}(C_k)$. Thus we only know that $s^{sub}(C) \leq 2 * s^{sub}(T)$.

$\square$

We note that resolution without factoring also has a problem with the above clause set, generating terms having exponential subterm size for it. This may indicate a deficiency in the search efficiency of resolution without factoring, even though it is complete for Horn clauses. By factoring the clauses

$$P_i(f(x,y)), \neg P_{i-1}(x), \neg P_{i-1}(y)$$

we can generate the clauses

$$P_i(f(x,x)), \neg P_{i-1}(x)$$

and then obtain a refutation with small subterm size. We still don't know whether factoring can always permit refutations with small subterm size to be found, and leave this as an open problem. However, a change in the clause set prevents this for many common resolution strategies. Consider the following clause set S_{PQR}:

$\{Q_1(x), \quad \neg P_1(x)\}$
$\{R_1(x), \quad \neg P_1(x)\}$
$\{P_2(f(x,y)), \quad \neg Q_1(x), \neg R_1(y)\}$
$\{Q_2(x), \quad \neg P_2(x)\}$
$\{R_2(x), \quad \neg P_2(x)\}$
$\{P_3(f(x,y)), \quad \neg Q_2(x), \neg R_2(y)\}$
...
$\{Q_{n-1}(x), \quad \neg P_{n-2}(x)\}$
$\{R_{n-1}(x), \quad \neg P_{n-2}(x)\}$
$\{P_n(f(x,y)), \quad \neg Q_{n-1}(x), \neg R_{n-1}(y)\}$
$\{P_1(g(z))\}$
$\{\neg P_n(x)\}$

For this clause set, the following strategies require subterm size that is exponential in the duplication complexity M_{dup} for refutations, even with factoring: P_1 deduction, hyper-resolution, A-resolution, and UR resolution. We can get the same result for negative resolution by using S_{PQR} with the signs of all literals changed. However, with the subterm factoring operation added, these strategies can find proofs by generating clauses having subterm size bounded by $2 * M'_{dup}(S)$, by theorem 2.5.25.

Having completed these preliminaries, we now begin analyzing the complexity of various strategies with respect to the duplication complexity measure M_{dup}. We first consider clause linking and a related strategy.

Theorem 2.5.26 *Depth-bounded clause linking has a quadruple exponential complexity with respect to the duplication complexity measure M_{dup}. This can be reduced to triple exponential by a modification of the satisfiability procedure. Length-bounded clause linking has double exponential complexity with respect to the duplication measure M_{dup}.*

Proof. Recall that depth-bounded clause linking means that one performs linking operations in order of their depth. We essentially showed in corollary 2.5.12 that the length (hence the depth) of a clause linking refutation from S is at worst exponential in the duplication complexity of S. This can generate clauses whose subterm size is double exponential in the duplication complexity of S (by theorem 2.2.10). The number of clauses having this subterm size can be triple exponential in the duplication complexity of S, by corollary 2.4.14. Testing unsatisfiability on this many clauses can take quadruple exponential time. This gives the upper bound.

For the lower bound, consider the clause set $S^n_{PQ} \cup T_{fg}$. The clauses S^n_{PQ} require a number of linking operations exponential in $s^{lin}(S)$ to generate the ground instances needed for the proof. In this many operations, the clauses T_{fg} can generate a triple exponential number of instances.

We can reduce this bound to triple exponential by testing all subsets of size equal to the length of the clause linking proofs constructed. That is, when we construct proofs of depth p, we test only subsets of size linear in p for satisfiability. There will be at most triple exponentially many such subsets, and even with exponential time for testing satisfiability, the worst case time overall is still triple exponential in the duplication complexity of S.

For length-bounded clause linking, we observe that the number of proofs of length p is single exponential in p, since the number of literals in the clauses never increases. For each proof $C_1, C_2, \ldots, C_p$, we can test the set $\{C_1, C_2, \ldots, C_p\}$ for satisfiability in time exponential in p in the worst case. This leads to a double exponential bound overall, since we may need exponential length clause linking refutations.

$\square$

It is remarkable to get so much variation in running time by such seemingly small changes in strategy. Even for the quadruple exponential version, it is possible that the fast running time of Davis and Putnam's method [Dav63, DP60, DLL62, Fit90] on the average can make the bound triple exponential in practice.

We now consider search strategies based on the size of the instances rather than on the length or depth of the proofs. Again we get some dramatic differences in the complexity.

Theorem 2.5.27 *Linear-size bounded clause linking is triple exponential with respect to the duplication complexity measure M_{dup}. This may be reduced to double exponential by testing subsets of cardinality equal to the size bound.*

Proof. Recall the definition of linear-size bounded search (definition 2.4.15). The upper bound is obtained as before, by noting that the linear size measure can be exponential with respect to the duplication complexity, and there can be a double exponential number of clauses within this size bound, requiring triple exponential time to test satisfiability. For the lower bound, consider the clause set T_{fg} together with the following clause set, where s and t are terms whose most general instance is exponentially large:

$$\{\{P(s), Q(s)\}, \{P(s), \neg Q(t)\}, \{\neg P(t), Q(s)\}, \{\neg P(t), \neg Q(t)\}\}$$

This will require the size bound to become exponentially large before a proof is found, and within this size bound, the clause set T_{fg} will generate a double exponential number of instances.

The worst case bound can be reduced to double exponential as before by testing small subsets for satisfiability. For this, when the size bound is n, we consider subsets of the instances containing n clauses. When n becomes exponential in the duplication complexity, this guarantees that a proof will be found.

□

Theorem 2.5.28 *Subterm-size bounded clause linking is triple exponential with respect to the duplication complexity measure M_{dup}. This may be reduced to double exponential by testing subsets of cardinality equal to the size bound, for satisfiability. If the subterm factoring operation is allowed, the bounds become double and single exponential, respectively.*

Proof. If S is the set of input clauses and n is its duplication complexity, then we may need to generate instances of size exponential in n to obtain the clause linking refutation. There can be a double exponential number of clauses having subterm size less than or equal to this size bound, by corollary 2.4.14. When the size bound reaches this value, then a proof will be found. The time to apply a propositional satisfiability test to them can then be triple exponential.

To reduce the time to double exponential, we can test subsets R of the generated instances such that R has cardinality m when the size bound is m. This will guarantee that a proof is found when m becomes exponential in n. There are a double exponential number of subsets of this size, and the propositional satisfiability test on each one takes double exponential time, leading to a double exponential bound overall.

The lower bound is obtained by letting S be $T_{fg} \cup S^n_{PQ}$ When the subterm size becomes exponential in n, a proof can be found, but then T_{fg} will have generated a double exponential number of instances.

For the subterm factoring operation, one obtains bounds that are better by an exponential, because one can insure that the subterm size needed for a refutation is linear in the duplication complexity of S.

□

These results show the strong (asymptotic) influence the use of a size bound can have on a strategy, and how the choice of size bound can make a large difference. We note as before that the good performance of Davis and Putnam's method may reduce the triple exponential bound to double exponential in practice.

We now define CLIN-D, a version of CLIN (clause linking) with rigid variables, and analyze its complexity properties.

Definition 2.5.29 *A CLIN-D proof from S is a sequence $S_1, S_2, \ldots, S_n$ of sets of clauses such that S_1 is empty and for all i, either S_{i+1} is $S_i \cup \{C\alpha_i\}$ for some $C \in S$ where $C\alpha_i$ is a variant of C, or S_{i+1} is $S_i\Theta_i$ and Θ_i is a linking substitution for S_i, that is, Θ_i is a most general unifier of two literals of S_i of opposite sign. The* length *of this proof is n. Such a proof is a* refutation *if $Gr(S_n)$ is unsatisfiable.*

Theorem 2.5.30 *Suppose S is unsatisfiable. Then there is a CLIN-D refutation of length at most $M_{dup}(S)(1 + s^{sub}_{max}(S))$.*

Proof. By theorem 2.5.14, there is a rigid proof of length $1 + M_{dup}(S)s^{sub}_{max}(S)$ from S starting with an amplification having $M_{dup}(S)$ clauses in it. To obtain a CLIN-D proof, it is only necessary to delete the first step of the rigid proof, and add at most $M_{dup}(S)$ steps to the beginning of the proof to introduce all the clauses in the amplification.

□

Theorem 2.5.31 *Length-bounded CLIN-D has a single exponential complexity with respect to the duplication measure M_{dup}.*

Proof. We are assuming that CLIN-D proofs are generated in order of their length. To get the upper bound, we note that the number of choices in constructing the proof is exponential in the length of the proof, and we just showed that the proof length is linear in the duplication complexity. Thus there are exponentially many proofs to consider, and the propositional satisfiability test for each one is of exponential complexity, leading to an exponential bound overall. To get the lower bound, we can take any propositional clause set for which Davis and Putnam's method takes exponential time, such as the pigeonhole problems.

□

CLIN-D is essentially different from the matings method of Andrews [And81], which essentially performs the propositional satisfiability test at the start in the sense of finding a spanning set. CLIN-D performs this test at the end, which is possibly more efficient, since efficient propositional satisfiability tests may be used. However, CLIN-D may do unnecessary work, since the generation of instances is not guided by the propositional satisfiability test. CLIN-D is also essentially different from CLIN, which does not have rigid variables. It is not clear how well CLIN-D would perform in practice, since it involves many nondeterministic choices.

We can modify CLIN-D to make it more flexible by also allowing the Θ_i to be replacements of a variable by terms of the form $f(x_1, \ldots, x_n)$ where the x_i are distinct new variables. We know that each literal in S_i has to unify with some other literal and each variable has to be replaced by some function or constant symbol. Depending on the number of function symbols and literals, we can always try to perform whichever operation has the smaller number of choices. This may make this method more efficient.

2.5.2 Resolution Based Methods

We now analyze resolution-based methods with respect to the duplication measure.

Theorem 2.5.32 *Depth-bounded binary resolution is of double exponential complexity with respect to the duplication complexity measure M_{dup}.*

Proof. Suppose S is the set of input clauses. Then there is a Herbrand set T for S of subterm complexity linear in the duplication complexity of S. It follows that the number of (distinct) literals in T is also linear in the duplication complexity, so there is a semantic tree of depth linear in this quantity. Therefore, there is a resolution proof of depth linear in the duplication complexity. If factoring steps are separated, the depth can become quadratic. Within a linear (or quadratic) depth, one can generate clauses of at most exponential subterm size, and there can be at most a double exponential number of them. The bound holds whether resolution and factoring are combined or separate.

For the lower bound, consider the clause set $P_n \cup S_{fg}$ where P_n is the pigeonhole problem of size n. The clause sets P_n require linear depth proofs, and have a linear duplication complexity (since they are propositional). Within a linear proof depth, the clauses S_{fg} can generate a double exponential number of resolvents.

□

Note that such a simple-minded resolution strategy is actually better than depth-bounded clause linking in the worst case, with respect to the duplication measure.

Theorem 2.5.33 *Length-bounded binary resolution is of double exponential complexity with respect to the duplication complexity measure M_{dup}, if resolution and factoring are separated.*

Proof. The proof is much the same as the above. For the lower bound, one notes that any clause derivable in linear depth is derivable in exponential length. The upper bound is based on theorem 2.4.18. The reason we cannot immediately derive this result from the previous one is that we are assuming that the length-bounded proofs are enumerated, one by one, and so the same clauses may be generated repeatedly. Possibly a better search method could reduce the work. For depth-bounded resolution, in contrast, each clause only needs to be generated once.

□

Theorem 2.5.34 *Linear-size bounded binary resolution has a double exponential complexity with respect to the duplication complexity measure M_{dup}.*

Proof. We are assuming that we only save resolvents whose linear term size (that is, the size of the clause) is within the size bound, and that this size bound is gradually increased until a proof is found. Suppose S is a set of clauses. Consider the Herbrand set T for S such that $s^{lin}(T)$ is minimal. Then $s^{lin}(T)$ can be at worst exponential in the duplication complexity of S. When the size bound reaches this value, we will obtain a proof (because the sum of the sizes of the literals in T is exponential in the duplication complexity of S). There can be at most a double exponential number of clauses generated within this size bound.

For the lower bound, let S be S_{fg} together with the clause set

$$\{\{P(s), Q(s)\}, \{P(s), \neg Q(t)\}, \{\neg P(t), Q(s)\}, \{\neg P(t), \neg Q(t)\}\}$$

where s and t are terms whose most general instance is exponentially large. In order to get the proof, we need to reach an exponential size, and within this size bound, S_{fg} can generate a double exponential number of clauses.

□

Definition 2.5.35 *The clauses set T^n_{fg} consists of the clauses $\{\neg T_i(x,y)$, $\neg T_i(y,z)$, $T_{i+1}(x,z)\}$ for $1 \leq i \leq n-1$ and the clauses $\{T_1(x, f(x))\}$ and $\{T_1(x, g(x))\}$.*

Theorem 2.5.36 *Subterm-size bounded binary resolution has double exponential complexity with respect to the duplication complexity measure M_{dup}. If the subterm factoring operation is used, the complexity becomes single exponential.*

Proof. Recall that we only save resolvents whose subterm size is within the size bound, and that this size bound is gradually increased until a proof is found. Suppose S is a set of clauses. Consider the Herbrand set T for S such that $s^{sub}(T)$ is minimal. Then $s^{sub}(T) \leq M'_{dup}(S)$. However, it suffices to generate clauses of subterm size exponential in $s^{sub}(T)$ to obtain a proof. There can be at most a double exponential number of clauses generated within this size bound. If the subterm factoring operation is used, we can guarantee that the clauses generated never have a subterm size larger than $2 * M'_{dup}(T)$, obtaining the exponential bound.

Although we cannot prove the lower bound in general at present, we can do it for many resolution strategies using the clause set $T^n_{fg} \cup S_{PQR}$. In order to generate a refutation from S_{PQR}, we need to generate terms having exponential subterm size, and within this size bound, T^n_{fg} can generate a double exponential number of clauses. This works for P_1 deduction, hyper-resolution, A-resolution, and UR-resolution. The same clause set with signs of literals reversed gives the lower bound for negative resolution.

$\square$

In the case of resolution, it does not seem to matter much in general which kind of a bound is used, as far as the number of exponentials is concerned, since all the strategies considered have a double exponential complexity. The exception is a subterm size bound with the subterm factoring operation allowed, which has a much better complexity. It might be worthwhile implementing this strategy and testing it. This is one advantage of such complexity analysis, namely, it suggests possibilities and combinations that one otherwise might never consider.

We consider one more kind of clause set that seems to explain to some extent the success of resolution in practice.

Theorem 2.5.37 *If S is UR-resolvable and has duplication complexity n, then there is a UR-refutation from S having less than n UR resolu-*

tion steps.

Proof. Consider the ground level. Suppose S is a unit resolvable set of ground clauses. Let $C_1, C_2, \ldots, C_n$ be a UR refutation, that is, each C_i is either in S or it is a UR-resolvent of previous clauses. Note that UR-resolvents are unit clauses. Also, a given clause need not appear more than once in this proof. If C_i is a UR-resolvent, then C_i is $\{L\}$ for some literal L in an input clause D_i in S. We claim that these clauses D_i are all distinct in a minimal UR-refutation. The only way this can fail is if two literals from some D_i appear in the proof in different places, say $D_i = D_j$ with $i < j$. This implies that all but one of the literals of D_i resolved with unit clauses, and all but one of the literals of D_j resolved with unit clauses. Thus there are enough unit clauses to resolve away all the literals of D_j, and we could have derived the empty clause instead at step j. This shows that the length of this proof need not be larger than the number of clauses in S used in the proof, that is, the duplication complexity of S. Lifting this result to the first-order case, we obtain the theorem.

□

Theorem 2.5.38 *Length-bounded binary resolution is of single exponential complexity with respect to the duplication measure M_{dup} for UR resolvable clause sets S.*

Proof. By the preceding theorem, we only need to consider proofs of length polynomial in the duplication complexity.

□

This may explain why resolution (without subterm factoring) performs well in practice, since many common clause sets are UR resolvable and have short proofs, that is, have small duplication complexity. We note that common implementations of resolution tend to prefer short proofs and short clauses, which tends to approximate length-bounded UR resolution or even length-bounded binary resolution.

2.5.3 British Museum Methods

We now try to determine some of the reasons for the various bounds that have been proved above. We show that some very simple-minded strategies achieve the same or better bounds, which implies that the bounds do not depend on detailed features of the methods. This also raises the question of whether some of these simple methods should be implemented to see if they perform well in practice.

Theorem 2.5.39 *Consider the following method, which we call* sub-term size clause enumeration.

> *For each size bound m, enumerate all ground instances D of clauses C of S such that $s^{sub}(D) \leq m$. Test this set of ground clauses for satisfiability.*

We call this a British Museum method. *This method is then double exponential with respect to the duplication complexity measure M_{dup}. If we only test subsets of ground clauses of size m or less for satisfiability, the method becomes single exponential.*

Proof. When m reaches the duplication complexity of S, we will obtain a proof. Within this size bound, we can generate an exponential number of ground clauses. It takes double exponential time to test their satisfiability. By considering small subsets, this worst-case time can be reduced to single exponential. Clause sets to achieve these bounds are easily constructed (assuming propositional satisfiability requires exponential time). For example, we can combine an unsatisfiable set of small duplication complexity and literal size larger than n with B_n or T_{fg}. □

Also, the good performance of Davis and Putnam's method in practice may mean that the double exponential bound is single exponential in practice. The fact that we can obtain performance that compares favorably with many versions of resolution and clause linking, suggests that such methods merit more study. That is, instance-based strategies, which generate ground instances of clauses and test for satisfiability, may be more effective than is commonly supposed.

Theorem 2.5.40 *Consider the following method, which we call* linear size clause enumeration.

> *For each size bound m, enumerate all ground instances D of clauses C of S such that $s^{lin}(D) \leq m$. Test this set of ground clauses for satisfiability.*

This is also a British Museum method. This method is then triple exponential with respect to the duplication complexity measure M_{dup}. If we only test subsets of ground clauses of size m or less for satisfiability, the method becomes double exponential.

Proof. When m becomes exponential in the duplication complexity of S, we will obtain a proof. Within this size bound, we can generate a double exponential number of ground clauses. It takes triple exponential time to test their satisfiability. By considering small subsets, this worst-case time can be reduced to double exponential. Clause sets to achieve these bounds are easily constructed (assuming propositional satisfiability requires exponential time).

□

We now present a method with good asymptotic performance even without any special device such as testing small subsets for satisfiability. If such methods are not good in practice, it would be interesting to refine the analysis to understand why.

Theorem 2.5.41 *Consider the following method, which we call* subterm size clause set enumeration.

> *For each size bound m, generate one by one all sets T of ground instances of S whose subterm size is not larger than m. Test each set T for satisfiability.*

This method is of single exponential complexity with respect to the duplication complexity measure M_{dup}.

Proof. The number of such T is exponential in m, and when m reaches $M'_{dup}(S)$ we will find a proof. Also, the test for satisfiability will take time exponential in m.

□

2.5.4 Other Methods

We now discuss the matings method of Andrews [And81].

Definition 2.5.42 *Suppose S is a set of clauses and S' is an amplification of S. A* path *in S' is a mapping ϕ from S' to literals such that for each $C \in S'$, $\phi(C) \in C$. A* mating *is a set of pairs of literals of S', having opposite sign. A mating $\mathcal{M}$ is* spanning for *S' if for every path in S' there is a pair (L, M) in $\mathcal{M}$ such that both L and M are in the (image of the) path.*

The matings method is to generate amplifications S' of S and find matings that are spanning for S'. For each mating, one then looks for a simultaneous most general unifier of all the pairs in the mating. If this can be found, then S is unsatisfiable. This complete method essentially guides the instantiation of S' to achieve unsatisfiability, and does the unsatisfiability test before the instantiation, in contrast to clause linking, which does it afterwards. This method works well on many second-order logic problems.

Theorem 2.5.43 *The matings method is single exponential (in time) with respect to the duplication complexity measure M_{dup}.*

Proof. For this, we assume that there is an increasing size bound m, and that for each m, we examine all amplifications S' of S whose subterm size is not more than m. For each such S', we seek a mating as specified. The number of such amplifications is exponential in the duplication complexity. The number of matings is also exponential, as is the test for the spanning property. The unification may be done in polynomial time. Thus the method overall is single exponential in the duplication complexity.

□

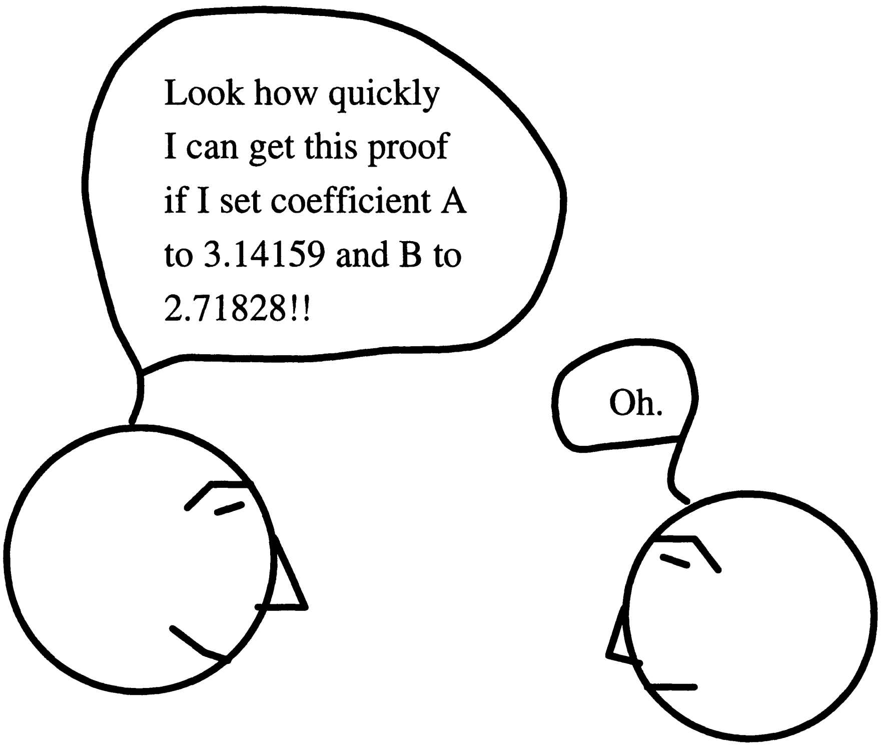
Theorem 2.5.44 *Length-bounded model elimination is of double exponential complexity with respect to the duplication complexity measure M_{dup}.*

Proof. Suppose S is a set of clauses and T is a Herbrand set for S such that the subterm size of T is minimal. Let n be the duplication complexity of S. Then n is not smaller than the subterm size of T. It follows that the number of literals in T is bounded by n. Therefore, one only needs to consider chains of length n to get a proof. The entire proof can then have a length exponential in n, leading to a double exponential complexity altogether.

As for the lower bound, in [Pla94b] we gave propositional clause sets for which model elimination requires exponential length proofs. By adding $S_{fg} \cup \{P(a)\}$ to such a clause set, model elimination will generate a double exponential number of clauses altogether (assuming the search is started from the literal $P(a)$).

□

Note that these results about duplication complexity seem to imply a dramatic difference in the efficiencies of various strategies on theorems that have a small number of clause instances needed but of unknown literal size, since the complexities vary from single to quadruple exponential. However, traditional strategies are single or double exponential in complexity, generally double, and it is only the clause linking-related strategies that can have a worse complexity for certain search strategies. A couple of variants of clause linking have single exponential complexity, as well as some enumerative strategies and matings. One variant of resolution has single exponential complexity, but appears to require subterm factoring to achieve it, an operation rarely if ever implemented to date.


Look how quickly I can get this proof if I set coefficient A to 3.14159 and B to 2.71828!!
Oh.

not inference based. From now on we mostly derive just upper bounds on the complexity.

To understand the complexity results, it helps to observe that if there is a proof within a given literal size measure, then the proof can involve an exponential number of clauses and an exponential number of literals.

2.6.1 Clause Linking

Theorem 2.6.1 *Depth-bounded clause linking has a quintuple exponential complexity with respect to the subterm size proof complexity measure M_{sub}. Length-bounded clause linking has a triple exponential complexity with respect to the subterm size proof complexity measure.*

Proof. Here we will just prove the upper bounds. If the subterm size proof complexity measure of S is n, then the largest clause can have a subterm size measure of at most kn where k is the maximum number of literals in a clause in S. (Note that $k < s^{lin}(S)$.) Thus the duplication complexity of S can be at most single exponential in kn. The depth required for a clause linking refutation can be proportional to $s^{lin}(T)$ where T is a Herbrand set for S. Thus the depth required may be double exponential in kn. This can generate clauses of triple exponential subterm size complexity, and there can be a quadruple exponential number of them. Applying a satisfiability procedure raises the time to quintuple exponential.

For length-bounded clause linking, we need proofs of length double exponential in kn. There can be at most a triple exponential number of such proofs. Each proof will be tested separately for satisfiability in double exponential time, leading to a triple exponential bound overall.

□

Depth-bounded clause linking can be reduced to quadruple exponential complexity by testing small subsets for satisfiability, as before.

Theorem 2.6.2 *Linear-size bounded clause linking has a triple exponential complexity with respect to the subterm size measure M_{sub}.*

Subterm-size bounded clause linking also has triple exponential complexity. However, if subterm factoring is used, this latter bound can be reduced to double exponential.

Proof. Again we just prove the upper bound. First, the linear size complexity can be exponential with respect to the subterm size measure. There are a double exponential number of clauses having a linear size within this exponential bound. It requires triple exponential complexity to test them for satisfiability.

For a size bound based on the subterm size measure, we saw in theorem 2.5.22 that clause linking may need to generate instances whose subterm size is exponential with respect to the subterm size proof complexity measure. There may be a double exponential number of clauses within the size bound. The satisfiability test raises this to triple exponential. However, if subterm factoring is used, we can reduce the bound by an exponential, because we only need to generate instances whose subterm size is bounded by $2 * M'_{dup}(S)$, by theorem 2.5.25.

□

In this case we can't use the small subsets idea, because we know nothing more about the size of a Herbrand set for S. However, the good performance of satisfiability tests in practice may mean that the usual performance is an exponential better. In particular, if we assume that Davis and Putnam's method runs in expected polynomial time, then the time bound for subterm-size bounded clause linking with subterm factoring becomes expected single exponential.

We now consider CLIN-D. For this, we need to modify the binary counter clause set B_n, as follows:

Definition 2.6.3 *The* modified binary counter *clause set* B'_n *consists of the following clauses:*

$\{p_i(X_1, X_2, \ldots, X_{n-1}, 1), \neg p_j(X_1, X_2, \ldots, X_{n-1}, 0)\}$,
$1 \leq i, j \leq 2$
$\{p_i(X_1, X_2, \ldots, X_{n-2}, 1, 0), \neg p_j(X_1, X_2, \ldots, X_{n-2}, 0, 1)\}$,
$1 \leq i, j \leq 2$
$\ldots$
$\{p_i(1, 0, 0, \ldots, 0), \neg p_j(0, 1, 1, \ldots, 1)\}$,
$1 \leq i, j \leq 2$
$\{p_i(0, 0, 0, \ldots, 0)\}$,
$1 \leq i \leq 2$
$\{\neg p_j(1, 1, 1, \ldots, 1)\}$,
$1 \leq j \leq 2$

Theorem 2.6.4 *CLIN-D is of double exponential complexity with respect to the subterm size proof complexity measure M_{sub}.*

Proof. Suppose S has subterm size proof complexity $M_{sub}(S)$ equal to n. Then there is a Herbrand set T having at most a number of literals that is exponential in n. Recall that the number of literals in a clause in S is bounded by $s^{lin}(S)$. Thus $|T|$ is exponential in $ns^{lin}(S)$, and the duplication complexity of S is exponential in $ns^{lin}(S)$. This counts as single exponential in n, according to our conventions. By theorem 2.5.31, CLIN-D is of single exponential complexity with respect to the duplication complexity. Therefore CLIN-D is of at worst double exponential complexity with respect to the subterm size proof complexity measure. The lower bound is obtained from B'_n, noting that this proof requires exponentially many instances. Each instance can be chosen in two or more ways, due to the subscripts i and j. Thus there are a double exponential number of choices, and CLIN-D will take double exponential time.

□

The preceding result holds even if we use some kind of subterm-size bounded CLIN-D, and also applies to the linear size proof complexity measure M_{lin}. Note also that B'_n is UR-resolvable, so the bound holds for UR-resolvable sets, too.

2.6.2 Resolution

Theorem 2.6.5 *Depth-bounded binary resolution is of triple exponential complexity with respect to the subterm size proof complexity measure M_{sub}.*

Proof. Suppose S has subterm size complexity n. Then there is a Herbrand set T having at most a number of literals that is exponential in n. Thus there is a binary resolution proof of depth exponential in n. Within this depth, we can generate clauses of double exponential subterm size complexity, and there can be a triple exponential number of them overall.

□

This bound is actually better than for depth-bounded clause linking. It does not appear that length-bounded binary resolution will do any better.

Theorem 2.6.6 *Linear-size bounded binary resolution has a triple exponential complexity with respect to the subterm size proof complexity measure M_{sub}.*

Proof. Suppose S has subterm size complexity n. Then there is a Herbrand set T having literals whose linear complexity is at most exponential in n. The number of literals can therefore be at worst double exponential in n. Since a clause is a set of literals, the number of clauses generated within the size bound is at most triple exponential in n.

□

Theorem 2.6.7 *Subterm-size bounded binary resolution has a triple exponential complexity with respect to the subterm size proof complexity measure M_{sub}. With subterm factoring, this bound can be reduced to double exponential. For UR-resolvable clause sets, with subterm factoring, the bound is single exponential.*

Proof. Suppose S has subterm size complexity n. Then there is a Herbrand set T having literals whose subterm size complexity is at most n. We saw in section 2.5 a Horn set S_{PQR} in which resolution needs to generate literals having a subterm size exponential in n. The number of literals can therefore be at worst double exponential in n. Since a clause is a set of literals, the number of clauses generated within the size bound is at most triple exponential in n. With subterm factoring, the bound can be reduced by an exponential, because we only need to generate clauses having a subterm size bounded by $2 * M'_{dup}(S)$ by theorem 2.5.25. For UR-resolvable clause sets, one never needs to generate clauses having more literals than appear in a clause in S. Thus the maximum subterm size of a generated clause will be $ns^{lin}(S)$. The number of generated clauses, and the work, will be exponential in this quantity, which still qualifies as single exponential in our formalism.

$\square$

This is the same complexity as clause linking with the same kind of a size bound, but we miss the practical efficiency of the propositional satisfiability test.

Theorem 2.6.8 *For UR-resolvable clause sets, subterm-size bounded UR resolution has a double exponential complexity with respect to the subterm size proof complexity measure M_{sub}. With subterm factoring, this bound can be reduced to single exponential.*

Proof. Similar to the above theorem.

$\square$

2.6.3 British Museum Methods

Theorem 2.6.9 *Let's reconsider the following subterm size clause enumeration method:*

> *For each size bound m, enumerate all ground instances D of clauses C of S such that $s^{sub}(D) \leq m$. Test this set of ground clauses for satisfiability.*

This method has double exponential complexity with respect to the subterm size proof complexity measure M_{sub}.

Proof. The number of such D generated is exponential in the subterm size complexity measure of S. The work to test satisfiability is then double exponential.

□

Again, the good performance of satisfiability methods may reduce this to single exponential in practice. If we assume Davis and Putnam's method runs in expected polynomial time, the time bound would be expected single exponential. The good asymptotic performance of this method in comparison to others listed here is remarkable, especially in view of the fact that it does not use unification or resolution. Perhaps this method should be implemented and tested. Refinements of this method to guarantee no pure literals, for example, are easy to imagine.

Theorem 2.6.10 *Consider the following linear size clause enumeration method:*

> *For each size bound m, enumerate all ground instances D of clauses C of S such that $s^{lin}(D) \leq m$. Test this set of ground clauses for satisfiability.*

This method has triple exponential complexity with respect to the subterm size proof complexity measure M_{sub}.

Proof. The linear size needed for the proof can be exponential in the subterm size measure of S. The number of such D generated can be double exponential in the subterm size complexity measure of S, and the work to test satisfiability can be triple exponential.

□

In this case, we can apply the satisfiability procedure to subsets of size exponential in m and reduce the upper bound by one exponential.

Theorem 2.6.11 *Consider the following subterm size clause set enumeration method:*

> *For each size bound m, generate one by one all sets T of ground instances of S whose subterm size is not larger than m. Test each set T for satisfiability.*

This method is of double exponential complexity with respect to the subterm size proof complexity measure M_{sub}.

Proof. If S has subterm size complexity n, then there will be a Herbrand set T for S all of whose literals have subterm size n or less. This T may have a number of literals that is exponential in n. Thus m may need to become exponentially large before a proof is found. The number of T generated can then be double exponential in n. The time to test each one for satisfiability is also double exponential, leading to a double exponential bound overall.

□

This bound may not be reduced to expected single exponential time, even if satisfiability routines run in expected polynomial time.

2.6.4 Other Methods

Theorem 2.6.12 *The matings method is double exponential with respect to the subterm size proof complexity measure M_{sub}.*

Proof. If S has subterm size complexity n, then the matings method may need to construct an amplification having an exponential number of clauses in it to get the proof. Processing all amplifications up to this size (or any one of the large ones) will take double exponential time. The lower bound is obtained from the set B'_n of clauses.

□

We note that matings does not appear to take advantage of the fast performance of satisfiability algorithms on many examples. This could imply that clause linking with a size bound based on subterm size will be better with respect to this measure. We note also that the lower bound is still valid for the linear size proof complexity measure, and also for UR-resolvable sets, since B'_n is UR-resolvable. These lower bounds probably apply to many connection methods, too; it is possible, on the other hand, that the behavior of connection methods is like that of model elimination.

Theorem 2.6.13 *Model elimination has triple exponential complexity with respect to the subterm size proof complexity measure M_{sub}.*

Proof. Suppose S has subterm size proof complexity n. Then there is a Herbrand set T for S such that T has a number of clauses exponential in n. Thus the number of literals in T is exponential in n. This means that model elimination may need to construct chains of exponential length, and the proof overall may be double exponential in length. (It really helps to consider the tableau formulation to see this easily.) Within a double exponential bound, there can be at worst a triple exponential number of proofs.

□

We observe that this complexity is worse than that of the best resolution variant, the best clause linking variant, and matings.

The subterm size measure M_{sub} seems most appropriate when the Herbrand set has many clauses, but the literals are small. For such problems, clause linking (with a suitable size bound) is better than binary resolution, matings, and model elimination. Although a resolution variant also has double exponential complexity, that variant cannot take advantage of the good performance of satisfiability algorithms in practice. Also, the subterm size clause enumeration method has a similar asymptotic complexity, and also takes advantage of the satisfiability test. However, it does not incorporate unification, as does clause linking.

For UR-resolvable clause sets, subterm-size bounded binary resolution with subterm factoring and UR resolution with subterm factoring have single exponential complexity with respect to this measure, regardless of assumptions about the performance of the satisfiability procedure. This shows a definite advantage of these strategies, since UR-resolvable (or nearly UR-resolvable) clause sets are fairly common among easy problems. Only if we assume that a satisfiability procedure runs in expected polynomial time can we obtain nearly comparable performance from other methods.

2.7 The Linear Size Measure

In general, when we analyze the complexity of strategies that search using a size-bounded search strategy, the analyses of the complexities of strategies with respect to the linear size proof complexity measure M_{lin} are the same as those with respect to the subterm size proof complexity measure M_{sub} using subterm factoring, since the number of literals within a linear size bound is exponential. That is to say, the performance of method X which searches according to subterm size or linear size, with respect to the linear proof complexity measure, will be asymptotically the same as the performance of method X (with subterm factoring) which searches according to a subterm size bound with respect to the subterm size proof complexity measure. This is because the subterm size measure is linearly bounded by the linear measure. Also, upper bounds for the subterm size measure are valid for the linear size measure, too, since subterm size is bounded by linear size. Thus the literals of subterm size m or less are a superset of the literals of linear size m or less. A disadvantage of the linear measure is that it does not relate so well to the duplication complexity measure as the subterm size proof complexity measure does (see Table 2.1 below). We list a number of results, mostly without proof.

Theorem 2.7.1 *Depth-bounded clause linking has quintuple exponential complexity with respect to the linear size proof complexity measure M_{lin}. Length-bounded clause linking has a triple exponential complexity with respect to the linear size proof complexity measure.*

Theorem 2.7.2 *Linear-size or subterm-size bounded clause linking is of double exponential time complexity with respect to the linear size proof complexity measure M_{lin}.*

Proof. Suppose S has linear proof complexity n. Then there is a Herbrand set T all of whose literals have linear size n or less. When the size bound reaches n, clause linking will find a proof. There can be at most an exponential number of clauses generated within this size bound. The time to test them for satisfiability is at most double exponential.

□

If one assumes that Davis and Putnam's method runs in expected polynomial time, the time is reduced to expected single exponential.

Theorem 2.7.3 *CLIN-D is of double exponential complexity with respect to the linear size proof complexity measure M_{lin}.*

We note that this result also applies to linear-size bounded CLIN-D and applies to UR-resolvable clause sets, and probably extends to many connection methods.

Theorem 2.7.4 *Depth-bounded binary resolution has triple exponential complexity with respect to the linear size proof complexity measure M_{lin}.*

Theorem 2.7.5 *Linear-size or subterm-size bounded binary resolution has a double exponential complexity with respect to the linear size proof complexity measure M_{lin}. For UR-resolvable clause sets, the bound is single exponential.*

Theorem 2.7.6 *For UR-resolvable clause sets, linear-size or subterm-size bounded UR resolution has single exponential complexity with respect to the linear size proof complexity measure M_{lin}.*

Theorem 2.7.7 *Linear or subterm size clause enumeration has double exponential complexity with respect to the linear size proof complexity measure M_{lin}. If one assumes that Davis and Putnam's method runs in expected polynomial time, this is reduced to expected single exponential.*

Theorem 2.7.8 *Subterm or linear size clause set enumeration has double exponential complexity with respect to the linear size proof complexity measure M_{lin}.*

Theorem 2.7.9 *Matings has a double exponential complexity with respect to the linear size proof complexity measure M_{lin}.*

Theorem 2.7.10 *Model elimination has a triple exponential complexity with respect to the linear size proof complexity measure M_{lin}.*

In general, with respect to this measure, linear or subterm-size bounded clause linking and linear or subterm-size bounded clause enumeration methods have an advantage over all other methods, since they not only have a double exponential complexity, but can also take advantage of the fast performance of satisfiability algorithms. Clause linking also has the advantage of unification, which enumeration methods lack. For UR-resolvable sets, linear-size bounded binary resolution and linear-size bounded UR-resolution are asymptotically the fastest, having a single exponential time bound.

2.8 Sets with Large Literals

We now attempt to find a measure for which resolution and other traditional strategies perform well, to help explain their popularity. The measure we construct is somewhat artificial, corresponding to our intuition about the weaknesses of these traditional methods. However, this measure is reasonable for many small, toy problems, and does help to give us some insight.

Definition 2.8.1 *The* literal count *of a clause set S is $\sum_{C \in S} |C|$, where $|C|$ is the number of literals in C. Thus the literal count of S is the total number of occurrences of literals in S.*

Consider clause sets whose literal count and duplication complexity is fixed, but the subterm size $s^{sub}(S)$ may vary. Intuitively, such clause sets have few literals and short proofs but large terms. We then may ask how the complexity of various strategies depends on the subterm size for such clause sets. We refer to this measure as *duplication-bounded subterm complexity.*

We will need some results about literal counts for minimal unsatisfiable clause sets.

Theorem 2.8.2 *In any minimal unsatisfiable set S of ground clauses with d elements, each clause has at most $d-1$ literals. Also, there is an A-resolution refutation of depth at most $d-1$.*

Proof. Consider an A-resolution refutation of S in which no tautologies are derived. Such a refutation must exist. Suppose a clause C in S has d (distinct) literals $L_1, L_2, \ldots, L_d$. Let $M_1, M_2, \ldots, M_d$ be the literals that resolve against $L_1, L_2, \ldots, L_d$, respectively; thus M_i is the complement of L_i. Suppose M_i descends from a clause C_i in the input set S. None of these C_i can be C, or else C would be a tautology, since it can resolve with itself. Since S has only d elements, for some $i \neq j$ we have $C_i = C_j$. Thus C_i contains both literals M_i and M_j. Suppose L_i resolves before L_j. The resolvent on L_i will then contain both L_j (from C) and M_j (from C_i), since we are doing A-resolution, which resolves predicate symbols in a fixed order. However, L_j and M_j are complementary, so this resolvent is a tautology, contradicting our assumption that this proof contains no tautologies.

To see that an A resolution refutation without tautologies can have depth at most d, consider a sequence $D_1, D_2, \cdots, D_d$, each D_k being a resolvent of D_{k-1} and some clause containing M_i. Reasoning as above, some M_i and M_j descend from the same input clause, and if M_i resolves first, then the resolvent will contain M_j. But in this sequence of clauses D_k, the literal M_j will never resolve away before D_j is derived, since we are doing A-resolution. So D_j will contain both M_j and L_j, and is therefore a tautology, contradiction.

□

Corollary 2.8.3 *If S is a minimal unsatisfiable set of ground clauses with d elements, then the literal count of S is at most $d(d-1)$.*

That this is optimal to within a constant factor may be seen from the clause set

P_1
$\neg P_1, P_2$
$\neg P_1, \neg P_2, P_3$
$\ldots$
$\neg P_1, \neg P_2, \cdots, \neg P_{n-1}, P_n$
$\neg P_n$

which is minimal unsatisfiable and has $n+1$ clauses and literal count $(n^2+n+2)/2$.

Theorem 2.8.4 *Consider clause sets S having bounded literal count and duplication complexity. For such clause sets, depth-bounded resolution has a polynomial dependence on $s^{sub}(S)$.*

Proof. Suppose the duplication complexity is bounded by d. By theorem 2.8.2, the depth of the proof need be no greater than d. Each depth of resolution can at most double the subterm size of the clauses, and so one needs only clauses having subterm size $2^d s^{sub}(S)$ for the proof. Since S is of bounded literal count, the number of clauses that can be derived in depth d is bounded by some function of d and the literal count of S that does not depend on $s^{sub}(S)$. Since resolving two clauses takes work polynomial in their subterm sizes, the total work is a polynomial in $s^{sub}(S)$ times a function that depends only on d and the literal count of S. Since the duplication complexity of S and its literal count are kept constant, the total work has a polynomial dependence on $s^{sub}(S)$.

□

Corollary 2.8.5 *For minimal unsatisfiable clause sets S having bounded duplication complexity, depth-bounded resolution has a polynomial dependence on $s^{sub}(S)$.*

Proof. This follows from the theorem and corollary 2.8.3.

□

This is relevant because many theorems on which provers are tested are minimal unsatisfiable clause sets, or only have a small number of clauses not needed in the proof. Furthermore, the proofs are often relatively short (or else the prover could not find them), implying that the duplication complexity is also small. Thus the conditions of the theorem and corollary, though restrictive, are reasonable for many of the theorems typically used to test theorem provers.

Similar polynomial dependencies on the subterm size can easily be established for length-bounded resolution, depth or length-bounded CLIN-D, matings, model elimination, and connection methods. With respect to the linear size $s^{lin}(S)$, one obtains an exponential dependence.

2.9 Unit Resolvable Clause Sets

The fact that resolution is so often used in practice leads one to question why, in view of its often poor asymptotic complexity. We have just given one justification for this, in terms of duplication-bounded subterm complexity. We believe that another reason is that many proofs may be obtained purely by unit resolution, and for UR-resolvable clause sets, by theorems 2.6.7 and 2.6.8, subterm-size bounded resolution with subterm factoring and subterm-size bounded UR-resolution with subterm factoring perform better than any other methods with respect to the subterm size measure M_{sub}. We note that Horn sets always have UR proofs, and are very common. The version of resolution that is typically implemented is linear-size bounded, not subterm-size bounded. But even linear size-bounded UR resolution has a complexity that is single exponential with respect to the linear proof complexity measure M_{lin} for unit resolvable clause sets, better than any other method except UR-resolution. The subterm factoring operation becomes significant for the subterm size measure M_{sub}; with respect to M_{sub}, linear or subterm-size bounded UR resolution has a double exponential complexity even on unit resolvable clause sets, as the set S_{PQR} of clauses shows. With subterm factoring, subterm-size bounded UR resolution has single exponential complexity with respect to the subterm size proof complexity measure M_{sub}, on unit resolvable clause sets.

All the other methods do not seem to be sensitive to the unit resolvable property. That is, they seem to have the same behavior on unit resolvable sets (even using size-bounded search) as they do with respect to arbitrary sets.

Even subterm-size bounded clause linking is double exponential with respect to the subterm or linear size proof complexity measure (if subterm factoring is used), so binary resolution and UR resolution on UR-resolvable sets have an advantage here, being single exponential. If we assume the satisfiability test is polynomial on the average, subterm-size bounded clause linking is better, but still not as fast in the worst case. Length-bounded resolution is single exponential with respect to duplication complexity for UR-resolvable clause sets, by theorem 2.5.38. Since UR-resolvable clause sets with small duplication complexity are common in toy problem sets, this may help to explain the good performance of resolution there. For UR-resolvable clause sets, the proof for an instance-based method will be found by the unit simplification part of Davis and Putnam's method, so for clause linking and similar instance-based strategies, the propositional part will be polynomial once enough instances have been found. So even here, clause linking can compete with resolution if we cut off long runs of the propositional satisfiability procedure, but this would sacrifice completeness in general. If Davis and Putnam's procedure is called too early, the clause set will be satisfiable and in practice, such clause sets usually run quickly for Davis and Putnam's method. To make clause linking exponential on unit resolvable sets, with the loss of completeness, we can restrict Davis and Putnam's method to do only unit simplifications. In fact, these simplifications are done by clause linking before Davis and Putnam's method is even called as a rule anyway, and we often find proofs there. This gives something of a theoretical justification for why the unit rules are important for clause linking.

So clause linking with a subterm size bound and subterm unification seems to extend subterm-size bounded resolution's good behavior on UR-resolvable clause sets relative to M_{sub} to the general case, assuming good behavior from the propositional satisfiability test. Although matings is of single exponential complexity relative to the duplication complexity measure M_{dup}, and also makes essential use of unification, it is double exponential relative to the subterm or linear size proof complexity measures, even for UR resolvable sets, and is therefore at a drastic disadvantage relative to UR resolution.

2.10 Implications for Choice of Strategy

When there are few instances needed for the proof but the literals are comparatively large, then the duplication proof complexity measure seems most appropriate. The best strategies then have exponential complexity; they are the following: subterm-size bounded clause linking, with subterm factoring; subterm-size bounded binary resolution, with subterm factoring; subterm size clause enumeration; subterm size clause set enumeration; and matings.

When there are many instances and small literals with respect to the subterm size measure (especially for highly non-Horn sets), then the subterm size proof complexity measure seems most appropriate. In this case, the best bound for any strategy is double exponential. The best strategies would seem to be the following: subterm-size bounded clause linking, with subterm factoring; subterm-size bounded binary resolution, with subterm factoring; subterm size clause enumeration; subterm size clause set enumeration; and matings. These are exactly the same strategies as above.

Assuming that Davis and Putnam's method (or some other satisfiability test) runs in expected polynomial time, then the methods that have expected single exponential time with respect to the subterm size measure are the following: subterm-size bounded clause linking with subterm factoring, and subterm size clause enumeration. This shows a distinct advantage of clause linking with subterm factoring over all other non-enumerative methods considered, and shows the importance of using a size bound. Of course, clause linking also has the advantage over enumerative methods of using unification.

When there are many instances and small literals with respect to the linear size measure, then the linear size proof complexity measure seems to be most appropriate. In this case, assuming that Davis and Putnam's method runs in expected polynomial time, the strategies running in expected single exponential time with respect to this measure are size-bounded clause linking with or without subterm factoring, and subterm or linear size clause enumeration. This shows a distinct advantage of size-based clause linking without subterm factoring, which is interesting because that is the only method that has been implemented, and it has often performed well in practice, even much better than resolution [LP92] on some problems. This analysis helps us to understand why, and for which types of clauses this is likely to occur.

For unit resolvable sets (including Horn sets), when the literals needed are small but the proof may be long, then the subterm size and linear size measures seem most suitable. With respect to the former, subterm size-bounded (binary and UR) resolution with subterm factoring have a single exponential complexity, and therefore are better than any other strategy listed above. For this, subterm factoring is essential. We gave in section 2.5 the set S_{PQR} of Horn clauses where resolution (and UR resolution) generate literals having a subterm size exponentially larger than necessary, and there can be a double exponential number of unit clauses generated within this size bound, leading to a double exponential bound without subterm factoring. With respect to the linear measure, subterm or linear-size bounded (binary and UR) resolution without subterm factoring are of single exponential complexity. Clause linking and subterm size clause enumeration come close, but they can only reach expected exponential complexity, and this under the assumption that the used satisfiability procedure runs in expected polynomial time. This suggests also that if clause linking or enumerative methods are used, then they should employ special rules for unit clauses to improve their performance on unit resolvable sets.

Another interesting class of clauses are those with small duplication complexity and small numbers of clauses and literals but for which $s^{sub}(S)$ may be large. That is, as we keep the duplication complexity of S fixed, and keep the number of clauses and their number of literals fixed, and increase $s^{sub}(S)$ (or, equivalently, $M_{sub}(S)$), as is done in S^n_{PQ}, we can ask how the complexity of various strategies is affected. This aspect was discussed in section 2.8. Clause linking and enumerative strategies will be exponential (or worse) with respect to $s^{sub}(S)$, since they have a hard time generating large terms. Size-bounded resolution will also be exponential with respect to $s^{sub}(S)$. However, it is not difficult to show that the following strategies are polynomial with respect to $s^{sub}(S)$: depth-bounded resolution, length-bounded resolution, matings, CLIN-D, connection methods, and model elimination. In short, all of the traditional unification-based strategies do comparatively well here. This is because these methods depend only on the propositional structure of proofs, and are not very sensitive to the sizes of terms produced by unifications (assuming terms are represented efficiently in a directed acyclic graph representation). This shows that these strategies are good when the proofs are relatively short but the terms needed may be large, which agrees with common sense and experience. However, this makes it more difficult to say which strategy is best in general, because the strategies that perform well with respect to term size, generally do not do well with respect to the linear or subterm size proof complexity measures. Also, of the strategies that perform well with respect to term size, only length-bounded CLIN-D, matings, and probably the connection methods are exponential with respect to duplication complexity, and therefore are to be preferred. But these strategies do not perform well even for UR resolvable sets with respect to the subterm or linear size measures.

A further interesting class of problems are UR-resolvable clause sets with small duplication complexity. We commented on these clause sets in theorem 2.5.38. We note that length-bounded UR-resolution and length-bounded binary resolution have reasonably good performance (single exponential) with respect to the duplication measure in this case, as do CLIN-D and matings. The subterm-size based enumerative methods also perform well here, assuming Davis and Putnam's method runs in polynomial time.

We therefore have six different classes of clauses to consider: those with small duplication complexity M_{dup}, those with a small subterm size measure M_{sub}, those with a small linear size measure M_{lin}, unit resolvable sets, unit resolvable sets with a small duplication complexity, and clause sets having large term size and a small duplication complexity.

We now discuss the overall performances of some of the strategies. Resolution is not single exponential complexity except in the unit resolvable case, and for one of its variants with respect to the duplication complexity. However, the unit resolvable case is very common, making resolution often good. Note that smallest first (that is, size-bounded) search (as resolution is often implemented) helps to do many unit resolutions. Also, the fact that resolution uses unification is an advantage. Another good feature of resolution is its deletion criteria such as subsumption deletion, whose value does not appear in this analysis.

Clause linking has a wide variety of asymptotic behaviors, depending on the manner of search and the size bound used. However, with subterm-size bounded search and subterm factoring, it is never much worse than the best strategy, assuming good behavior from the propositional satisfiability test. The one possible exception is for clause sets with small duplication complexities and large term sizes. It is also somewhat less efficient than binary and UR resolution on UR resolvable clause sets.

We also note that subterm-size clause enumeration seems to have a surprisingly good asymptotic performance, and therefore deserves further study, despite its lack of use of unification. Another promising enumerative method that involves semantics and orderings is the *or-*

dered semantic hyper-linking strategy [Pla94a], although it may have to be adapted to use subterm size to obtain good asymptotic behavior. One reason for the promise of this method is that it can be made goal-sensitive by the appropriate use of semantics. Recall that a method is *goal-sensitive* if every inference or instance used is related to the particular clauses in the theorem being proved, as opposed to clauses that encode general axioms.

Matings has an advantage over clause linking with respect to the duplication complexity measure, because it does not require the use of small subsets or the assumption that a satisfiability procedure runs in expected polynomial time. It also does not require subterm factoring.

CLIN-D has an advantage over matings in that it can take advantage of efficiencies in Davis and Putnam's method. It works well (exponential time) with respect to the duplication complexity measure, but for the subterm size measure, it is double exponential regardless of whether the satisfiability procedure runs fast. This makes it worse than clause linking, surprisingly. It seems that the duplication by combination referred to in [Pla94b, Pla94c] is the problem here. With respect to the linear size measure, CLIN-D is double exponential regardless of whether the propositional test is fast. In general, CLIN-D is like matings and (probably) the connection methods of [Bib87] in its performance, we feel, and illustrates well their advantages and disadvantages with respect to other strategies.

Model elimination never looks very good in this analysis. However, it is a set of support strategy, and our analysis does not consider this property. Also, for Horn sets or unit resolvable sets, caching can improve the performance of model elimination, and we have not analyzed the performance of model elimination with caching for Horn sets (which are fairly common). The performance of model elimination with caching would probably be about the same as UR-resolution for Horn sets, but model elimination has the advantage of goal-sensitivity. Other methods (with caching) that would have the same advantages for Horn sets are the simplified and modified problem reduction formats [Pla82, Pla88].

The best strategies overall, when term sizes are not too large and assuming satisfiability can be done quickly on the average, are subterm size-bounded clause linking with subterm factoring and subterm-size clause enumeration. For UR-resolvable sets, subterm-size bounded UR resolution with subterm factoring has a slight advantage. For clause sets with short proofs but large term sizes, traditional strategies such as resolution and matings have an advantage. It would be interesting to find strategies that perform well on all three classes of problems. For this, it appears that a combination of strategies will be required. It is an interesting problem to determine which mix of strategies or new strategy will perform well with respect to all of these measures. It currently looks like the following combinations are attractive: 1) Subterm-size bounded clause linking with subterm factoring, or subterm-size clause enumeration, together with 2) length-bounded CLIN-D, matings, or (probably) a connection method, for problems with large terms, together with 3) subterm-size bounded UR-resolution with subterm factoring. These methods could simply be run in parallel, but it might make more sense to use clauses derived in one method as simplifiers for the other methods. This is one benefit of our complexity analysis, namely, it suggests which combinations of strategies to use to obtain good performance overall. Now one does not have to say that it is an arbitrary choice to combine UR resolution with some other method.

So we see from this discussion that we obtain some fairly definite and strong preferences between strategies based on complexity considerations, and some suggestions for good combinations of strategies. Of course, it can be that for specific problems and small values of the complexity measures that these general conclusions do not always hold. There might be other strategies with good performance, too.

Note that this analysis reveals features that were not evident from the mostly propositional analysis of [Pla94b, Pla94c]. We see, then, some of the advantages of going to a first-order framework. It would be interesting also to analyze some traditional deductive systems such as sequent style, Hilbert style, and other systems from a similar standpoint, for first-order formulas containing quantifiers.

—	M_{pd}	M_{pl}	M_{dup}	M_{sub}
M_{pd}	—	M_{pl}	poly(M_{dup})	[exp(M_{sub})]
M_{pl}	exp(M_{pd})	—	exp(M_{dup})	[dexp(M_{sub})]
M_{dup}	exp(M_{pd})	exp(M_{pl})	—	exp(M_{sub})
M_{sub}	exp(M_{pd})	exp(M_{pl})	poly(M_{dup})	—

Table 2.1: Table of relations among complexity measures

2.11 Relations among complexity measures

In the previous sections, we introduced several different complexity measures, and we analyzed the complexity of various theorem proving strategies with respect to these measures. In this section, we study the relationships among the complexity measures. These relationships allow us transfer the complexity of a strategy with respect to one measure to complexities with respect to other measures.

In Table 2.1, we list the relations among the four complexity measures: depth measure (M_{pd}), length measure (M_{pl}), duplication measure (M_{dup}), and literal size measure (M_{sub}). We use *exp* to abbreviate "exponential" and *dexp* to abbreviate "double exponential", as before.

An entry in row i and column j (i, j) denotes the worst case bound of the measure in row i with respect to the measure in column j. For example, entry $(1, 2)$ is exp(M_{pd}). It means that the length measure is exponential with respect to the depth measure, that is, $M_{pl} = O(exp(M_{pd}))$. All bounds in Table 2.1 are tight, except for those in square brackets. We now derive the entries in Table 2.1; we will include a worst case example when the tightness of a bound is not obvious.

Theorem 2.11.1 *For a set of first order clauses S, the length measure, duplication measure, and literal size measure of S are all exponentially bounded by the depth measure of S. All three bounds are tight.*

We prove theorem 2.11.1 by deriving the entries in the first column of table 2.1. For the first column, the depth measure is fixed, and we derive the worst case bounds for the other three measures. We construct

```
:- p_1(X), p_2(X), ..., p_n(X).
p_1(s(s(X))) :- p_1(X).
p_2(s(s(s(X)))) :- p_2(X).
p_3(s(s(s(s(s(X)))))) :- p_3(X).
...
p_n(s(s...s(X))) :- p_n(X).
p_1(0).
p_2(0).
...
p_n(0).
```

Figure 2.1: An example of the common prime divider [Let93]

a binary resolution proof with minimal depth bound. In an extreme case, the proof corresponds to a tree of height M_{pd} and branching factor 2. There are at most $2^{(pd+1)} - 1$ nodes in the proof, and at most 2^{pd} leaf nodes in the tree. Every leaf node corresponds to an input clause. The length measure and the duplication measure are bounded by the total number of nodes and the number of leaf nodes in the tree, respectively. Thus entry (2,1), the length measure, is $exp(M_{pd})$. Entry (3,1), the duplication measure, is also $exp(M_{pd})$. We derive the bounds of literal size by recursion. Assume the literal size of a clause at depth n is $T(n)$, that is, a clause only has literals of size less than or equal to $T(n)$. A literal can have M distinct variables, and after one step of resolution, the total size of the instances of these M distinct variables is always less than $T(n)$. If a directed acyclic graph representation of terms is used, the maximal literal size of a clause at depth $n + 1$ is less than $T(n) + T(n)$. Thus entry (4,1), which equals $T(M_{pd})$, is $exp(M_{pd})$. If a linear representation of terms is used, the maximal literal size of a clause at depth $n + 1$ is bounded by $T(n) * T(n)$, because a literal can have $T(n)$ copies of a variable, which is instantiated to a term of size $T(n)$. Thus the linear literal size measure is double exponential with respect to the proof depth. In Figure 2.1, the literal size measure is exponential with respect to to the proof depth, so the bound for entry (4,1) is tight. This is so because in order to find a proof, one needs to construct an X of the form $s^m(0)$ for an m that is exponential in

n, but the proof depth (and length) is relatively small. The number of successors in the clauses of Figure 2.1 are given by successive prime numbers, and so such an X must be divisible by the first n primes.

Theorem 2.11.2 *For a set of first order clauses S, the depth measure, duplication measure, and literal size measure of S are linearly, exponentially, and exponentially bounded by the length measure of S, respectively. All three bounds are tight.*

For the second column, the length measure is fixed, and we derive the upper bounds for the depth measure, duplication measure and literal size measure. It is trivial to see that the depth measure is always less than the length measure. Thus entry (1,2), the depth measure is M_{pl}. The duplication measure is exponential with respect to the proof depth measure, thus it is exponential with respect to the proof length measure. By the same argument, the literal size measure is also exponential with respect to the proof length measure. Thus entry (3,2) and (4,2) are $exp(M_{pl})$. The binary counter clause set B_n shows that the exponential bound for the duplication measure is tight. Figure 2.1 shows that the exponential bound for the literal size measure is tight.

Theorem 2.11.3 *For a set of first order clauses S, the depth measure, length measure, and literal size measure of S are polynomially, exponentially, and polynomially bounded by the duplication measure of S, respectively. All three bounds are tight.*

For the third column, the duplication measure is fixed. In some unsatisfiable instance set S' of an input clause set S, only M_{dup} copies of S are needed, and there are at most $M_{dup} * s^{lin}(S)$ literals in S'. We can construct an A-resolution proof for S' in which one literal is resolved away at each level. The same A-resolution proof can be lifted to prove unsatisfiability of S. The A-resolution proof has a depth of at most $M_{dup} * s^{lin}(S)$. The A-resolution proof has a length of at most $exp(M_{dup} * s^{lin}(S))$. Since A-resolution is a special case of binary resolution, the depth measure and the length measure are polynomially bounded and exponentially bounded with respect to the duplication measure, respectively. Haken [Hak85] shows that an exponential number of resolvents are needed for every resolution proof of the pigeonhole

$length(0,0).$
$length(f(X),s(Y))$:- $length(X,Y).$
$length(g(X),s(Y))$:- $length(X,Y).$
$p(X)$:- $length(X,s(s(s(\ldots s(0)\ldots)))).$
$p(X)$:- $p(f(X)),p(g(X)).$
:- $p(0).$

Figure 2.2: Duplication measure is exponential with respect to literal size measure

problem. The pigeonhole problem is a propositional problem, and its duplication measure equals the input clause size. Thus the bound of length measure with respect to duplication measure is tight. We now analyze the literal size measure. By theorem 2.2.19, the subterm size of an instance set S', $s^{sub}(S')$, is bounded by $|S'|s^{sub}(S)$, which is in turn bounded by $M_{dup} * s^{lin}(S)$. Thus the subterm size measure $M_{sub}(S)$ of S is polynomial with respect to the duplication measure, that is, entry (4,3) is $poly(M_{dup})$. The linear literal size measure is exponential with respect to the duplication measure.

Theorem 2.11.4 *For a set of first order clauses S, the depth measure, length measure, and duplication measure of S are exponentially, double exponentially, and exponentially bounded by the literal size measure of S, respectively. The bound for the duplication measure is tight.*

For the fourth column, the literal size measure is fixed. In an unsatisfiable instance set S' of input set S, the literal size of S' is $M_{sub}(S)$, and thus there are at most $exp(M_{sub}(S))$ distinct literals. By the same argument in the last paragraph, an A-resolution of depth $exp(M_{sub}(S))$ exists. Thus entry (1,4), the depth measure, is $exp(M_{sub})$. Because the length measure M_{pl} is bounded by $exp(M_{pd})$ and M_{pd} is bounded by $exp(M_{sub})$, M_{pl} is bounded by $dexp(M_{sub})$. Whether the bounds for the depth measure and the length measure are tight is an open question. The duplication measure is bounded by the number of instances needed in a proof, which is less than the number of literals in S', thus entry (3,4) is $exp(M_{sub})$. Figure 2.2 also shows that the bound in entry (3,4) is tight, since one needs an exponential number of small ground clauses to obtain a refutation.

Remarks: Note that the entries in Table 2.1 denote worst case upper bounds as opposed to analytical relations between complexity measures. Entry (i, j) does not usually have a direct relation with entry (j, i). For example, the literal size measure is exponential with respect to the depth measure, and the depth measure is also exponential with respect to the literal size measure. This occurs because the worst cases are achieved in different sets of first order problems. Although the bounds in Table 2.1 are tight, the composition of two bounds are not necessarily tight. For example, the duplication measure is exponential with respect to the length measure, which in turn is exponential with respect to the depth measure. However, the duplication measure is exponential instead of double exponential with respect to the depth measure.

The relations in Table 2.1 allow us to compare different theorem proving strategies. Assume that a theorem proving strategy is a breadth first strategy with respect to complexity measure $\mathcal{Y}_1$ and has complexity of $f(\mathcal{Y}_1)$, and $\mathcal{Y}_1$ is tightly bounded by $g(\mathcal{Y}_2)$. We argue that the strategy is likely to have a tight complexity of $f(g(\mathcal{Y}_2))$ with respect to complexity measure $\mathcal{Y}_2$. We illustrate this by an example. Length-bound binary resolution is exponential with respect to the length measure. The length measure is exponential with respect to the duplication measure. We now construct a set of first order problems $\mathcal{S}$. Each problem S in the set $\mathcal{S}$ contains the pigeonhole problem P_n of size n together with S_{fg}. The pigeonhole problem is propositionally unsatisfiable, so the duplication measure M_{dup} of S is $|P_n|$. The minimal binary resolution proof length, i.e. the length measure, of S is $exp(n)$, which is also $exp(M_{pd})$. Within the proof length of $exp(M_{pd})$, $exp(exp(M_{pd}))$ resolvents can be generated from C. Thus length-bound binary resolution has complexity $exp(exp(M_{pd}))$ on the problems in $\mathcal{S}$. Similar constructions can be carried out for many other breadth first theorem proving strategies. The basic idea is to include in the input clauses a group that exhibits the complexity of $f(\mathcal{Y}_1)$ and another group that exhibits the complexity of $g(\mathcal{Y}_1)$.

Finally, we attempt to address the question which measure is best. Of course, the appropriate measure will vary with the application. However, those measures not based on particular inference schemes seem better from a philosophical standpoint. We have to choose, then, between the duplication complexity measure and two term size measures, linear and subterm size. The subterm size measure is better related to duplication complexity, and therefore is preferable to the linear size measure. The linear size measure has the advantage of permitting simpler implementations. Still, the most natural measures seem to be subterm size M_{sub} and duplication M_{dup}. It is not clear which, if either, is better. Note that there are also other measures based on the Herbrand sets such as clause size (2 measures) and sum of clause sizes (2 measures). With respect to clause size, we can easily show that many methods are double exponential, since we can have exponentially many clauses. With respect to the sum of clause sizes measure, we can easily devise theorem proving methods that are single exponential, since we have a linear number of clauses of small size. For the clause size and sum of clause sizes measures, it is better to use subterm size, as before. Duplication complexity seems better than sum of clause sizes, however, since it is smaller, but still permits an exponential theorem prover.

For hard problems, the lengths of the proofs and the duplication complexity will likely be large. This means that even a method that is single exponential with respect to the duplication complexity will probably be too slow. However, on the other side, we mention that Andrews' matings prover is exponential with respect to the duplication complexity, but has found some fairly interesting proofs. The explanation seems to be that the expressive power of second-order logic, employed by this prover, permits shorter proofs and smaller amplifications. Still, this may be an exceptional phenomenon. Also, hard theorems will probably involve many input clauses. This makes goal-sensitivity more important, to reduce irrelevant inferences, and makes it unlikely that the clause set will be Horn or UR-resolvable. The only chance that we have to obtain such hard proofs automatically, then, is to use methods that perform well with respect to the literal size measures M_{sub} or M_{lin}. Even for complex proofs, it is conceivable that the sizes of the literals are still reasonably small, and so a method that is exponential with respect to M_{sub} has a chance of being of practical value for such hard proofs. The only methods that are expected single exponential with respect to M_{sub} or M_{lin}, assuming that a propositional satisfiability test runs in expected polynomial time, are subterm-size bounded clause linking with subterm factoring, subterm-size clause enumeration, or some other enumeration method such as ordered semantic hyper-linking [Pla94a]. It will be advantageous to choose goal-sensitive versions of these strategies. The enumerative approach becomes even more feasible when there are equations, since then the enumerative methods only need to generate ground clauses that are in normal form with respect to a term-rewriting system. Since many ground clauses are not in normal form, this increases the sizes of the terms that can be generated. All in all, we feel that these several strategies, especially ordered semantic hyper-linking [Pla94a], have the most long-term promise for a generally useful theorem prover.

2.12 Conclusions and future work

We see that much can be said about the complexities of various strategies on various kinds of input clauses. This can help to give insight into their performance in practice. However, this analysis does not take into account factors such as careful programming technique or the unusual distributions of problems that may arise in practice. Also, it does not consider all aspects of a strategy, such as the fact that model elimination is a set-of-support strategy, or the benefits to be obtained from caching. The classification into exponential, double exponential, et cetera, is also coarse and could be refined, yielding additional insights. The assumption that Davis and Putnam's method runs in expected polynomial time is also open to question, although it seems plausible based on experience, and has been verified analytically for a number of reasonable probability distributions on formulas. One might also argue that we are estimating the average case performance of methods using a propositional satisfiability test, but the worst-case time of methods that do not. This places these other methods at an unfair disadvantage. However, Davis and Putnam's method often stops early when it detects satisfiability. Inference-based methods such as resolution and model elimination have to exhaust the search space in order to detect satisfiability, and this usually does not happen early. Therefore, there is some reason to believe that inference-based methods will have an average-case performance that is not much better than their worst-case performance. There also seems to be no reason why the matings method should perform better on the average than in the worst case. Another issue related to the complexity of theorem provers is that truly powerful provers may need to use semantics, specialized rules of inference that were not considered here, and even human guidance. It is also possible that a more careful analysis may bring out other advantages and disadvantages of strategies (and their refinements) that are not obvious here. We encourage others to fill in the gaps we have left. In addition, this analysis can be extended to strategies with special rules for the equality predicate; for a start in this direction, see [PSK95]. Still, we believe that we have made a significant contribution. Especially as machines become faster and faster, asymptotic performance will play a larger and larger role, and constant factors of speed obtained by good programming technique or choice of programming language will become relatively less important. We hope that insights gained from this work will help to direct implementors into fruitful methods and combinations of methods. In addition, the many specific example clause sets given above to exhibit worst-case behavior of strategies may be useful in their own right as tests of the efficiencies of various strategies.

Bibliography

[And81] P. B. Andrews. Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28:193–214, 1981.

[AS92] Owen Astrachan and M. Stickel. Caching and lemma use in model elimination theorem provers. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, 1992.

[BE93] W. Bibel and E. Eder. Methods and calculi for deduction. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, chapter 3, pages 71–193. Oxford University Press, Oxford, 1993.

[BF93] Peter Baumgartner and Ulrich Furbach. Consolution as a Framework for Comparing Calculi. *Journal of Symbolic Computation*, 16(5):445–477, 1993.

[BFL94] M. Baaz, C. Fermüller, and A. Leitsch. A Non-Elementary Speed Up in Proof Length by Structural Clause Form Transformation. In *LICS'94*, pages 213–219, Los Alamitos, California, 1994. IEEE Computer Society Press.

[BG90] Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. In Mark Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, pages 427–441, New York, 1990. Springer-Verlag.

[BH96] M. Bonacina and J. Hsiang. On the modelling of search in theorem proving - towards a theory of strategy analysis. 1996. Unpublished.

[BH98] Maria Paola Bonacina and Jieh Hsiang. On the modelling of search in theorem proving – towards a theory of strategy analysis. *Information and Computation*, 147:171–208, 1998.

[Bib82] W. Bibel. A comparative study of several proof procedures. *Artificial Intelligence*, 12:269–293, 1982.

[Bib87] W. Bibel. *Automated Theorem Proving*. Vieweg, Braunschweig/Wiesbaden, 1987. Second edition.

[Bon] Maria Paola Bonacina. A model and a first analysis of distributed-search contraction-based strategies. *Annals of Mathematics and Artificial Intelligence*, to appear.

[BT88] S. R. Buss and G. Turán. Resolution proofs of generalized pigeonhole principles. *Theoretical Computer Science*, 62(3):311–317, 1988.

[Bun83] A. Bundy. *The Computer Modelling of Mathematical Reasoning*. Academic Press, New York, 1983.

[CL73] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973.

[CP94] Ritu Chadha and D. Plaisted. Correctness of unification without occur check in prolog. *Journal of Logic Programming*, 18:2:99–122, 1994.

[CR79] S. A. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, March 1979.

[Dav63] M. Davis. Eliminating the irrelevant from machanical proofs. In *Proceedings Symp. of Applied Math*, volume 15, pages 15–30, 1963.

[DG84] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[Ede92] E. Eder. *Relative Complexities of First-Order Calculi.* Vieweg, Braunschweig, 1992.

[Egl96] U. Egly. On Different Structure-preserving Translations to Normal Form. *Journal of Symbolic Computation*, 22:121–142, 1996.

[Fit90] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, 1990.

[Gou94] Jean Goubault. The complexity of resource-bounded first-order classical logic. In P. Enjalbert, E.W. Mayr, and K.W. Wagner, editors, *11th Symposium on Theoretical Aspects of Computer Science*, pages 59–70, Caen, France, February 1994. Springer Verlag LNCS 775.

[GU89] G. Gallo and Y. Urbani. Algorithms for testing the satisfiability of propositional formulae. *Journal of Logic Programming*, 7:45–61, 1989.

[Hak85] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[HR91] J. Hsiang and M Rusinowitch. Proving refutational completeness of theorem-proving strategies: the transfinite semantic tree method. *J. Assoc. Comput. Mach.*, 38(3):559–587, July 1991.

[KBL93] H. Kleine Buening and T. Lettman. Search space and average proof length of resolution. Unpublished, 1993.

[KN92] D. Kapur and P. Narendran. Double-exponential complexity of computing a complete set of AC-unifiers. In *Proceedings 7th IEEE Symposium on Logic in Computer Science*, pages 11–21, Santa Cruz, California, 1992.

[Kor85] R. E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[Let93] R. Letz. On the polynomial transparency of resolution. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 123–129, 1993.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. 2nd edn.

[Lov69] D. Loveland. A simplified format for the model elimination procedure. *J. ACM*, 16:349–363, 1969.

[Lov78] D. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, New York, 1978.

[LP92] S.-J. Lee and D. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9(1):25–42, 1992.

[Lyn94] Christopher Lynch. Local simplification. In *Constraints in Computational Logics*, Munich, Germany, September 1994.

[McC90] W. McCune. Otter 2.0 (theorem prover). In M.E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, pages 663–4, July 1990.

[PA96] D. Plaisted and G. Alexander. Propositional search efficiency and first-order theorem proving. In Ding-Zhu Du, Jun Gu, and Panos Pardalos, editors, *Proceedings of the Workshop on SATISFIABILITY PROBLEM: THEORY AND APPLICATIONS*, March 12-13 1996.

[Pla82] D. Plaisted. A simplified problem reduction format. *Artificial Intelligence*, 18:227–261, 1982.

[Pla84] D. Plaisted. Complete problems in the first-order predicate calculus. *Journal of Computer and System Sciences*, 29(1):8–35, 1984.

[Pla88] D. Plaisted. Non-Horn clause logic programming without contrapositives. *Journal of Automated Reasoning*, 4:287–325, 1988.

[Pla90] D. Plaisted. Mechanical theorem proving. In R. Banerji, editor, *A Sourcebook on Formal Techniques in Artificial Intelligence.* Elsevier, Amsterdam, 1990.

[Pla94a] D. Plaisted. Ordered semantic hyper-linking. Technical Report MPI-I-94-235, Max-Planck Institut fuer Informatik, Saarbruecken, Germany, 1994.

[Pla94b] D. Plaisted. The search efficiency of theorem proving strategies. In *Proceedings of the Twelfth International Conference on Automated Deduction*, pages 57–71, 1994. Lecture Notes in Artificial Intelligence 814.

[Pla94c] D. Plaisted. The search efficiency of theorem proving strategies: an analytical comparison. Technical Report MPI-I-94-233, Max-Planck Institut fuer Informatik, Saarbruecken, Germany, 1994.

[Pla94d] D. Plaisted. The search space size for a class of resolution strategies. In *Workshop der GI-Fachgruppe Logik in der Informatik*, Paderborn, Germany, May 1994. organized by Prof. H. Kleine Buening.

[PSK95] D. Plaisted and Andrea Sattler-Klein. Proof lengths for equational completion. Technical Report SEKI Report SR-95-06, University of Kaiserslautern, Kaiserslautern, Germany, 1995.

[PZ] M. Paramasivam and Y. Zhu. Personal communication.

[Rob65a] J. Robinson. Automatic deduction with hyper-resolution. *Int. J. Comput. Math.*, 1:227–234, 1965.

[Rob65b] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[Sla67] J.R. Slagle. Automatic theorem proving with renameable and semantic resolution. *J. ACM*, 14:687–697, 1967.

[ST85] M.E. Stickel and W.M. Tyson. An analysis of consecutively bounded depth-first search with applications in automated deduction. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 1073–1075, 1985.

[Tam90] T. Tammet. The resolution program: able to decide some solvable classes. In *International Conference on Computer Logic, 1988*, pages 300–312, 1990. Springer Verlag LNCS 417.

[Tam91] T. Tammet. Using resolution for deciding solvable classes and building finite models. In *Baltic Computer Science*, pages 33–64, 1991. Springer Verlag LNCS 502.

[Tse68] G.S. Tseitin. On the complexity of derivation in propositional calculus. In A.O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 234–259. V.A. Steklov Mathematical Institute, Leningrad, 1968. English translation: Consultants Bureau, New York, 1970, pp. 115–125.

[Urq87] A. Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.

[WOLB84] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications.* Prentice Hall, Englewood Cliffs, N.J., 1984.

[WRC65] L. Wos, G. Robinson, and D. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the Association for Computing Machinery*, 12:536–541, 1965.

[Zam72] N.K. Zamov. On a bound for the complexity of terms in the resolution method. *Trudy. Mat. Inst. Steklov*, 128:5–13, 1972.

[Zam89] N.K. Zamov. Maslov's inverse method and decidable classes. *Annals of pure and applied logic*, 42:165–194, 1989.

[ZK88] H. Zhang and D. Kapur. First order theorem proving using conditional rewrite rules. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction*, pages 1–20. Springer-Verlag, 1988.

Index