MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

The Search Efficiency of Theorem Proving Strategies: An Analytical Comparison

David A. Plaisted

MPI–I–94–233 July 1994

INFORMATIK

Im Stadtwald
D 66123 Saarbrücken
Germany

Authors' Addresses

David Plaisted
Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
Germany
`plaisted@mpi-sb.mpg.de`

or

Department of Computer Science
CB# 3175, 352 Sitterson Hall
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina 27599-3175
USA
`plaisted@cs.unc.edu`

Publication Notes

A short version of this paper appeared at the Twelfth Conference on Automated Deduction in Nancy, France, June - July, 1994.

Acknowledgements

This work was done while the 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.

Abstract

We analyze the search efficiency of a number of common refutational theorem proving strategies for first-order logic. 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.

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 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.

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.

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.

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.

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 behavoirs. 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].

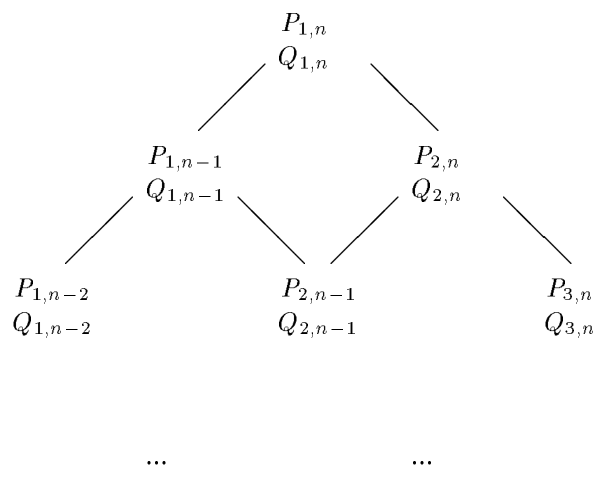
5.1 Hard sets of clauses for the strategies

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:

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

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

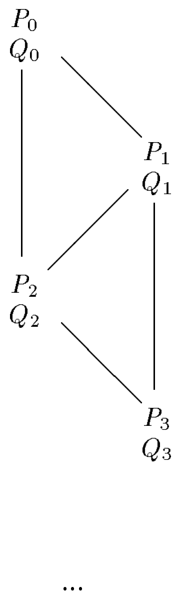


We can think of backward chaining theorem proving strategies on this set of clauses as ways of moving P-pebbles and Q-pebbles around on this graph. Initially, there is a P-pebble on the $(1, n)$ vertex. At each step, we are permitted to remove a pebble. If we remove a P pebble from vertex (i, j), we 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:

$$
\begin{array}{rrll}
\textit{goal clause} & & :-\ P_0, Q_0 & \\
\textit{type 1 clauses} & P_i & :-\ P_{i+1}, P_{i+2}, & 0 \le i < 2n-2 \\
 & P_i & :-\ Q_{i+1}, Q_{i+2}, & 0 \le i < 2n-2 \\
 & Q_i & :-\ P_{i+1}, Q_{i+2}, & 0 \le i < 2n-2 \\
 & Q_i & :-\ Q_{i+1}, P_{i+2}, & 0 \le i < 2n-2 \\
\textit{type 2 clauses} & P_{2n-1} & :-\ P_{n-1} & \\
 & P_{2n} & :-\ P_n & \\
 & Q_{2n-1} & :-\ Q_{n-1} & \\
 & Q_{2n} & :-\ Q_n &
\end{array}
$$

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 [Rob65].

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.

5.3 P_1 deduction

P_1-deduction is the strategy that resolves two clauses only if one of them is positive [Rob65]. 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.

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 5.10.

It is conceptially 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 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 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.

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. (And 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.

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 $P_1 \wedge P_2 \wedge ... \wedge P_n \supset P$. 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 beween 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 $Q_1, ..., Q_n$ are the vertices of $D(S)$ with no outgoing edges, then these are goal vertices and there must be a goal clause $\{\neg Q_1, ..., \neg Q_n\}$ 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.

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 $P \ :- \ P_1, ..., P_n$ 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.

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 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 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.*

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

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

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 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 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 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.

□

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 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 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.

□

This result was first presented in [Pla94]. 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 don't 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 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.

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 5.1 that the search space for breadth-first A-ordering is also polynomial, regardless of the ordering.

This is a good 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 constrast 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 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 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* 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 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 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 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 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 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 5.16 for individual clauses.

Lemma 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 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 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 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 anologue of theorem 5.10 for A-ordering resolution, too, even with a good ordering.

□

We now continue a line of investigation begun earlier in section 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.

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.

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 unsatisfiable. 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.

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.

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.

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.

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.

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]. 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.

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.

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.

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.

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-deduction and 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.

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. However, 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.

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, $Sym(S_n^2)$, $Sym(S_n^3)$, $U(T_n^2)$, $U(T_n^3)$, $M(T_n^2)$, $M(T_n^3)$, $Sym(U(T_n^2))$, $Sym(U(T_n^3))$, $Sym(M(T_n^2))$, $Sym(M(T_n^3))$, $N(T_2^n, n)$, and $N(T_3^n, n)$.

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.

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's 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.

References

[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.

[BG90] Leo Bachmair and Harold 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.

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

[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] Heng Chu and D. Plaisted. Semantically guided first-order theorem proving using hyper-linking. In *Proceedings of the Twelfth International Conference on Automated Deduction*, pages 57–71, 1994. Lecture Notes in Artificial Intelligence 814.

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

[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.

[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.

[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.

[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.

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

[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.

[Pla94] 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.

[Rob65] J. Robinson. Automatic deduction with hyper-resolution. *Int. J. Comput. Math.*, 1:227–234, 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.