MAX-PLANCK-INSTITUT FÜR INFORMATIK

Ordered Semantic Hyper-Linking

David A. Plaisted

MPI–I–94–235 August 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

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 propose a method for combining the clause linking theorem proving method with theorem proving methods based on orderings. This may be useful for incorporating term-rewriting based approaches into clause linking. In this way, some of the propositional inefficiencies of ordering-based approaches may be overcome, while at the same time incorporating the advantages of ordering methods into clause linking. The combination also provides a natural way to combine resolution on non-ground clauses, with the clause linking method, which is essentially a ground method. We describe the method, prove completeness, and show that the enumeration part of clause linking with semantics can be reduced to polynomial time in certain cases. We analyze the complexity of the proposed method, and also give some plausibility arguments concerning its expected performance.

1 Introduction

There are at least two basic approaches to the study of automated deduction. One approach concentrates on solving hard problems, especially those of interest to human mathematicians. There have been some notable successes in this area, and even some proofs of hitherto unproven conjectures. This has served to give the field of automated deduction some respectability among mathematicians and the general public. Such proofs may be done with or without human interaction. For this approach, it is of secondary interest whether the prover that solves these hard problems is very weak for other, easy problems. Another approach to automated deduction concentrates on building provers that perform well on a broad range of problems, with a minimum of human interaction. In this approach, it does not make sense to try hard problems if one's prover still has difficulty with easy problems. The philosophy is to develop as far as possible theorem provers that have general problem solving ability. The advantage of this approach is that one is likely to obtain provers that are more well-rounded, and in the long run possibly more powerful; a disadvantage is that the results may appear less spectacular, especially in the early stages.

We have concentrated on the latter approach. In this endeavor, we have developed a number of provers over the recent years, including the modified problem reduction format of [Pla88] and its extensions, the clause linking method of [LP92], and clause linking with semantics [CP94a], among others. We have emphasized first-order logic without equality. These provers have become increasingly more powerful, each able to solve a considerable range of problems out of reach of its predecessor, with little or no human guidance. In addition, these provers can compete respectably with well-known powerful theorem provers on certain types of problems. For this, we have emphasized Prolog implementation, as a way to rapidly implement and test a wide variety of ideas with little manpower. This means that our provers have a disadvantage with respect to provers carefully implemented in C or LISP, since the underlying language is less efficient; despite this, the performance has been impressive. We now would like to carry this investigation a step further, and incorporate some kind of ordering methods.

Semantic hyper-linking [CP94a] was developed to retain the propositional advantages of hyper-linking [LP92] while adding natural semantics and goal-sensitivity [Pla94a]. We discuss the issues of propositional efficiency and goal-sensitivity in [Pla94a, Pla94b], thereby highlighting what we feel are some inefficiencies in many common theorem proving strategies. Even hyper-linking without semantics performs much better than resolution and similar strategies on some hard problems, particularly non-Horn problems. We show in [CP94a] that hyper-linking with semantics is sometimes much better still. However, there are still some problem with semantic hyper-linking that we would like to solve. The cooperation between semantic hyper-linking and rough resolution does not seem as clean as it could be. Rough resolution [CP93b] is a version of ordered resolution developed to eliminate large literals from proofs; this is helpful because semantic hyper-linking has difficulties generating such large literals. Therefore the cooperation of these two methods seems attractive, and indeed, we show in [CP93b] that this cooperation improves the effectiveness of semantic hyper-linking on a number of problems. However, the definition of rough resolution seems arbitrary; the logical way to eliminate large literals is to use ordered resolution, as described in [BG90, HR91]. Also, the manner in which the semantic tree is constructed and searched seems to have an arbitrary element to it; this also makes this part of the method harder to describe formally. In addition to semantic hyper-linking and rough resolution, UR (unit resulting) resolution is a component of the prover described in [CP94a]. The motivation for this is that rough resolution eliminates large literals from proofs, UR resolution eliminates the Horn parts of proofs, and what remains is a non-Horn proof with small literals; clause linking performs well on such problems. However, the Horn property is really irrelevant here, since clause linking performs well on problems with small literals, whether they are Horn problems or not. Therefore it seems logical to eliminate UR resolution. Also, the choice of which rough resolutions and UR-resolutions are performed seems to be arbitrary, to some extent; we prefer small clauses and small proofs, essentially. Our motivation in the development of ordered semantic hyper-linking is to simplify semantic hyper-linking as much as possible, and in this way hopefully to extend its peaks of strength to a wider class of problems. This should also allow for a small and easily understood implementation.

In addition, we are interested in removing some of the propositional inefficiencies from term-rewriting based theorem proving strategies. Such strategies essentially reduce to A-ordering [Sla67] on first-order logic without equality. However, in [Pla94b] we show some simple sets of clauses on which A-ordering produces an exponential search space, regardless of the ordering chosen. On the other hand, term-rewriting methods are often very efficient, and extend naturally to other specialized theories [BG94]. Furthermore, certain sets of clauses are easily decided by strategies based on ordering [FLTZ93]. That is, these ordering based strategies are a decision procedure, always terminating and indicating whether the set of input clauses is satisfiable or not. However, these sublanguages of first-order logic are not decidable in this way by clause linking. Therefore, we would like to present semantic clause linking in a format that facilitates the transition to term-rewriting based theorem proving strategies, thereby also providing a way to remove some of their notable inefficiencies, while preserving some of their advantages.

The idea of semantic hyper-linking is to show that a set S of clauses is unsatisfiable by the failure of a systematic search for a model of S. Ordered semantic hyper-linking is similar, but it organizes the search a little differently. The basic principle behind ordered semantic hyper-linking is the following: Suppose we have a set $p_1, p_2, \ldots, p_n$ of independent choices to make in a process of examining a set of possibilities. Thus there are potentially 2^n combinations of choices altogether. Then we assume that there is an ordering on these choices, and we make the simplest choice first. Suppose p_1 is the simplest of the choices; then we first consider the alternatives p_1 and $\mathrm{not}(p_1)$. For each such alternative, we recursively attempt to solve the problem. The reasoning is that we may solve the problem before the more complex choices are even seen, thereby saving effort. This seems to be a natural strategy from the standpoint of human problem solving. In the application to theorem proving, the set of choices is infinite, and the order in which they are made has other implications, but the idea is still the same.

Furthermore, a problem with semantic hyper-linking is that sometimes the enumeration of ground terms is necessary. We would like to have a method that is based on unification instead of on the enumeration of ground terms. The proposed method incorporates unification in a natural way, and for certain kinds of semantics we show that the enumeration phase can be done in polynomial time. There is an additional reason to believe that this new version will have better performance. The work required by semantic hyper-linking is strongly influenced by the number of "eligible literals" that are generated. The proposed method should reduce this number, thereby making the method more efficient and permitting proofs that require more rounds of search.

We also consider some complexity issues, as discussed in part by [Gou94]. His approach is non-clausal, but the same analysis applies to a clausal framework. He shows that the fundamental problem associated with the mating [And81] or connection [Bib87] approach is Σ_2^p complete. These approaches first choose a number of copies of each of the input clauses, and then seek to find a single substitution that makes the given set of copies of the clauses propositionally unsatisfiable. (For a general set of clauses, an arbitrary number of copies may be needed.) Goubault's result has the consequence that the effort to prove a theorem is at worst exponential in the size of a minimal proof, using a suitable implementation of these approaches, where proof size is measured by the number of instances of the input clauses that appear in the proof. More precisely, if S is the set of input clauses and each clause appears at most n times in the proof, then the effort is exponential in n times the size of S, written as a character string. This is actually not a bad bound; many other methods are considerably worse, at least relative to this measure of proof size. For example, clause linking can be double or even triple exponential in this measure of proof complexity. The reason is that a proof containing n copies of the input clauses may involve a number of unifications proportional to n; each such unification can increase the size of the terms by a constant factor. Thus terms that are exponentially large can be generated, the number of terms within this exponential size bound is double exponential, and the time to handle a double exponential set of ground clauses can be triple exponential. By altering the measure of term size, so that repeated subterms are counted only once, this can be reduced to double exponential. We consider the behavior of ordered clause linking and argue that although the worst-case bound is double exponential, there is reason to believe that in many cases this performance will be single exponential or even better. Of course, there may be proofs in which many copies of the clauses are needed but the term sizes are all small; for such clauses, clause linking would probably be faster than the mating approach.

It is interesting that our proposed theorem prover incorporates a number of well-known AI techniques, including case analysis, explanation-based generalization, procedural semantics (to describe the semantics of the set of clauses), backtracking, ordering criteria, and of course unification and first-order logic. Therefore this prover may have some independent interest from the standpoint of artificial intelligence.

2 Orderings on Interpretations and Clauses

The idea of ordered clause linking with semantics is to place a total ordering on the set of atoms, and then based on this to define a lexicographic total ordering on the set of interpretations of these atoms. Now, semantic hyper-linking can be seen as a systematic search for a model of a set S of clauses; if this search fails, then we know that S is unsatisfiable. During this search, a semantic tree is essentially constructed. Ordered semantic hyper-linking works in a similar way. In fact, both strategies are somewhat similar to model elimination in this respect [Lov69]; this similarity may be more apparent for ordered semantic hyper-linking than for semantic hyper-linking. However, instead of a semantic tree, we have a transfinite semantic tree, as in [HR91]. Also, in ordered semantic hyper-linking we specify more precisely how this tree is constructed. That is, the interpretations are examined in a sequence $I_0, I_1, I_2 \ldots$ consistent with the total ordering; the first interpretation I_0 to be considered is the one that is least in this ordering. For this interpretation we find a clause C_0 not satisfied by I_0; this clause is a minimal such contradicting clause in a specified ordering on clauses. The next interpretation I_1 considered is the smallest interpretation that is not "obviously" contradicted by the clauses found so far (in this case, C_0). The search continues in this manner, so we have $I_0 < I_1 < I_2 \ldots$ and $C_0, C_1, C_2, \ldots$. Not only is the sequence in which the interpretations are examined completely determined in this way, but the choice of which clauses C_i are found is also largely determined (up to minimality). If S is unsatisfiable, eventually the set $\{C_0, C_1, C_2, \ldots, C_i\}$ will be unsatisfiable; this will be detected by the prover, and the search will stop.

We now define these orderings on atoms and interpretations more precisely. We assume that some first-order language is specified in terms of a finite set $\mathcal{F}$ of function and constant symbols, a finite set $\mathcal{P}$ of predicate symbols, and a list $\mathcal{X}$ of variables. Then we are interested in the satisfiability problem of sets S of first-order clauses over this language. Let $\mathcal{T}(\mathcal{F}, \mathcal{X})$ be the set of terms formed from the function symbols $\mathcal{F}$ and the variables $\mathcal{X}$, and let $\mathcal{T}(\mathcal{F})$ be the set of ground terms (terms without variables) formed from $\mathcal{T}$ and $\mathcal{F}$. Let $\mathcal{A}$ be the set of *atoms*, that is, expressions of the form $P(t_1, \ldots, t_n)$, where $P \in \mathcal{P}$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. For orderings $<$, we define $x > y$ as equivalent to $y < x$. A partial ordering $<$ is said to be *well-founded* if there are no infinite sequences $x_1, x_2, x_3, \ldots$ with $x_1 > x_2 > x_3 \ldots$. We assume that there is a total well-founded ordering $<$ on $\mathcal{A}$. If the ordering is order-isomorphic to ω, then $\mathcal{A}$ can be enumerated as $A_1, A_2, A_3, \ldots$ where $A_1 < A_2 < A_3 \ldots$; this will not be possible if the ordering corresponds to a higher ordinal. The ordering on $\mathcal{A}$ may be based on the size (number of symbol occurrences) in the atoms A_i, or it may be one of the orderings used to show the termination of term-rewriting systems [DJ90, Pla93]. Of course, there are also other possibilities.

We now specify the ordering on interpretations. A *literal* is an atom or an atom preceded by a negation sign $\neg$. A literal without a negation sign is called *positive* and one with it is called *negative*. The literals L and $\neg L$ are called *complementary*. We write $\overline{L}$ for the complement of L; thus if L is positive then $\overline{L}$ is $\neg L$, and $\overline{\neg L}$ is L. If $A \in \mathcal{A}$, then we call A and $\neg A$ literals *over* $\mathcal{A}$. An *interpretation* I is (for our purposes) a subset of $\mathcal{A}$, or, equivalently, a mapping from $\mathcal{A}$ to $\{true, false\}$. If I maps A to $true$ then we write $I \models A$ and say that I satisfies A. Otherwise, $I \not\models A$ and we say that A contradicts I. We say that $I \models \neg L$ iff $I \not\models L$. We say that two interpretations I and J *agree on a literal* L if ($I \models L$ iff $J \models L$); that is, they both assign L $true$ or they both assign L $false$. If I and J are two distinct interpretations of $\mathcal{A}$, let $d(I, J)$ be the least atom A (with respect to $<$) such that I and J do not agree on A. Such a least atom must exist, because the ordering on atoms is well-founded. Let I_0 be a special interpretation called the *initial interpretation*; this is typically supplied by the user. We order the interpretations with respect to I_0 as follows: Let I and J be two different interpretations of $\mathcal{A}$. Let A be $d(I, J)$. Then if I agrees with I_0 on A, we say $I < J$, otherwise $J < I$. Thus the smallest literals have the greatest influence on the ordering, and the interpretations that agree with I_0 are smaller in the ordering than interpretations that disagree with I_0, other things being equal. Note that I_0 is the minimal interpretation in this ordering. This ordering on interpretations is not well-founded, if $\mathcal{A}$ is infinite. Still, it turns out that certain sets of interpretations of interest to us have minimal elements. We will mainly be interested in interpretations that differ from I_0 in finitely many places; we now develop some of their properties.

Definition 2.1 *Given an interpretation I of $\mathcal{A}$ and literals L_i over $\mathcal{A}$, let $I[L_1, L_2, \ldots, L_n]$ be defined as follows:*

$$I[L_1, L_2, \ldots, L_n] \models L \text{ iff } \begin{array}{ll} (a) & L \in \{L_1, \ldots, L_n\} \text{ or if} \\ (b) & L \notin \{L_1, \ldots, L_n\} \text{ and } \overline{L} \notin \{L_1, \ldots, L_n\} \text{ and } I \models L \end{array}$$

Thus $I[L_1, L_2, \ldots, L_n]$ is like I except for the finite list $[L_1, L_2, \ldots, L_n]$ of "exceptions;" for an earlier use of this notation see [CP94a].

Theorem 2.2 *Suppose I is an interpretation over $\mathcal{A}$ and $J = I_0[L_1, L_2, \ldots, L_n]$. Then if $I < J$, $d(I, J) \in \{L_1, L_2, \ldots, L_n\}$.*

Proof. If $I < J$, then I and J are unequal; thus there must be a literal $d(I, J)$. If $d(I, J) \notin \{L_1, L_2, \ldots, L_n\}$, then J agrees with I_0 on the atom $d(I, J)$, so $J < I$. $\square$

This result classifies the interpretations that are smaller than $I_0[L_1, L_2, \ldots, L_n]$ into n distinct groups, depending on which L_i is equal to $d(I, J)$.

We now specify the ordering on literals and clauses. We order literals so that if A and B are atoms and $A < B$ then $A < B$ (as literals), $A < \neg B$, $\neg A < B$, and $\neg A < \neg B$. However, the literals A and $\neg A$ are not ordered with respect to one another. A *clause* is a finite set of literals, regarded as their disjunction. A clause is a *tautology* if it contains a literal and its negation. A clause is a *ground clause* if it contains no variables (and similarly for literals, atoms, and terms). We define a partial ordering $<$ on non-tautologous clauses by the multiset extension of the ordering on literals: If C is the empty clause (which contains no literals), then $C < D$ for every non-empty clause D. Also, if C and D are clauses, then let L and M be their maximum respective literals, which exist if C and D are non-tautologous. Then if $L > M$, $C > D$, and if $L < M$, $C < D$, and if $L = M$, then $C > D$ if $C - \{L\} > D - \{M\}$ and $C < D$ if $C - \{L\} < D - \{M\}$. If L and M are complementary, then C and D are not ordered. We note that this multiset ordering on clauses is well-founded, because the underlying ordering on literals is. We will not further use this ordering on clauses, and present it here mainly to make this point clear. An interpretation I satisfies a ground clause C if I satisfies some (that is, at least one) literal in C, and in this case we write $I \models C$. Otherwise, we say that C contradicts I and write $I \not\models C$. A *substitution* is a mapping from variables to terms. If Θ is a substitution and C is a clause then $C\Theta$ represents the clause obtained by replacing variables of C by terms, as specified by Θ. Such a clause $C\Theta$ is called an *instance* of C. A similar terminology applies to terms, atoms, and literals, so we can apply substitutions to them, for example. For our purposes, we say that an interpretation satisfies a non-ground clause if it satisfies all of its ground instances. An interpretation I satisfies a set S of clauses if it satisfies every clause C in S. An interpretation I is a *model* of a ground clause C if $I \models C$, and I is a model of a set S of ground clauses if $I \models C$ for all $C \in S$. Least models of single ground clauses and finite sets of ground clauses exist, as we now show.

Theorem 2.3 *Suppose $C = \{L_1, L_2, \ldots, L_n\}$ is a ground clause, and suppose $L_1 < L_2 < \ldots < L_n$. Suppose $I_0 \not\models C$. Then the least model of C is $I_0[L_n]$.*

Proof. We note that $I_0[L_n]$ is a model of C. We show that no smaller interpretation J is a model of C. Suppose $J < I_0[L_n]$. Then by theorem 2.2, $d(I, J) = L_n$. This implies that J agrees with I_0 on L_n, and so C contradicts J. □

Theorem 2.4 *Let G be a finite set of ground clauses. If G is satisfiable, then G has a least model.*

Proof. Let $A_1, \ldots, A_n$ be the atoms that appear (positively or negatively) in clauses in G. There are only finitely many interpretations of these atoms; at least one of them, say I, is a model, since G is satisfiable. Let I_{min} be the least such model, in our ordering on interpretations. This must exist, since there are only finitely many such finite interpretations. Extend I_{min} to an interpretation $I_0[L_1, L_2, \ldots, L_n]$ of $\mathcal{A}$ where each L_i is either A_i or $\neg A_i$ and $I_{min} \models L_i$ for all i. Then J is a model of G, since I_{min} is, and it is a least model, because I_{min} is as small as possible among the interpretations of the A_i and the interpretations of the other literals have been chosen as small as possible. Any smaller interpretation would have to differ from J on some literal L_i, by theorem 2.2, which is not possible by the way I_{min} was chosen. □

Definition 2.5 *If G is a set of ground clauses over $\mathcal{A}$, let $lm(G)$ be its least model, that is, the smallest interpretation I in the ordering on interpretations such that I satisfies all elements of G.*

In ordered clause linking with semantics, we accumulate such a set G and in the process keep track of its least model $lm(G)$. However, we do not store G in its original form, but apply certain simplifications to it so that only the features relevant for the current minimal model are retained. For this, we not only make use of least models but also *least contradicting clauses* of interpretations. However, this necessitates the introduction of another ordering on clauses. We would have liked to define ordered semantic hyper-linking entirely in terms of the ordering $<$ on clauses, but it turns out that this is incomplete if the ordering $<$ has order type greater than ω. Therefore we introduce another ordering $<_{cl}$ on clauses and use it to choose contradicting instances. This means that for purposes of completeness, ordered semantic hyper-linking may need to use two different orderings, which seems to be somewhat remarkable. For this purpose, we assume that $<_{cl}$ is a partial ordering such that for all ground clauses C, all but finitely many ground clauses D satisfy $C <_{cl} D$. An example of such an ordering is to order the clauses by size, that is, the total number of occurrences of symbols, with two clauses unordered if they have the same number of symbols. Since there are only finitely many non-variable symbols that can appear in a clause, this ordering satisfies the finiteness property given above. We note that $<_{cl}$ is well-founded. We say a clause C is *minimal* in a set of clauses if there is no clause D in the set such that $D <_{cl} C$; note that a set of clauses can have more than one, but at most finitely many, minimal elements.

We now show that minimal contradicting clauses always exist; that is, if I is not a model of a set S of (non-ground) clauses, then there is a minimal ground instance D of a clause C of S such that D contradicts I.

Theorem 2.6 *Suppose S is a set of (possibly) non-ground clauses over $\mathcal{A}$ and I is an interpretation of $\mathcal{A}$ that is not a model of S. Then there is a ground instance D of some clause C of S such that for all other ground instances D' of clauses in S, if $I \not\models D'$ then either $D <_{cl} D'$ or D and D' are unrelated by $<_{cl}$.*

Proof. The set of such ground clauses D' such that $I \not\models D'$ is non-empty, since $I \not\models S$. Since the ordering $<_{cl}$ on ground clauses is well-founded, this set of ground clauses has a minimal element D, as claimed. Note that there may be more than one such minimal D, but at most only finitely many, by the way $<_{cl}$ is defined. This result holds even if S is infinite, by the way. □

Definition 2.7 *Let $mi(S, I)$ be some such clause D, that is, a ground instance D of some clause in S such that $I \not\models D$ and such that there is no other ground instance C of a clause in S such that $C <_{cl} D$ and such that $I \not\models C$. If there is more than one such instance D, we assume that $mi(S, I)$ is one of them, chosen in an arbitrary manner. We call such a clause D a* minimal contradicting instance *for I.*

Later we will discuss algorithmic aspects of computing $mi(S, I)$.

3 The Search Procedure

The task now is to devise a procedure that will search through the set of all interpretations in a manner consistent with the ordering, finding clauses that contradict each interpretation. For this, we maintain a finite list $\mathcal{C}$ of "relevant" ground clauses that record the progress made in the search so far; this list contains instances of clauses from S as well as instances of clauses derived from S by A-ordering resolution. The goal of the search is to continually increase the least model $lm(\mathcal{C})$ of $\mathcal{C}$, that is, to make $lm(\mathcal{C})$ larger and larger in the ordering on interpretations. This least model is called the *current interpretation*. At the beginning $\mathcal{C}$ is empty and $lm(\mathcal{C})$ is I_0. As elements are added to $\mathcal{C}$, and sometimes removed, this least model becomes larger and larger in our ordering on interpretations. If S is unsatisfiable, then eventually $\mathcal{C}$ will contradict all interpretations, and there will not be a least model any more. At this point, the empty clause will be in $\mathcal{C}$ and the search will stop. The invariant that $\mathcal{C}$ possesses is captured by the following definitions.

Definition 3.1 *If C is a (non-tautologous) clause, let $max(C)$ be the maximal literal in C in the ordering $<$. This exists because clauses are finite.*

Definition 3.2 *A list $\mathcal{C}$ of clauses is* ascending *if it is of the form $C_1, C_2, \ldots, C_n$ where $max(C_1) < max(C_2) < \ldots$ and where for all $i \leq n$, $lm(\{C_1, C_2, \ldots, C_i\}) \not\models C_{i+1}$. Thus C_1 contradicts I_0, C_2 contradicts the least model of C_1, and so on. The literals $max(C_i)$ are called* eligible literals*, in harmony with the use of this term in [CP94a].*

Theorem 3.3 *If $\mathcal{C} = C_1, C_2, \ldots, C_n$ is ascending, then so are all its prefixes. Also, if $C_1, C_2, \ldots, C_n$ is ascending, then $I_0 \not\models max(C_i)$ for all i and $lm(\{C_1, C_2, \ldots, C_n\})$ $= I_0[max(C_1), max(C_2), \ldots, max(C_n)]$. Furthermore, this latter interpretation disagrees with I_0 on the literals $max(C_j)$.*

Proof. The part about prefixes is immediate. Let I be $I_0[max(C_1), max(C_2), \ldots,$ $max(C_n)]$. For the rest, we use induction on i. We show that $I_0 \not\models max(C_i)$ for all i. By induction, $lm(\{C_1, C_2, \ldots, C_{i-1}\}) = I_0[max(C_1), max(C_2), \ldots, max(C_{i-1})]$. Since the literal $max(C_i)$ is larger than any literal $[max(C_1), max(C_2), \ldots, max$ $(C_{i-1})]$, $lm(\{C_1, C_2, \ldots, C_{i-1}\})$ agrees with I_0 on $max(C_i)$. By the definition of ascending, $lm(\{C_1, C_2, \ldots, C_{i-1}\}) \not\models C_i$, therefore $lm(\{C_1, C_2, \ldots, C_{i-1}\}) \not\models$ $max(C_i)$. Since I_0 agrees with $lm(\{C_1, C_2, \ldots, C_{i-1}\})$ on $max(C_i)$, $I_0 \not\models max(C_i)$.

To show that I and I_0 disagree on the literals $max(C_i)$, we have just shown that $I_0 \not\models max(C_i)$ for all i, but $I \models max(C_i)$ by the way it is constructed.

To show that I is $lm(\mathcal{C})$, we show that I is a model of $\mathcal{C}$ but no smaller interpretation J is a model of $\mathcal{C}$. Now, I is a model of $\mathcal{C}$ since it satisfies all the literals $max(C_i)$. If $J < I$ then J must differ from I on some eligible literal, by theorem 2.2. Suppose $d(I, J)$ is $max(C_i)$. Then $J \not\models max(C_i)$ (since $I \models max(C_i)$ and I and J differ on $max(C_i)$). It remains to show that J does not satisfy the other literals of C_i. We know that $lm(\{C_1, C_2, \ldots, C_{i-1}\}) \not\models C_i$, by the definition of ascending. By induction, we know that $lm(\{C_1, C_2, \ldots, C_{i-1}\}) = I_0[max(C_1), max(C_2), \ldots,$ $max(C_{i-1})]$. Therefore $I_0[max(C_1), max(C_2), \ldots, max(C_{i-1})]$ does not satisfy C_i. Therefore J does not satisfy C_i, since J agrees with $I_0[max(C_1), max(C_2), \ldots,$ max $(C_{i-1})]$ on all literals smaller than $max(C_i)$, and thus the literals of C_i other than $max(C_i)$ are not satisfied by J either. □

Now, when the search procedure begins, the set $\mathcal{C}$ is empty, and this set is then successively modified by adding to it some clause instance D contradicting its least model. Whenever this is done, it is necessary to do some processing on $\mathcal{C}$ to preserve the ascending property. This processing involves performing certain resolutions, as well as deleting certain elements from $\mathcal{C}$, in a manner that will be described. Letting $simp(\mathcal{C}, D)$ denote the result of this processing, we have the following overall algorithm for ordered semantic hyper-linking:

```
while {} ∉ 𝒞 do
   if lm(𝒞) ⊨ S then return "satisfiable"
   else D ← mi(S, lm(𝒞));
        𝒞 ← simp(𝒞, D) fi
 od;
end while;
return "unsatisfiable"
```

3.1 Processing the list $\mathcal{C}$ of relevant clauses

There are two kinds of operations that take place during the processing of $\mathcal{C}$ involved in the call to "simp." The first kind is to perform ordered resolutions between D and the last clause of $\mathcal{C}$, when possible. For this, we define a_res(C,D) as follows:

Definition 3.4 *Suppose C and D are ground clauses and suppose there is a literal L such that $L = max(C)$ and $\overline{L} = max(D)$. Then $a\_res(C,D) = (C - \{L\}) \cup (D - \{\overline{L}\})$.*

Now, a_res(C,D) will be an (A-ordering) resolution involving the maximal literals in C and D. We note that A-ordering resolution is in itself a complete theorem proving method for propositional logic, and has natural extensions to first-order logic. The second kind of operation that takes place during "simp" is to eliminate elements from $\mathcal{C}$ that are made irrelevant by these A-ordering resolutions, that is, elements of $\mathcal{C}$ that can be eliminated without affecting $lm(\mathcal{C})$. We have the following procedure for "simp":

```
procedure simp([C_1, C_2, ..., C_n], D);
     if max(C_n) and max(D) are complementary then
          D' ← a_res(C_n, D);
          return simp([C_1, C_2, ..., C_{n-1}], D')
     else if max(C_n) > max(D) then
               return simp([C_1, C_2, ..., C_{n-1}], D)
               else return [C_1, C_2, ..., C_n, D] fi fi
end simp
```

To show correctness, we define an interpretation $I^*(\mathcal{C}, D)$ where $\mathcal{C}$ is an ascending list of clauses and D is a clause contradicting $lm(\mathcal{C})$. This interpretation will have the property that $lm(\mathcal{C}) < I^*(\mathcal{C}, D)$ and if $I \leq I^*(\mathcal{C}, D)$ then $I \not\models \mathcal{C} \cup D$. Therefore, if $\mathcal{C} \cup D$ is satisfiable, then $I^*(\mathcal{C}, D) < lm(\mathcal{C} \cup D)$. Thus this definition gives a lower bound on the least model (if it exists). We show that each processing step in "simp" does not decrease $I^*(\mathcal{C}, D)$. And, at the beginning, $I^*(\mathcal{C}, D)$ is larger than $lm(\mathcal{C})$. Therefore, at the end we will obtain an ascending set of clauses whose least model is larger than it was at the beginning. Or, if $I^*(\mathcal{C}, D)$ is the maximal interpretation, that is, the interpretation that disagrees with I_0 everywhere, then this property is preserved, and we will eventually derive the empty clause $\{\}$. $I^*(\mathcal{C}, D)$ can be the maximal interpretation only if $\mathcal{C} \cup D$ is unsatisfiable.

Definition 3.5 *Suppose $\mathcal{C}$ is an ascending list of clauses. Let C_n be the last clause in the list $\mathcal{C}$. Suppose D is a ground clause contradicting $lm(\mathcal{C})$. Define $I^*(\mathcal{C}, D)$ as follows:*

$I^*(\mathcal{C}, D) \models L$ *iff*

1. $L > max(D)$ *and* $I_0 \not\models L$ *or*
2. $L \leq max(D)$ *and* $lm(\mathcal{C}) \models L$

We note that since $I^*(\mathcal{C}, D)$ imitates $lm(\mathcal{C})$ for literals less than or equal to $max(D)$, and $max(D)$ contradicts $lm(\mathcal{C})$, therefore $I^*(\mathcal{C}, D)$ does not satisfy D. Also, for literals larger than $max(D)$, $I^*(\mathcal{C}, D)$ is chosen to be as large as possible in the ordering on interpretations, and for smaller literals, $I^*(\mathcal{C}, D)$ agrees with $lm(\mathcal{C})$. Therefore, $I^*(\mathcal{C}, D) > lm(\mathcal{C})$. (We cannot have equality because $I^*(\mathcal{C}, D)$ differs from I_0 on infinitely many literals, assuming that $\mathcal{A}$ is infinite.) Also, if J is an interpretation and $J < I^*(\mathcal{C}, D)$ then J does not satisfy $\mathcal{C} \cup D$. If $J < lm(\mathcal{C})$ this is immediate. If $J = lm(\mathcal{C})$ then D contradicts J. If $J > lm(\mathcal{C})$ then $d(J, lm(\mathcal{C})) > max(D)$ since $I^*(\mathcal{C}, D)$ and $lm(\mathcal{C})$ agree on literals not larger than $max(D)$. Therefore D contradicts J in this case, too. Thus $lm(\mathcal{C} \cup D) > I^*(\mathcal{C}, D)$. And if there is no least model for $\mathcal{C} \cup D$, then we can at least say that all interpretations J less than or equal to $I^*(\mathcal{C}, D)$ fail to satisfy $\mathcal{C} \cup D$.

To show correctness, we need to show that every processing step in "simp" preserves the property that D contradicts $lm(\mathcal{C})$ and does not decrease $I^*(\mathcal{C}, D)$. Let C_n be the last element of $\mathcal{C}$. If $max(D) > max(C_n)$ then the list $\mathcal{C}$ with D added to the end, is ascending, because D contradicts $lm(\mathcal{C})$. If $max(D) < max(C_n)$ then C_n does not enter into the definition of $I^*(\mathcal{C}, D)$ at all, so deleting C_n does not affect $I^*(\mathcal{C}, D)$. Also, D still contradicts $lm(\{C_1, C_2, \ldots, C_{n-1}\})$ since this cannot be distinguished from $lm(\mathcal{C})$ on literals smaller than $max(C_n)$. If $max(D)$ and $max(C_n)$ are complementary (the only remaining case), then the A-ordering resolution replaces D by a resolvent containing all the literals of D and C_n except the two maximal ones. This resolvent will still contradict the least model of $[C_1, C_2, \ldots, C_{n-1}]$ since C_n did (by the definition of ascending) and all the literals of D except $max(D)$ also contradicted this least model (since D contradicted $lm(\mathcal{C})$). Also, $I^*(\{C_1, C_2, \ldots, C_{n-1}\}, a\_res(C_n, D)) > I^*(\mathcal{C}, D)$ because the maximal literal of $a\_res(C_n, D)$ is smaller than the maximal literal of D.

We now consider the case in which $\mathcal{C} \cup D$ is unsatisfiable. If at some stage in this processing, $I^*(\mathcal{C}, D)$ becomes the maximal interpretation, then no ascending list can be produced, since all ascending lists of clauses are satisfiable. This means that the empty clause $\{\}$ is derived, and the search terminates.

3.2 An example

Suppose that the set $\mathcal{A}$ of atoms is $P_1, P_2, P_3, \ldots$ ordered by $P_1 < P_2 < P_3 < \ldots$ and that S contains the following clauses:

P_5
$\neg P_5, \neg P_8$
$P_6, \neg P_{10}$
$\neg P_6, P_8$
$P_{10}, \neg P_{11}$
P_{11}

Suppose I_0 interprets P_i as **true** for all i, that is, $I_0 \models P_i$. Then we generate the following sequence of current interpretations $I_i = lm(\mathcal{C}_i)$, and the corresponding clauses $mi(S, I_i)$:

I_0	**(now the clause $\neg P_5, \neg P_8$ is chosen)**
$I_0[\neg P_8]$	**(now $\neg P_6, P_8$ is chosen)**
	(resolvent $\neg P_5, \neg P_6$, from the above two clauses)
$I_0[\neg P_6]$	**(now $P_6, \neg P_{10}$ is chosen)**
$I_0[\neg P_6, \neg P_{10}]$	**(now $P_{10}, \neg P_{11}$ is chosen)**
$I_0[\neg P_6, \neg P_{10}, \neg P_{11}]$	**(now P_{11} is chosen)**
	(resolvent P_{10} is generated from $P_{10}, \neg P_{11}$ and P_{11})
	(resolvent P_6 is generated from P_{10} and $P_6, \neg P_{10}$)
	(resolvent $\neg P_5$ is generated from P_6 and $\neg P_5, \neg P_6$)
$I_0[\neg P_5]$	**(now P_5 is chosen)**
	(resolvent $\{\}$ is generated from $\neg P_5$ and P_5)

We show the five ascending lists of clauses that are generated, too:

$\{\neg P_5, \neg P_8\}$
$\{\neg P_5, \neg P_6\}$ **(after "simp" is called)**
$\{\neg P_5, \neg P_6\}$, $\{P_6, \neg P_{10}\}$
$\{\neg P_5, \neg P_6\}$, $\{P_6, \neg P_{10}\}$, $\{P_{10}, \neg P_{11}\}$
$\{\neg P_5\}$

3.3 Completeness

We can argue the completeness of this method in a manner similar to the completeness proof in [PACL92]. However, the fact that our partial orderings may have order type greater than ω complicates the argument a little. Also, the fact that clauses D that are chosen at one point may be discarded later complicates the proof. The general idea is to show that if S is unsatisfiable then there are only a finite number of ground instances that can ever be chosen as minimal contradicting instances. Therefore there are only a finite number of atoms that ever appear in $\mathcal{C}$. This essentially reduces the problem to one involving finite interpretations. Since there are only finitely many finite interpretations, and by the ordering on interpretations, the same one cannot be seen more than once, eventually the method must stop; if S is unsatisfiable, then the only way for the method to stop is to generate the empty clause.

We now show that only a finite number of clauses can be chosen.

Theorem 3.6 *Suppose S is an unsatisfiable set of clauses. Then the set $\{mi(S, I) : I$ is an interpretation over $\mathcal{A}\}$ is finite.*

Proof. Let T be a finite unsatisfiable set of ground instances of clauses in S; such a set T exists, by the so-called Herbrand's theorem. Then for every interpretation I there is a clause D in T such that $I \not\models T$. Note that $mi(S, I)$ is minimal in $<_{cl}$ among clauses contradicting I; thus we cannot have $D < mi(S, I)$. However, for all but finitely many ground clauses C, $D < C$. Thus $mi(S, I) \in \cup_{D \in T}\{D' : \neg(D < D')\}$, which is finite. □

Definition 3.7 *Suppose we arbitrarily choose some finite unsatisfiable set T of ground instances of S. Then we say a ground instance C of S is* small *if it is in the set $\cup_{D \in T}\{D' : \neg(D < D')\}$, and it is* large *otherwise. We say that an atom is small if it appears (positively or negatively) in a small clause, and it is large otherwise. Also, a literal $\neg A$ is small if A is small, otherwise $\neg A$ is large.*

Theorem 3.8 *Ordered semantic hyper-linking is complete, that is, if a set S of clauses is unsatisfiable, then eventually the empty clause $\{\}$ will be derived.*

Proof. As above, we observe that the clauses $mi(S, I)$ will be small for all current interpretation I, and there are only finitely many such clauses. It follows that all the current interpretations I constructed will be of the form $I_0[\mathcal{L}]$, where $\mathcal{L}$ is a subset of the small literals. This is a finite set of interpretations. Each current interpretation constructed is larger than its predecessor in the ordering $>$ on interpretations; thus the same interpretation cannot be seen twice, and the search must eventually stop. The only way that this can happen is for the empty clause to be generated, if S is unsatisfiable. $\square$

3.4 Efficiency

We briefly note one advantage of this approach over semantic hyper-linking as described in [CP94a]; that is that the growth in the number of eligible literals is better controlled. The number of eligible literals has a strong effect on the work required to find a minimal clause D contradicting the current interpretation. The procedure "simp" will automatically perform A-ordering resolutions when the maximal literal of the contradicting clause D is the complement of an existing eligible literal; each such resolution has the effect of removing an eligible literal from $\mathcal{C}$. In semantic hyper-linking without an ordering, it is rare for eligible literals to be removed. Typically the set of eligible literals grows rapidly, making the search procedure time-consuming after a few rounds of search and making it difficult to find proofs that require more than a few rounds of search. However, the other parts of semantic hyper-linking are powerful enough so that many proofs can be found within three or four rounds of search. In addition, the way that the literals are ordered in semantic hyper-linking makes the propositional satisfiability test fast. Still, we think it would be an advantage to be able to handle many rounds of search efficiently.

4 Finding Minimal Contradicting Instances

The preceding discussion has not dealt with the practical aspects of how the minimal contradicting instance $mi(S, I)$ is found. We have dealt with this to some extent in [CP94a]. The problem is, given an interpretation of the form $I_0[L_1, L_2, \ldots, L_n]$ and a set S of (possibly) non-ground clauses, to test whether $I_0[L_1, L_2, \ldots, L_n] \models S$ and if not, to find a minimal ground instance D of some clause C in S such that D contradicts $I_0[L_1, L_2, \ldots, L_n]$ (if such a clause D exists). For this purpose, as we shall see, it is helpful to choose an I_0 that is *decidable*, that is, given a non-ground clause C we can decide whether $I_0 \models C$. There are a number of kinds of interpretations I_0 that are decidable; among them are the syntactic interpretations, interpretations with a finite domain, and interpretations over the reals in which all the functions and predicates can be expressed in terms of linear arithmetic with inequality. We say that an interpretation I is *syntactic* if for any two atoms A and B with the same predicate symbol, $I \models A$ iff $I \models B$. Such interpretations, depending only on the signs and predicate symbols of literals, are fairly limited in expressiveness, but are sometimes useful anyway, as we show in [CP94a]. As an example of an interpretation in terms of linear arithmetic with inequality, we might interpret $P(x, y)$ as $2x > y$ and we might interpret $f(x, y)$ as **if** $(x > y)$ **then** x **else** y. Now, if $I_0 \not\models C$, then we will need to find a ground instance $C\Theta$ of C such that $I_0 \not\models C\Theta$. A problem is that even if $I_0 \not\models C$, such a ground instance may not exist; the reason is that some of the elements of the domain of I_0 may not be values of any finite ground terms. The question, given clause C, does there exist a substitution Θ such that $C\Theta$ is a ground clause and $I_0 \not\models C\Theta$, seems to be harder than deciding if

$I_0 \models C$. We say an interpretation I_0 is *Herbrand decidable* if this question about Θ is decidable. We note that syntactic interpretations and interpretations with a finite domain are Herbrand decidable. We don't know whether interpretations over the reals in which the functions and predicates can be expressed using linear arithmetic with inequality, are Herbrand decidable. If I_0 is not Herbrand decidable, then we may have a current interpretation I that satisfies S without being able to detect this, and we may then spend an infinite amount of time fruitlessly searching for a ground instance $C\Theta$ that contradicts I. However, this will not affect the completeness of ordered semantic hyper-linking, because if such a $C\Theta$ exists, it will eventually be found. If I_0 is Herbrand decidable, then the question whether $I_0[L_1, L_2, \ldots, L_n] \models S$ is also decidable, as we will now show.

We first specify in more detail how such an instance $C\Theta$ can be found, if it exists, and moreover an instance that is minimal, assuming that I_0 is decidable. Our approach is to construct a set Z of literals with the following property: The set of ground instances of literals in Z is exactly the set of ground literals L such that $I_0 \not\models L$ and neither L nor its complement $\overline{L}$ appear in the list $[L_1, L_2, \ldots, L_n]$ of eligible literals. Then one can show that D contradicts $I_0[L_1, L_2, \ldots, L_n]$ iff every literal of D is either the complement of an eligible literal or an instance of a literal in Z. For, literals of D that are complements of some L_i will be false in $I_0[L_1, L_2, \ldots, L_n]$ since $I_0[L_1, L_2, \ldots, L_n] \models L_i$ for all i. Also, literals that are instances of elements of Z will be false in $I_0[L_1, L_2, \ldots, L_n]$ because $I_0[L_1, L_2, \ldots, L_n]$ agrees with I_0 on such literals. To obtain such instances D, then, we can take a clause C of S and apply a substitution Θ such that all literals L of $C\Theta$ are instances of some literal in Z or complements of some eligible literal. That is, we find most general Θ such that for all L in $C\Theta$, there exists a literal M in Z or in the set of complements of eligible literals such that L and M are identical.

We note that finding such a Θ is reminiscent of the hyper-linking method of [LP92]. If all the literals in Z are ground literals, this can be done by a matching procedure, in which the literals of C are matched one by one; if some literals in Z are non-ground, we may need to do successive unifications. Then if the resulting instance D is non-ground, it is necessary to instantiate the variables with ground terms in some manner. Since we are only interested in minimal ground instances, we can replace each variable by a ground term that is minimal in the clause ordering, assuming that the clause ordering $<_{cl}$ is *monotone*. For our purposes, we say that the clause ordering is monotone if it can be extended to a partial ordering on terms such that for terms s and t, $s <_{cl} t$ implies $C[s] <_{cl} C[t]$ for a clause $C[s]$ containing an occurrence of s. Furthermore, we require that for any term t there are at most finitely many terms s for which $\neg(t <_{cl} s)$. Note that this implies that there are at most finitely many minimal terms, by the condition just given. However, in order to find a minimal instance, it may be necessary to replace each variable by all minimal terms in all possible ways, which can be expensive. For this purpose, we write $s\#t$ to mean $\neg(s > t) \wedge \neg(t > s)$ and we write $C\#D$ similarly for clauses C and D. Then we can choose an ordering $<_{cl}$ so that $s\#t$ implies $C[s]\#C[t]$; for example, ordering terms by their size satisfies this condition. For such an ordering, we can obtain a minimal instance of C by replacing all variables by arbitrarily chosen minimal terms.

Now, the problem is to generate Z. Without knowing more about I_0, not much can be said. If the test $I_0 \models L$ is decidable for ground L, then one can enumerate all ground literals L, discard eligible literals and their complements, and test whether $I_0 \models L$, and in this way generate, or at least enumerate, Z. The fact that Z is infinite need not be a problem, because one only needs to enumerate Z in ascending order in the ordering $<_{cl}$ in order to find a minimal contradicting instance D. As soon as one instance D has been found, literals L such that $D <_{cl} L$ need not be examined, and this eliminates all but finitely many literals. For this we need to assume that if $L \in C$ and $L \neq C$ then $L <_{cl} C$. In [CP94b] we indicate how specialized decision procedures can be used to aid in the generation of such sets Z of literals, in some cases. Of course, such an enumerative method cannot detect if the current interpretation satisfies C (or S). Here we choose a different approach that permits a relatively small set Z' to be generated independent of I_0, and delays the consideration of I_0 to a later stage. This approach also permits us to detect if the current interpretation satisfies S, if I_0 is Herbrand decidable.

We make some comments about the complexity measures used. These are defined in terms of the sizes of various structures, considered as character strings. Alternatively, the size of a clause, set of clauses, etc. is the number of occurrences of symbols in it. So if we say that something can be computed in time polynomial in a set S of clauses, we mean that the running time is bounded by a polynomial in the length of S, written out as a character string; this is the usual complexity measure.

4.1 Disunification

Let Z' be a set of literals such that a ground literal L is an instance of a literal in Z' exactly when neither L nor its complement appear in the set of eligible literals. Thus we have a kind of disunification problem. It turns out that a finite set Z' always exists, since the eligible literals are ground and there are finitely many of them. We are using the fact that there are only finitely many function and predicate symbols in all. Also, Z' can be computed in time polynomial in the list of eligible literals. To see this, let $Z'(E)$ be a desired Z' of literals as desired where E is the set of literals that are either eligible literals or their complements. We first note that the positive and negative literals can be handled separately. Let L_{pos} be the set of all positive literals and let L_{neg} be the set of all negative literals. Let E_{pos} be the positive literals in E and let E_{neg} be the negative literals in E. Then $Z'(E) = (Z'(E_{pos}) \cap L_{pos}) \cup (Z'(E_{neg}) \cap L_{neg})$. We can then extend this further, to consider the sets of positive and negative literals having specified predicate symbols, and solving for Z' for each such subclass. If there are n predicate symbols in all, we obtain a total of $2n$ subproblems to solve, whose solutions can then be combined to obtain the desired Z'. If a predicate symbol with a specified sign does not appear in E, the subproblem has a simple solution; then $P(x_1, \ldots, x_n)$ or $\neg P(x_1, \ldots, x_n)$ is in $Z'(E)$ for suitable P, since none of its instances will be in E. Otherwise, we can consider the sets of literals having various function or constant symbols in some chosen position. Each such further division reduces the problem into a number of subproblems equal to the number of function and constant symbols, and partitions the set E further into disjoint subsets; eventually we obtain trivial problems, whose solutions can be combined.

4.2 Generating eligible instances

Suppose the set Z' of literals has been generated, as specified above. We generate instances of the input clauses as follows: For each clause C in S, we partition C into two disjoint sets of literals C_α and C_β. This is done in all possible ways. For each such partition, we unify the literals in C_α with complements of eligible literals, and those in C_β with literals from Z'. This also must be done in all possible ways. In this way, we obtain an instance D of C which can be expressed as $D_\alpha \cup D_\beta$ where the literals in D_α are complements of eligible literals and the literals in D_β are instances of literals of Z', that is, they do not unify with eligible literals or their complements. We call such an instance an *eligible instance*. Note that there are only finitely many eligible instances, and they can be found by a simple enumeration procedure. In order to ensure that D contradicts $I_0[L_1, L_2, \ldots, L_n]$, it is necessary to find a substitution γ such that $I_0 \not\models (D_\beta)\gamma$. We also want this instance to be minimal in the clause ordering $<_{cl}$. If γ is as specified, then $I_0[L_1, L_2, \ldots, L_n] \not\models D\gamma$, reasoning as above: The literals in D_α are not satisfied by the current interpretation because they are complements of eligible literals. The literals in $(D_\beta)\gamma$ are not satisfied by the current interpretation $I_0[L_1, L_2, \ldots, L_n]$ because they are not satisfied by I_0. If I_0 is Herbrand decidable, then the test whether such a γ exists is decidable, and in this way we can test if the current interpretation satisfies S. If such a γ exists, we want to find one such that $D\gamma$ is minimal in the ordering $<_{cl}$. The search for γ depends of course on I_0 and $<_{cl}$. If the clause ordering is based on the *directed acyclic graph size* of the terms, that is, the size is the number of distinct subterms that appear, independent of how often they appear, then this computation seems to be rather complicated, despite the favorable theoretical properties of this size measure. However, for certain types of interpretations and literal orderings, the search for such a γ producing a small clause can be done quickly, as we will show in the next section. This is also dealt with to some extent in the paper [CP94b]. It is likely also that this search can always be done fast if I_0 is an interpretation with a (small) finite domain and if the clause ordering $<_{cl}$ is monotone. However, we can say something more in case I_0 is syntactic. If I_0 is syntactic, then $I_0 \not\models (D_\beta)\gamma$ iff $I_0 \not\models D_\beta$, which latter condition can be checked just by examining the signs and predicate symbols of the literals. This permits some clauses to be rapidly eliminated from further processing.

4.3 Complexity analysis

Now, the generation of all the eligible instances requires an amount of work that depends on the number and sizes of the eligible literals and the number of literals in the clauses. However, suppose that there is a constant bound on the number of literals in each clause of S. Then the generation of the eligible instances can be done in time polynomial in S and the set of eligible literals, since one has to look at all combinations obtained by unifying each literal in C with each complement of an eligible literals or with each element of Z'. If the set of eligible literals unioned with Z' has n elements and a clause C has k literals, then there are n^k possibilities, which is still polynomial, for a fixed k. For each possibility, the work is polynomial (or even linear), using efficient unification algorithms. Note that n is polynomial because the number of elements of Z' is polynomial, by the disunification arguments given above. The assumption that k is bounded is reasonable, because it is possible to convert clauses with many literals into clauses with three literals in a satisfiability-preserving manner by introducing new predicate symbols.

We have just shown that if the number of literals in clauses of S is bounded, then the eligible instances can be generated in polynomial time, regardless of I_0. However, in order to generate a minimal contradicting instance, it is necessary to instantiate the eligible instances with a substitution γ as specified above. If I_0 is syntactic and the clause ordering $<_{cl}$ is monotone and has properties concerning # as specified in section 4, then such a γ can be obtained simply by checking the signs and predicate symbols and replacing all variables by a minimal term; the entire process of generating a minimal contradicting instance can then be done in time polynomial in S and the set of eligible literals, that is, polynomial in their lengths written out as character strings. If I_0 has a finite domain, then we can find for each domain element d a minimal ground term t_d whose interpretation under I_0 is d; such terms can be found by a simple iteration procedure. Then to find a minimal instance $D\gamma$ such that $I_0 \not\models (D_\beta)\gamma$, it is only necessary to consider γ replacing variables by terms of the form t_d for various d in the domain. This is a finite number of possibilities, exponential in the number of variables in D_β. If we use an ordering based on directed acyclic graphs, that is, counting the number of distinct subterms that appear, but not counting how often they appear, then the generation of a minimal contradicting instance can be more complicated. This is because such directed acyclic graph-based orderings are not monotone, since the contribution of a subterm to the clause depends on subterms that appear elsewhere in the clause. However, because of the favorable complexity properties of the directed acyclic graph-based orderings, it may be worthwhile to use them anyway. In general, it makes sense to look at the eligible instances in order of their size; then it is possible that a small contradicting instance may be found early and the instantiation of the larger eligible instances may be avoided.

5 Additional Ordered Resolutions

Since the basic search procedure performs A-ordering resolution, it seems reasonable to do additional resolutions; these might have the effect of doing part of the work in advance, and thereby speed up the search. So we might add to the input clause set S a set S' of clauses generated by A-ordering resolution from S, and consider these clauses S' as additional input clauses. One would expect that these additional clauses might lead to the finding of certain contradictions earlier. They also have the effect of eliminating large literals from proofs, as mentioned in [CP94a]. Since the basic search procedure has difficulty generating large literals, this combination is reasonable, and we have found that in practice such a combination (with rough resolution instead of A-ordering resolution to eliminate large literals) often works well. We can balance the work between ordered semantic hyper-linking and A-ordering resolution in some way so that both are done in parallel; one method to use is to divide the total time spent equally between them.

The question remains what ordering to use for these A-ordering resolutions, and which A-ordering resolutions to do. As for the ordering, we can choose some literal ordering $<_{lit}$ compatible with the ordering $<$ on ground literals. That is, we can say $L <_{lit} M$ (for non-ground literals L and M) if there is a ground substitution Θ such that $L\Theta < M\Theta$. Note that we really need to use the ordering $<$ here, not $<_{cl}$, to compare $L\Theta$ and $M\Theta$. Then we can restrict literals resolved on, to literals that are maximal in this ordering $<_{lit}$. As for which A-ordering resolutions to perform, we observed in section 4.3 the strong dependence of the efficiency of ordered semantic hyper-linking on the number of literals in clauses of S, so it is reasonable to keep the number of literals small. However, we want to perform the A-ordering search in a complete manner, so that a reasonably large portion of the search space is explored. For this, it is necessary to consider not just the number of literals in a clause, but the size (number of symbol occurrences) of the clause. Preferring clauses of small size can be done in a completenesss-preserving manner, and will also tend to keep the number of literals small. The question remains exactly how this can be done. One way to do this is to keep a list of all pairs of clauses that have not yet been resolved together, and always resolve the pair of clauses whose sum of sizes is as small as possible. This produces clauses of small size (number of symbol occurrences), which will tend to have few literals, but requires a quadratic amount of space to store these possible resolutions. Another method is the Otter [McC89] approach, in which a small clause is repeatedly chosen and resolved against all other clauses. This avoids the expensive bookkeeping, but has the problem that this small clause will also resolve against large clauses, possibly producing large clauses very early. We propose a compromise, in which a list $\mathcal{L} = C_1, C_2, C_3, \ldots$ of clauses is constructed as follows: The clauses are entered into this list $\mathcal{L}$ smallest first. Whenever a clause is entered into the list, it resolves (using A-ordering resolution) against all clauses that are already on the list. In this way, all resolutions will eventually be done, but we have some guarantee that when two clauses are resolved, both of them are small. Also, the amount of bookkeeping is kept to a minimum. When we are done resolving C_i against C_j for all $j \leq i$, then we find the smallest clause D such that D is in S or has been produced by an earlier resolution, and such that D is not already in the list $\mathcal{L}$. We then enter this smallest clause D in the list as C_{i+1} and resolve D against C_j for $1 \leq j \leq i$, continuing the process.

Now, for certain problems, it is important to produce large A-ordering resolvents early, if they are directly derived from the negation of the theorem; this is the case for example in Bledsoe's five limit problems [Ble90]. Therefore, we would like to make this A-ordering resolution procedure sensitive to support criteria as well as size. For this purpose, we can say that a clause C is *semantically supported* if it contradicts the user-given interpretation I_0. Both input clauses and clauses generated by resolution can be semantically supported. We then can modify the above A-ordering search strategy so that on alternate choices of D, a semantically supported clause is chosen. Thus, we alternate between choosing D as the smallest clause not yet in $\mathcal{L}$, and the smallest semantically supported clause not yet in $\mathcal{L}$; this D is then added to the list and resolved against all clauses already in $\mathcal{L}$. This will tend to favor resolutions involving supported clauses, even if the clauses are large. This can only be done systematically if I_0 is decidable (or better, Herbrand decidable).

5.1 Explanation-based generalization

The idea of explanation-based generalization (EBG) is to extract some general principles from a specific argument, enabling the argument to be applied to a wider range of situations. This principle can be applied in the ordered resolution phase of ordered semantic hyper-linking. We note that in the procedure "simp" of section 3.1, A-ordering resolutions are performed between ground clauses C_n and D; both C_n and D are instances of clauses in S or clauses produced from S by prior A-ordering resolutions. Suppose C'_n and D' are the more general (possibly non-ground) clauses of which C_n and D, respectively, are instances. Then by general properties of resolution, it follows that there is an A-ordering resolvent C of C_n and D such that a_res(C_n, D) is an instance of C. Therefore, it seems reasonable to store these more general clauses C'_n, D', and C along with their respective instances C_n, D, and a_res(C_n, D). In this way, we produce lemmas that can be added to the set of input clauses; such lemmas are likely to be relevant to the proof and may be generated again during the search. By adding them to S, we may avoid sections of the search in which the same resolutions are performed over and over again. It makes sense to generate these lemmas, in addition to performing the A-ordering resolutions of the previous section, since these lemmas may not be generated by the list-based search method described there. The fact that a uniform ordering is used in the search may make the lemma mechanism more effective; the search method used in semantic hyper-linking without ordering can vary the literal ordering for different interpretations, making it less likely that a lemma found earlier will be useful later on.

6 Replacement Rules

We found that the use of replacement rules considerably enhanced the performance of semantic hyper-linking, and so it is reasonable to include them in ordered semantic hyper-linking, too. However, it is necessary to adapt them to the current context. Suppose $C_1, C_2, \ldots, C_n$ is the current set $\mathcal{C}$ of relevant clauses. From this we construct a set EL of *explicit literals*; these are the literals satisfied by $lm(\mathcal{C})$

that actually appear in the clauses C_i. Thus, the set EL is defined as the set of literals L such that either L or $\overline{L}$ is a member of $\cup_i C_i$ and such that $lm(\mathcal{C}) \models L$. Now, if we can show that $EL \cup S$ is unsatisfiable, then we have already contradicted the current interpretation $lm(\mathcal{C})$ and need not search for a minimal contradicting instance. For, if we can show that $EL \cup S$ is unsatisfiable, then we know that the clause $\{\overline{L} : L \in EL\}$ is a logical consequence of S, and can be used as if it were the minimal contradicting instance. In fact, we typically find a small subset of the clause $\{\overline{L} : L \in EL\}$ that is a consequence of S, which may be more useful for the search.

To see whether $EL \cup S$ is unsatisfiable, we use some incomplete but often effective methods. In the general AI context, these may be viewed as "obvious inferences" or "associations" that are readily made. One method is to use *natural replacement rules*; these are implications of the form $L_1 \wedge L_2 \wedge \ldots \wedge L_n \rightarrow L$ such that all variables in the literal L appear elsewhere in the implication, and such that some clause $\{\overline{L_1}, \overline{L_2}, \ldots, \overline{L_n}, L\}$ is a member of S. These replacement rules may be used by unifying the L_i with elements of EL; then the corresponding instance of L will be a ground literal (since all variables of L also appear elsewhere). This ground literal is a logical consequence of EL and can be (temporarily) added to EL. This operation may be repeated a number of times, and if complementary literals appear in EL, then we know that $EL \cup S$ is unsatisfiable. From this derivation, a clause that is a subset of $\{\overline{L} : L \in EL\}$ may be extracted, which is a logical consequence of S, and can be used in place of the minimal contradicting instance.

Another kind of replacement rules are *definitional* replace rules; this is a slight simplification of the *minimal* replace rules of [CP93a]. The idea of definitional replace rules is to capture clauses that represent definitions, and the effect of applying them is to expand the definitions; this is particularly useful in set theory and modal logic. A definitional replace rule is of the form $L \rightarrow L_1 \vee L_2 \vee \ldots \vee L_n$ such that all variables in $L_1, L_2, \ldots, L_n$ appear in L and such that some clause $\{\overline{L}, L_1, \ldots, L_n\}$ is a member of S. If one has a definition of the form $L \equiv A$ where L is a literal and A is a formula involving no quantifiers, one can verify that such clauses will be generated when this definition is converted to clause form. Sometimes such formulae are generated even if A contains quantifiers.

Such replacement rules are used in an inverse way to natural replacement rules; if L has an instance $L\Theta$ in EL then the implication $L\Theta \rightarrow L_1\Theta \vee L_2\Theta \vee \ldots \vee L_n\Theta$ can be added to EL. Also, if L has an instance $L\Theta$ appearing on the right-hand side of some such implication that has previously been added to EL, then this instance can be added to EL. In this way, EL is augmented by a set of ground instances of clauses in S. These instances are the instances that one would consider when expanding definitions. We can then test if EL together with these ground instances is unsatisfiable, using something like Davis and Putnam's method [DP60]. If so, then a clause that is a subset of $\{\overline{L} : L \in EL\}$ may be extracted, which is a logical consequence of S; thus we know that $EL \cup S$ is unsatisfiable.

In order to control the application of replace rules, it seems most reasonable to use a time bound based on the time used to search for a minimal contradicting instance. It seems reasonable to first search for a contradiction using natural replace rules. If this fails, then definitional replace rules can be applied, again controlled by the time bound. If this fails, then we have generated a set G of ground clauses that can be processed a little more; this set G contains EL (considered as unit clauses) together with the ground instances generated by definitional replacement. Since G consists of ground clauses, it has only finitely many models. We can examine all these models J of G one by one, and for each such model J, we can again apply natural replacement to J to see if it can be contradicted. We note that since G is finite, the models J are essentially finite, too. Each such model J can be considered as a set $\mathcal{U}$ of unit clauses, that is, the set of literals L such that $J \models L$ and such that L or its complement appears in G. Then we can perform natural replacement on this set $\mathcal{U}$; this may find some contradictions that were nearly, but not quite, found by definitional replacement. Such contradictions (demonstrations that $S \cup \mathcal{U}$ is unsatisfiable) need to be found for *all* models J of G in order to demonstrate that $S \cup EL$ is unsatisfiable.

7 Complexity Analysis

We now consider the complexity required by ordered clause linking, in terms of the complexity of the shortest proof from S. We consider both worst-case bounds and give plausibility arguments for better performance. In the introduction, we noticed that clause linking has a triple exponential bound in the complexity of the proof, which can be reduced to double exponential if a suitable ordering on literals is used. The same arguments apply to ordered clause linking. However, we have reason to believe that the performance will be better than this. We know that if S is unsatisfiable then there is an unsatisfiable set T of ground instances of S. Suppose we measure the complexity of the proof by the complexity $c(T)$ of T, that is, its length when written out as a character string. Or, equivalently for our purposes, we can measure the complexity of the proof by the complexity of the largest clause in T. Note that this complexity measure does not economize on repeated occurrences of the same subterm. Now, suppose our clause ordering is based simply on the length of the clause, written out as a character string, that is, the sum of the sizes of the literals in the clause. Then each clause appearing in the proof has complexity at most $c(T)$, so we will find the proof when all clauses of complexity $c(T)$ or less have been generated (or earlier). The number of such clauses is exponential in $c(T)$, and so the time required to test their satisfiability may be double exponential in $c(T)$. We note that this measure is independent of how many clauses appear in T, if the "largest clause" complexity measure is used. This will give our method a better comparison with the methods of [Gou94], which need to count the number of elements in T.

Another favorable factor for our method is that the satisfiability of a set of ground clauses can be tested in expected polynomial time, for many probability distributions. And in practice, methods similar to Davis and Putnam's method often decide satisfiability of sets of propositional clauses very fast. We have also found this to be the case in the clause linking theorem prover. Since our search procedure is similar to Davis and Putnam's procedure in its systematic search for a model, we would expect a similar time bound to apply to it, too; thus we may expect in practice that the time required by our method is single exponential. We note further that this is based on the assumption that all clauses of complexity $c(T)$ or less are generated. Our method is very selective about which clauses are generated, so that it is reasonable to assume that only a small subset of the clauses of complexity $c(T)$ or less will be generated. For example, we can show that our method will only generate *logically minimal* ground instances, that is, ground instances that are not logical consequences of smaller (with respect to $<_{cl}$) ground instances. Equivalently, a ground instance C is logically minimal if there is some interpretation I such that C is a minimal instance of S contradicting I. The question whether a clause is a logical consequence of simpler clauses is also relevant for the methods of [BG90, BG94], it turns out. One would expect that the number of logically minimal ground instances of a given size is much smaller than the total number. In fact, we do not even generate all the logically minimal clauses. The fact that A-ordering resolutions are done in the "simp" procedure means that many interpretations are not even examined. That is, we only generate clauses that are logically minimal when such A-ordering resolvents are also considered, so this can eliminate some clauses from being logically minimal. But, as a worst case bound, our method is exponential in the number of logically minimal ground instances; this is always finite when S is unsatisfiable.

One might ask whether it would be just as efficient to generate *all* ground instances of clauses in S, and then consider them in order of size. Thus we would have a list $C_1, C_2, C_3, \ldots$ of all ground instances of S, with the smallest ones occurring earlier in the list. We could then test if C_i is a logical consequence of C_i, for $i \leq j$, and if so, delete C_i from the list. This produces a smaller sublist $C_1', C_2', C_3', \ldots$ of ground instances, of which finite prefixes can be tested for satisfiability. Our method is more efficient in that the non-logically minimal instances are never even generated. Furthermore, even some of the logically minimal instances are avoided, as explained above.

7.1 Estimating the size of the tree

We can give additional evidence that the work required by ordered semantic hyper-linking is often small. We note that it is only the small atoms that influence the search, as defined in definition 3.7. Let us consider the smallest n atoms in $\mathcal{A}$ and the probability that a clause C over these atoms will contradict an arbitrary interpretation I. Suppose that there are m 3-literal ground clauses C in all. Then the chance that a random interpretation I will not satisfy a specific clause C is $1/8$ (since each of 3 literals must be mapped to $false$). Therefore the chance that the interpretation will satisfy the clause C is $7/8$;, and the chance that I will satisfy m 3-literal clauses is $(7/8)^m$, if the clauses are chosen independently. The expected number of models of m independently chosen clauses (considering only the first n literals) is then $2^n(7/8)^m$ since there are 2^n interpretations altogether for n atoms. We note that $(7/8)^6 < 1/2$, so if $m \geq 6n$ then $2^n(7/8)^m < 1$ and the expected number of models is less than one. This means that the probability is very small that we will have to search past the first n atoms to find a contradicting instance. This is evidence that if there are many clauses then the size of the tree is small, on the average. What we are given is non-ground clauses, in general, instead of ground instances, so the determining quantity for this analysis is the number of their ground instances of various sizes, and how they depend on one another.

References

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

[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.

[BG94] Leo Bachmair and Harold Ganzinger. Ordered chaining for total orderings. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, pages 435–450, New York, 1994. Springer-Verlag.

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

[Ble90] W. W. Bledsoe. Challenge problems in elementary calculus. *Journal of Automated Reasoning*, 6:341–359, 1990.

[CP93a] Heng Chu and D. Plaisted. Model finding in semantically guided instance-based theorem proving. *Fundamentica Informaticae*, 1993. to appear.

[CP93b] Heng Chu and D. Plaisted. Rough resolution: a refinement of resolution to remove large literals. In *Proceedings of the Eleventh National Conference on Artificial Intelligence(AAAI-93)*, pages 15–20, July 1993.

[CP94a] 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 192–206, 1994. Lecture Notes in Artificial Intelligence 814.

[CP94b] Heng Chu and D. Plaisted. The use of presburger formulas in semantically guided theorem proving. In *Third International Symposium on Artificial Intelligence and Mathematics*, January 1994. no proceedings.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North-Holland, Amsterdam, 1990.

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

[FLTZ93] C. Fermueller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Methods for the Decision Problem*. Springer-Verlag, New York, 1993. Vol. 679 of *Lecture Notes in Computer Science*, Springer, Berlin.

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

[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.

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

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

[McC89] W. W. McCune. *Otter 1.0 Users' Guide*. Mathematics and Computer Science Division, Aggonne National Laboratory, Argonne, Illinois, January 1989.

[PACL92] D. Plaisted, G. Alexander, Heng Chu, and S.-J. Lee. Conditional term rewriting and first-order theorem proving. In *Proceedings of the Third International Workshop on Conditional Term-Rewriting Systems*, pages 257–271, July 8 - 10 1992. Invited talk.

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

[Pla93] D. Plaisted. Equational reasoning and term rewriting systems. In D. Gabbay and J. Siekmann, editors, *Handbook of Logic in AI and Logic Programming*, volume 1. Oxford University Press, 1993.

[Pla94a] D. Plaisted. The search efficiency of theorem proving strategies. In *Proceedings of the Twelfth International Conference on Automated Deduction*, pages 57–71, 1994. Lecture Notes in Artificial Intelligence 814.

[Pla94b] D. Plaisted. The search efficiency of theorem proving strategies: an analytical comparison. Technical Report MPI-I-94-233, Max-Planck Institut fuer Informatik, Saarbruecken, Germany, 1994.

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

Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server `ftp.mpi-sb.mpg.de` under the directory pub/papers/reports. If you have any questions concerning ftp access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Regina Kraemer
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `kraemer@mpi-sb.mpg.de`

MPI-I-94-234	S. Matthews, A. K. Simpson	Reflection using the derivability conditions
MPI-I-94-233	D. A. Plaisted	The Search Efficiency of Theorem Proving Strategies: An Analytical Comparison
MPI-I-94-232	D. A. Plaisted	An Abstract Program Generation Logic
MPI-I-94-230	H. J. Ohlbach	Temporal Logic Proceedings of the ICTL Workshop
MPI-I-94-228	H. J. Ohlbach	Computer Support for the Development and Investigation of Logics
MPI-I-94-226	H. J. Ohlbach, D. Gabbay, D. Plaisted	Killer Transformations
MPI-I-94-225	H. J. Ohlbach	Synthesizing Semantics for Extensions of Propositional Logic
MPI-I-94-224	H. Aït-Kaci, M. Hanus, J. J. M. Navarro	Integration of Declarative Paradigms Proceedings of the ICLP'94 Post-Conference Workshop Santa Margherita Ligure, Italy
MPI-I-94-223	D. M. Gabbay	LDS – Labelled Deductive Systems Volume 1 — Foundations
MPI-I-94-218	D. A. Basin	Logic Frameworks for Logic Programs
MPI-I-94-216	P. Barth	Linear 0-1 Inequalities and Extended Clauses
MPI-I-94-209	D. A. Basin, T. Walsh	Termination Orderings for Rippling
MPI-I-94-208	M. Jäger	A probabilistic extension of terminological logics
MPI-I-94-207	A. Bockmayr	Cutting planes in constraint logic programming
MPI-I-94-201	M. Hanus	The Integration of Functions into Logic Programming: A Survey
MPI-I-93-267	L. Bachmair, H. Ganzinger	Associative–Commutative Superposition
MPI-I-93-265	W. Charatonik, L. Pacholski	Negativ set constraints: an easy proof of decidability
MPI-I-93-264	Y. Dimopoulos, A. Torres	Graph theoretical structures in logic programs and default theories
MPI-I-93-260	D. Cvetković	The logic of preference and decision supporting systems
MPI-I-93-257	J. Stuber	Computing Stable Models by Program Transformation