1997 Final Report

Describing new results under the research project entitled

Complexity of Algorithms for Problems in Proposition Logic

Covering the period

January 1, 1994 - March 31, 1997

Funded by the

Office of Naval Research
Mathematical Sciences Division
800 N. Quincy
Arlington, VA 22217-5660

Navy Grant No. N00014-94-1-0382

Submitted by

Professors John Franco and John Schlipf,
Principal Co-Investigators,
Department of Computer Science,
University of Cincinnati,
Cincinnati, Ohio 45221-0008

April 26, 1997

Contents

Preface

This is a self-contained final report intended to: (1) state the principal results; (2) explain, briefly, necessary terminology; (3) provide a motivating context; and (4) discuss the impact of the results. While the report is aimed primarily at the program manager, it contains sufficient detail to be circulated and archived.

The report has the following organization:

Significance of and summary of main results are enumerated in Section 1.

Motivation and terminology for the two main topics, polynomial-time subclasses of satisfiability and algorithms for propositional logic programs, are in Sections A.1 and A.2. The beginning of each section provides background needed to understand the significance of the results. Each subsection is associated with a particular technical points. Results are stated in these subsections as propositions without proof and are summarized in Section 1.

Professional activities and invitations are listed in Section A.3. These include conferences, workshops, journal articles, editing, invitations, and external Ph.D. thesis committee work.

Collaborators on our research topics are credited in Section A.4.

1 Main Results and Their Significance

Our work under ONR sponsorship has produced results illuminating the nature of certain well-known polynomial time subclasses of Satisfiability. Most of these classes are interesting because they have arisen from consideration of Linear Programming concepts to formulations of Satisfiability. We have also achieved progress in the related area of the well-founded semantics for logic programming. This section presents an overview of our results. More detail is provided in Appendix A. Both this Section and Appendix A are updated versions of the 1995 annual report submitted to the Office of Naval Research in October, 1995.

1.1 Polynomial Time Subclasses of Satisfiability

A focus of considerable research is the characterization of subclasses of CNF Satisfiability that can be solved in polynomial time. The hope is that suitably large, non-trivial classes can be found, particularly classes of unsatisfiable formulas.

1.1.1 Single look-ahead unit resolution

We investigate the class of problems solved with single look-ahead unit resolution (SLUR). This class, discovered by us in 1994, is peculiar in that it may be defined based on an algorithm rather than on properties of formulas. The algorithm, called SLUR, selects variables sequentially and arbitrarily and considers both possible values for each selected variable. If, after a value is assigned to a variable, unit resolution does not result in a clause that is falsified, the assignment is made permanent and variable selection continues. If all clauses are satisfied after a value is assigned to a variable (and unit resolution is applied), the algorithm returns a satisfying assignment. If unit resolution, applied to the given formula or to both sub-formulas created from assigning values to the selected variable on the first iteration, results in a clause that is falsified, the algorithm reports that the formula is unsatisfiable. Otherwise, SLUR reports it has given up. An implementation of algorithm SLUR is given in Appendix A.1.1. From now on we use SLUR to mean the class and the algorithm.

The significance of our results on SLUR, detailed in Appendix A.1.1, is as follows.

1. "The results give a simple but enlightening characterization of a class of SAT problems that can be solved in linear time by unit resolution with 1-step look-ahead. Extended Horn and balanced formulas are special cases." (comment of

an OR reviewer - Timberline conference)

2. At least two major classes subsumed by SLUR require a recognition step before a polynomial time algorithm can be applied to solve a given formula. In the case of hidden extended Horn formulas the question of recognition is not known to be solved in polynomial time.

3. The class defined by SLUR seems to be much larger than either hidden extended Horn or balanced formulas.

The following are some comments on these results.

1. The most serious drawback with SLUR is that it cannot determine unsatisfiability unless the given formula is rather trivially unsatisfiable.

2. "I find it interesting that the algorithm seems simpler than the conditions under which it is a decision procedure. It seems almost simpler to give the algorithm and then define the decidable class as the class on which the algorithm determines satisfiability." (comment of an AI reviewer - Timberline conference)

3. It seems the SLUR idea can be generalized to subsume the class of q-Horn formulas as well. Such an algorithm has been announced by Boros [3].

4. The algorithms associated with the class of balanced formulas seem quite complex in comparison to SLUR and several detailed papers have been written about these algorithms.

1.1.2 Single look-ahead with limited backtracking

There are several ways to improve SLUR. One is to add power to the look-ahead step. For example, just adding a 2-SAT checker enables SLUR to solve numerous formulas that it previously could not. Algorithm SLUR can also be improved by adding a form of limited backtracking. Specifically, we choose a polynomial p and allow SLUR, given a formula $\mathcal{C}$ as input, to backtrack at most $p(|\mathcal{C}|)$ times before giving up if satisfiability or unsatisfiability has not been determined. The resulting modification of SLUR is called ISLUR for improved single lookahead unit resolution.

The significance of our results on ISLUR, detailed in Appendix A.1.2, are:

1. It appears that adding limited backtracking to SLUR does not significantly widen the class of unsatisfiable formulas solved in polynomial time. We define

a *sparse* formula and observe that a preponderance of unsatisfiable formulas are sparse. We show that ISLUR gives up on any sparse formula.

2. The above pessimistic result holds for several polynomial time tests for unsatisfiability that can be added to the unit resolution look-ahead of SLUR.

3. It remains to be seen whether limited backtracking significantly adds to the class of satisfiable formulas solved in polynomial time.

The following is a comment on these results.

> The result of [11] is that all resolution proofs of a k-CNF formula are exponential in length if, "for some numbers a, d, such that $0 < d < a/8$, (1) for every family $\mathcal{C}'$ of at most an clauses from $\mathcal{C}$ there are at least $|\mathcal{C}'|/2$ variables involved in precisely one clause from $\mathcal{C}'$; (2) there is a collection of sets D of dn variables such that, for every family $\mathcal{C}'$ of at most an clauses from $\mathcal{C}$, every truth assignment with domain D has an extension that satisfies $\mathcal{C}'$." We view our result as stronger due to the weakness of (1): our pessimistic result holds even when there are families with fewer than $|\mathcal{C}'|/2$ variables involved in one clause.

1.1.3 Hierarchy of implicational formulas

Falsifiability for pure implicational formulas (only the implies operator $\rightarrow$ is allowed) is surprisingly rich in providing hierarchies of subclasses of progressively increasing hardness, eventually reaching subclasses that are NP-complete [27]. This richness may be extended to general propositional formulas.

We studied both falsifiability and satisfiability for the broader class of *implicational formulas* (where only $\rightarrow$ plus a constant $\mathbf{f}$ for false are allowed). We have investigated an algorithm IMP that has the following characteristic. Given an implicational formula $\mathcal{F}$ with each atom occurring in $\mathcal{F}$ at most twice and with at most k occurrences of $\mathbf{f}$, algorithm IMP determines satisfiability of $\mathcal{F}$ in $O(k^k|\mathcal{F}|)$; we show a similar bound applies for falsifiability. Heusch's comments show that his falsifiability problem reduces easily to ours. (Replace Heusch's distinguished variable z with $\mathbf{f}$.)

The following summarize the results of Appendix A.1.3.

1. Algorithm IMP determines satisfiability for class S_k (the set of all implicational formulas with at most k occurrence of $\mathbf{f}$ and all variables occurring at most twice) with $O(k^k|\mathcal{F}|)$ complexity. This is an improvement over the best existing algorithm which has complexity $O(|\mathcal{F}|^k)$. We thus also have a polynomial time

algorithm for falsifiability even if k grows as fast as $\log(|\mathcal{F}|)/\text{loglog}(|\mathcal{F}|)$. The problem is NP-complete for arbitrary k.

2. It is straightforward, and efficiently accomplished, given a k-CNF formula $\mathcal{C}$, to construct an implicational formula $\mathcal{F}$, or, given an implicational formula $\mathcal{F}$, to construct a CNF formula $\mathcal{C}$, where $\mathcal{C}$ is satisfiable if and only if $\mathcal{F}$ is falsifiable. However, we know of no direct and succinct expression of the class of multi-level formulas that transforms to an implicational formula with at most k occurrences of $\mathbf{f}$. Thus, we view the implicational form as facilitating a simple expression of a hierarchy of classes of multi-level formulas of possibly progressively increasing hardness.

3. In light of the last point, the study of such a hierarchy might in some sense reveal the boundary between polynomially solvable and hard classes of satisfiability.

4. The hierarchy proposed by Heusch differs from that of, for example, Gallo and Scutella [24] in that the test for membership in theirs can be accomplished in $O(|\mathcal{F}|^k)$ steps whereas the test for membership in Heusch's hierarchy is trivial.

Our results are important because they show that Heusch's hierarchy is fixed-parameter-tractable (the first SAT hierarchy we know of to have this property) and suggest that study of Heusch's hierarchy may reveal the nature of easy and hard problems. See Appendix A.1.3 for more information on the significance of these results.

1.1.4 Size of polynomially solvable subclasses

To measure the "size" of subclasses such as hidden extended Horn, simple extended Horn, balanced, or SLUR we use a parameterized probabilistic model for generating random formulas and determine over what parameter subspace a randomly generated formula is a member of a particular subclass with probability tending to 1. Comparing regions of the parameter space where formulas likely belong to particular subclasses gives some idea of the size and scope of such subclasses.

The results of Appendix A.1.4, under a standard model known as the variable-length model $\mathcal{R}_{m,n,p}$ (m clauses from $2n$ literals, each occurring in a clause with probability p) are summarized below and illustrated in Figure 1.

1. If $\langle m, n, p\rangle$ is a point in the parameter space where a random formula is not extended Horn with probability tending to 1, then the formula is not balanced with probability tending to 1.

2. For a rather small region of the parameter space, a random formula is Horn with probability tending to 1. For the same region, a random formula is simple extended Horn with probability tending to 1.

3. For all points in the parameter space above the line $pn = \sqrt{n/m}$, a random formula is neither extended Horn nor balanced with probability tending to 1.

4. For virtually all points of the parameter space as depicted in Figure 1, SLUR solves a random formula with probability tending to 1.

5. The results of Appendix A.1.4 are based on a paper that also shows the relationship between these polynomially solvable subclasses and effective algorithms for satisfiability [22].

6. A reviewer of the above paper says: "I found the paper quite interesting because (i) it summarizes well the history of the polynomial average-time analyses and (ii) it also exhibits fundamental problems about the way of doing the analyses and about poly-time algorithms themselves as well."

A possibly more interesting standard model is the fixed-clause-width model $\mathcal{M}_{m,n,k}$ (m clauses, each taken uniformly from all possible k literal clauses from n variables). The probabilistic performance of resolution-based and other algorithms has received quite serious consideration under this model. It is known that Davis-Putnam style algorithms almost always find solutions to random formulas when $m/n < c_1 2^k/k$, c_1 a constant, but even resolution can only build exponential size proofs of unsatisfiability for almost all random formulas when $m/n > c_2 2^k$, c_2 a constant. Additionally, a simple algorithm that reduces a given formula by successively finding and eliminating pure literals (variables that occur only positively or negatively) finds a satisfying assignment for almost all random formulas when $m/n < 1$. We have found the following about special subclasses of SAT.

1. A random formula is a member of at least one of the classes SLUR, q-Horn, extended Horn, etc. with probability tending to 0 if $m/n > c_3/k^2$.

2. Another class of formulas consists of those that can be solved by a polynomial time matching algorithm on a bipartite graph with one set of vertices representing variables, the other set of vertices representing clauses, and edges between variable vertices and clause vertices whenever the corresponding variable is in the corresponding clause. If a matching covering all clause vertices exists, then a satisfying truth assignment follows easily. We have shown that a random formula is a member of this class with probability tending to 1 if $m/n < c_4$, c_4 a constant.

The following are comments on these results.

1. The two results above are surprising taken together. It has been thought that q-Horn [4], and possibly SLUR, are in some sense extremely broad polynomial time subclasses of Satisfiability. However, the matching result above suggests there are broader subclasses.

2. Given the results above, the question is whether there is a polynomial time subclass PS of Satisfiability such that a random formula is a member of PS with probability tending to 1 if $m/n < c_5 2^k/k$ or, better, if $m/n < c_6 2^k$, c_5 and c_6 constants. We know there is a c_5 such that a preponderance of random formulas are easy. We expect to be able to identify such formulas as belonging to a polynomial time subclass of Satisfiability. However, so far, we are not even close to this. Why is this?

3. Our analysis shows that q-Horn, SLUR, and other classes are vulnerable to certain cycles in the following graphs. Given a CNF formula $\mathcal{F}$, construct a graph containing one vertex for each clause in $\mathcal{F}$ and such that an edge exists between two vertices if and only if the two corresponding clauses have a variable in common (regardless of polarity). Although the number of "killer" cycles is low, the probability that one exists tends to 1 about where the probability that a cycle exists tends to 1. This seems to be a feature of the "unbiased" and symmetric distribution used to obtain the results.

1.2 Logic Programming Semantics

The strong theoretical interest in the stable and well-founded semantics needs to be matched by practical implementations for the theoretical research to get significant applications. The difficulty is that the standard algorithm for the well founded semantics is too slow for most practical applications, and the stable semantics is co-NP-complete in the worst case, and only in very special cases are fast implementations known.

Currently, there is active research on more efficient implementations of both logic programming semantics. In the work described here, for a fairly general class of logic programs, we have found a speedup in the worst-case behavior of the standard algorithm for computing the well-founded semantics that breaks what appeared to be a likely lower time bound.

1.2.1 The well-founded semantics algorithm

The results of Appendix A.2.2 are summarized below.

1. The first algorithm improves, albeit by a small margin, on current algorithms for computing the well-founded semantics. It can be applied to all normal propositional logic programs, although the speedups are provable only for the somewhat specialized classes listed there.

2. More importantly, that algorithm shows that the previous quadratic $\mathbf{O}(|\mathcal{A}||\mathcal{P}|)$ bound can be broken in a broad variety of circumstances. Thus it should provide incentive for further research. (We, for example, expect to return to this problem.)

3. We do not yet know how the algorithm will perform on practical examples. In fact, only now are researchers *starting* to gather proposed benchmark programs, and there is no accepted distribution for experimentation with random programs.

4. Normal logic programs, under the stable semantics, provide an especially simple uniform way to represent arbitrary NP problems; for example, reductions of various NP-complete problems to the existence of stable models seems to be simpler than reductions to propositional satisfiability. Thus it is expected to be highly useful as a specification tool.

5. Besides being of interest in its own right, computing the well-founded semantics can be used to help compute the stable semantics. We are continuing experimentation with some non-obvious techniques in this direction, heuristics which we hope will substantially speed up the search for stable models for many logic programs. Thus we expect that any speedup in computing the well-founded semantics will give substantially greater speedups to such a computation of the stable semantics.

6. The standard algorithms for the well-founded semantics are simplifications of various similar algorithms [42, 43, 1, 17, 35]. The algorithm is referred to as the Van Gelder alternating fixed point algorithm since that presentation [42] was the first clean exposition, generalizing ideas of [43]. The quadratic-time of the algorithm is an easy folklore observation.

7. Finally, the section concludes with two theorems from the dissertation of Jennifer Seitzer, a Ph.D. student of John Schlipf. She showed that in some specialized sets of logic programs the well-founded semantics, and sometimes even

the stable semantics, can be computed in linear time. Though these results apply to fairly strongly constrained sets of formulas, they do call attention to structural properties of logic programs that cause NP-completeness. (By comparison, 2-satisfiability is polynomial time, whereas existence of stable models of sets of 2-variable rules is NP-complete.)

1.3 Seminal Survey on Satisfiability Algorithms

A considerable amount of time under ONR sponsorship has been devoted to a survey on Satisfiability algorithms that will appear as part of the 1996 DIMACS volume which constitutes the proceedings of the DIMACS Satisfiability workshop. This project was begun three years ago by Jun Gu and Paul Purdom. John Franco and Ben Wah have recently joined as co-authors. The project will expand to a book on the subject, and an IEEE tutorial.

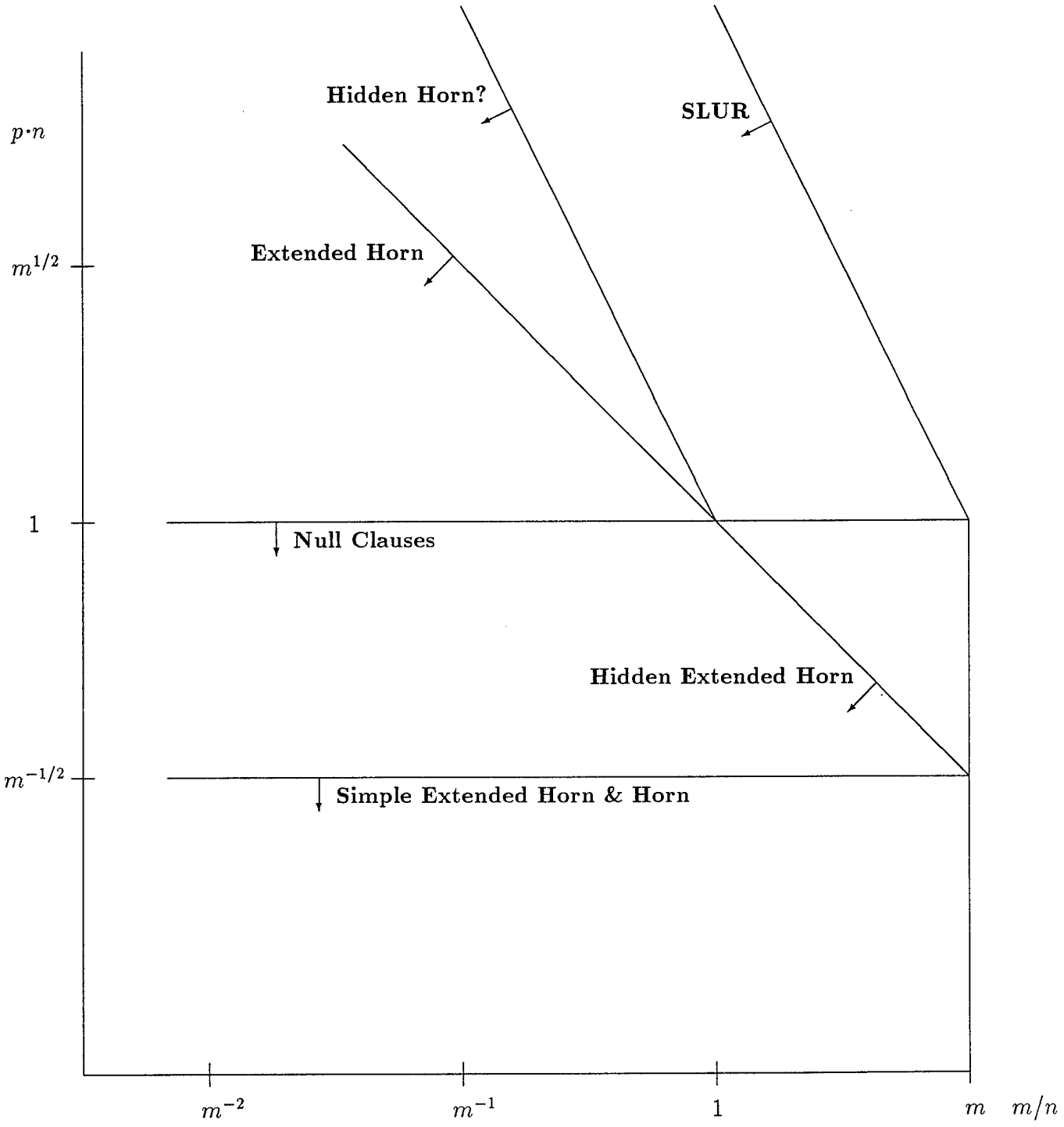

Fig 1: The parameter space of model $\mathcal{R}_{m,n,p}$ partitioned by dominant formula subclasses. Pick a point in the parameter space. Locate the lines with names of subclasses on the side of the line facing the chosen point. A random formula generated with parameters set at the specified point is in the named subclasses with probability tending to 1.

A Details of Results Under ONR Sponsorship

A.1 Polynomial Time Subclasses of Satisfiability

It is well known that CNF (Conjunctive Normal Form) Satisfiability is NP-complete. A CNF Satisfiability formula consists of a set $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ of m subsets (called clauses) of a set $L = \{x_1, \bar{x}_1, x_2, \bar{x}_2, ..., x_n, \bar{x}_n\}$ of n positive and n negative literals. In what follows we assume, unless otherwise stated, that no two complementary literals are in the same subset. The positive literals of L are called variables. Variables are assigned values from the set $\{0, 1\}$ and $x_i = 1$ if and only if $\bar{x}_i = 0$ for all $1 \leq i \leq n$. Given a CNF formula, the question is to determine whether there is an assignment of values to the variables of L such that every clause in $\mathcal{C}$ contains a literal of value 1. If such an assignment exists, the formula is said to be satisfiable. CNF Satisfiability is well studied because, using a transformation of Tseitin, a Satisfiability formula $\mathcal{F}$ in a form other than CNF, even using alternate connectives such as $\rightarrow$, may be efficiently transformed to a CNF formula $\mathcal{C}$ such that $\mathcal{F}$ is satisfiable if and only if $\mathcal{C}$ is satisfiable; often an efficient such transformation in the other direction is unknown.

A focus of considerable research is the characterization of subclasses of CNF Satisfiability that can be solved in polynomial time. The hope is that suitably large, non trivial classes can be found, particularly classes of unsatisfiable formulas.

Several obvious subclasses of Satisfiability are polynomially solvable. For example, a formula containing no more than two occurrences of a literal and its complement [38], or a formula containing at least one negative literal in every clause are easily solved in polynomial time. A more interesting class, due to its relationship with logic programming, is the class of Horn formulas: a formula is Horn if all clauses of two or more literals contain at least one negative literal and at most one positive literal, and all unit clauses contain positive literals.

Extensions to Horn formulas have been proposed. Among the most notable extensions are those developed by way of Linear Programming and are based on the fact that if a constraint matrix satisfies a particular property, then a solution to a LP formulation of CNF Satisfiability either is a 0-1 integer solution or can easily be transformed to a 0-1 integer solution. Two well-known polynomially solvable extensions to Horn formulas have been developed based on this idea.

The class called extended Horn formulas, due to Chandru and Hooker [6], can best be visualized in its equivalent graph theoretic version [37] as follows.

Definition A.1 Let C be a clause and let R be a rooted directed tree with root s and

with edges uniquely labeled with variables in L. Then C is *extended Horn w.r.t. R* if the positive literals of C label a dipath P of R and the set of negative literals in C label an edge-disjoint union[1] of dipaths $Q_1, Q_2, ..., Q_t$ of R with exactly one of the following conditions satisfied:

1. $Q_1, Q_2, ..., Q_t$ start at the root s.
2. $Q_1, Q_2, ..., Q_{t-1}$, (say), start at the root s, and Q_t and P start at a node $q \neq s$.

A CNF formula $\mathcal{C}$ is *extended Horn w.r.t. R* if each clause $C \in \mathcal{C}$ is extended Horn w.r.t. R. A formula is *extended Horn* if it is extended Horn w.r.t. some such rooted directed tree R.

A formula known to be extended Horn can be solved in polynomial time. Unfortunately, the problem of recognizing an extended Horn formula is not known to be polynomially solvable. The following restricted class can be recognized in polynomial time [37].

Definition A.2 Clause C is *simple extended Horn w.r.t. R* if it is extended Horn w.r.t. R and Condition 1 above is satisfied. A CNF formula $\mathcal{C}$ is *simple extended Horn w.r.t. R* if each clause $C \in \mathcal{C}$ is simple extended Horn w.r.t. R. A formula is *simple extended Horn* if it is simple extended Horn w.r.t. some such rooted directed tree R.

The second class of formulas is what we call the class of *balanced formulas* defined by [13, 39].

Definition A.3 Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a CNF formula. Associate with $\mathcal{C}$ a $m \times n$ $(0, \pm 1)$-matrix M as follows: The rows of M are indexed on $\mathcal{C}$ and the columns are indexed on the positive literals of L such that the entry M_{ij} is a $+1$ if $x_j \in C_i$, a -1 if $\bar{x}_j \in C_i$ and a 0 otherwise. Then $\mathcal{C}$ is a *balanced formula* if, in every submatrix of M with exactly two nonzero entries per row and per column, the sum of the entries is a multiple of four.

The recognition problem for balanced formulas is solved in polynomial time. However, recognizing and solving a balanced formula is still a bit tricky.

The Operations Research perspective has successfully produced extensions of classes of polynomially solved formulas, perhaps because it provides a clearer visualization of the nature of satisfiability. In this case, the visualization is a polytope and the question is: Under what conditions does the polytope, if split in half, either

[1] We noted that edge-disjointness is unnecessary

have solutions in both halves and such conditions remain satisfied in both halves, or have no solutions in either half and it is easy to test that? Clearly, the question of satisfiability would be easy on such a polytope. Our contribution involves stepping back from this OR approach to see how this basic idea applies to conventional combinatorial algorithms for Satisfiability. The result is an algorithm that solves a class broader than extended Horn, simple extended Horn, balanced formulas, or even hidden extended Horn. Furthermore, this algorithm works without a recognition step [36].

A.1.1 Single look-ahead unit resolution

Specifically, we apply a variant of the Davis-Putnam-Loveland algorithm to a CNF formula. The variant does not allow backtracking but looks one level ahead in both branches representing both possible assignments to a variable. If a partial assignment cannot be extended in one branch because some clause becomes null after applying unit resolution, the other branch is explored. If the partial assignment can be extended in either branch, one of the two is explored arbitrarily. If both branches lead to null clauses after applying unit resolution, the algorithm terminates with a "don't know." If a satisfying assignment is found, it is returned. If, before the first split, a null clause is generated while applying unit resolution, then "unsatisfiable" is returned. The algorithm SLUR [36] is given as follows:

Algorithm **UCR** ($\mathcal{C}$)
Input: A CNF formula $\mathcal{C}$
Output: A CNF formula $\mathcal{C}'$ without unit clauses

While there is a unit clause $\{l\}$ in $\mathcal{C}$ do the following:
If l is a positive literal set $\mathcal{C} := \{C - \{\bar{l}\} : C \in \mathcal{C}, l \notin C\}$.
Otherwise set $\mathcal{C} := \{C - \{l\} : C \in \mathcal{C}, \bar{l} \notin C\}$.
Output $\mathcal{C}$.
End Algorithm **UCR**

Algorithm SLUR($\mathcal{C}$)
Input: A CNF formula $\mathcal{C}$
Output: A satisfying truth assignment for $\mathcal{C}$, "unsatisfiable," or "give up"

Initialize $t := \emptyset$.
Initialize $\mathcal{C}$:=**UCR**($\mathcal{C}$).
If $\emptyset \in \mathcal{C}$ then output "unsatisfiable" and halt.

While $\mathcal{C}$ is not empty do the following:
Select a variable v appearing as a literal of $\mathcal{C}$.
Set $\mathcal{C}_1$:=**UCR**($\{C - \{v\} : C \in \mathcal{C}, \bar{v} \notin C\}$).
Set $\mathcal{C}_2$:=**UCR**($\{C - \{\bar{v}\} : C \in \mathcal{C}, v \notin C\}$).
If $\emptyset \in \mathcal{C}_1$ and $\emptyset \in \mathcal{C}_2$ then output "give up" and halt.
Otherwise, if $\emptyset \notin \mathcal{C}_1$ set $\mathcal{C} := \mathcal{C}_1$.
Otherwise, set $t := t \cup \{v\}$, and set $\mathcal{C} := \mathcal{C}_2$.
Output $t \cup \{v : v$ was eliminated by **UCR** along the chosen path$\}$.
End Algorithm SLUR

The algorithm works on extended Horn and hidden extended Horn formulas because of the following. Let T denote any Davis-Putnam-Loveland search tree (rooted) for an arbitrary formula $\mathcal{C}$. Each node in T represents a set of clauses closed under unit clause resolution. The leaves correspond to partial truth assignments for which satisfiability or unsatisfiability has been determined. All edges of T are directed away from the root.

Proposition A.1 *If $\mathcal{C}$ is extended Horn, then there is a directed path from every internal node of T to a leaf representing a solution.*

Proposition A.2 *If the root of T is an unsatisfiable leaf (i.e., the empty clause is an element of the root), then $\mathcal{C}$ is unsatisfiable even if $\mathcal{C}$ is not an extended Horn formula.*

The above two propositions plus the fact that SLUR is a generalization of the algorithm used to solve balanced formulas imply the following.

Proposition A.3 *Algorithm SLUR solves $\mathcal{C}$ (never gives up on $\mathcal{C}$) if $\mathcal{C}$ is (i) extended Horn, (ii) hidden extended Horn, (iii) simple extended Horn, or (iv) balanced.*

Although it is easily seen that SLUR can be implemented with quadratic complexity, it is possible, by dove-tailing the two look-aheads and abandoning the computation on one side once the other side has completed, to implement SLUR as a linear time algorithm [39].

The class of problems solved by SLUR without giving up is larger than the hidden extended Horn and balanced classes combined as the following examples show.

Proposition A.4 *Suppose that algorithm SLUR solves formula $\mathcal{C}$ without giving up, and let $\mathcal{C}'$ be a set of clauses that are logical consequences of $\mathcal{C}$. Then algorithm SLUR also solves formula $\mathcal{J} = \mathcal{C} \cup \mathcal{C}'$ without giving up.*

Proposition A.5 *Suppose $\mathcal{C}$ is a formula containing clauses C_1, C_2, C_3, C_4 and x_i, x_j are variables where $\{x_i, x_j\} \subseteq C_1$, $\{x_i, \bar{x}_j\} \subseteq C_2$, $\{\bar{x}_i, x_j\} \subseteq C_3$, and $\{\bar{x}_i, \bar{x}_j\} \subseteq C_4$. Then $\mathcal{C}$ is not hidden extended Horn.*

Example A.6 Let $\mathcal{C}$ be any non-empty formula solved without giving up by SLUR, let C be any clause in $\mathcal{C}$, and let

$$\mathcal{J} = \mathcal{C} \cup \{C \cup \{x_{n+1}, x_{n+2}\}, C \cup \{x_{n+1}, \bar{x}_{n+2}\}, C \cup \{\bar{x}_{n+1}, x_{n+2}\}, C \cup \{\bar{x}_{n+1}, \bar{x}_{n+2}\}\}.$$

Algorithm SLUR solves $\mathcal{J}$ (by Proposition A.4) but $\mathcal{J}$ is not hidden extended Horn (by Proposition A.5).

Example A.7 Let $\mathcal{C}$ be the CNF formula asserting that exactly an even number of variables of L are true. E.g., for $L = \{x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3\}$ we have

$$\mathcal{C} = \{\{\bar{x}_1, x_2, x_3\}, \{x_1, \bar{x}_2, x_3\}, \{x_1, x_2, \bar{x}_3\}, \{\bar{x}_1, \bar{x}_2, \bar{x}_3\}\}.$$

Algorithm SLUR solves $\mathcal{C}$. But $\mathcal{C}$ is not hidden extended Horn (by Proposition A.5). Nor is $\mathcal{C}$ balanced (by inspection of the matrix). Moreover, $\mathcal{C}$ is not equivalent to any proper subset of itself, so $\mathcal{C}$ cannot be constructed from a hidden extended Horn or balanced formula, as in Proposition A.5 and Example A.6.

A.1.2 Single look-ahead with limited backtracking

There are several possible ways to improve SLUR. One way, mentioned above, is to add power to the look-ahead step. As used above, SLUR relies only on the unit clause rule to determine unsatisfiability for some assignment to some variable. Other possibilities abound: for example, just adding a 2-SAT checker enables SLUR to solve numerous formulas that it previously could not.

Algorithm SLUR can also be improved by adding a form of limited backtracking. That is, instead of giving up immediately when both look-aheads produce an empty clause, SLUR can be allowed to continue searching for a satisfying truth assignment from a variable and value that did not previously produce an empty clause. We choose a polynomial p and allow SLUR to backtrack at most $p(|\mathcal{C}|)$ times before giving up if

satisfiability or unsatisfiability has not been determined. Call the resulting algorithm, with any fast look-ahead additions, ISLUR. There seems to be a chance that limited backtracking in ISLUR can overcome a major weakness of SLUR: the inability to solve complex unsatisfiable formulas.

Unfortunately, the performance of ISLUR is disappointing when presented with a "typical" unsatisfiable k-CNF formula [21]. Let $H(\mathcal{C})$ be a function that determines what variable will be assigned values at what point during the execution of ISLUR on k-CNF formula $\mathcal{C}$. Let $V \subset L$ be a subset of literals such that, for $1 \leq i \leq n$, x_i is in V if and only if $\bar{x}_i$ is. Let $\mathcal{C}' \subset \mathcal{C}$ be a subset of subsets of L such that all subsets in $\mathcal{C}'$ contain at least one literal taken from V. The value of $common(V,\mathcal{C}')$ is the total number of positive and negative literals in $\mathcal{C}'$ taken from V. As an example, if

$$\mathcal{C}' = \{\{x_1, x_2, \bar{x}_3\}, \{\bar{x}_2, x_4, \bar{x}_5\}, \{x_3, \bar{x}_5, x_6\}, \{\bar{x}_4, \bar{x}_6, \bar{x}_7\}\}$$

and

$$V = \{x_3, \bar{x}_3, x_4, \bar{x}_4, x_5, \bar{x}_5\}$$

then $common(V,\mathcal{C}') = 6$.

Definition A.4 Fix $0 < \epsilon < 1$ and $0 < \gamma < 1$. Call a k-CNF formula $\mathcal{C}$ with the following property *sparse*: for all subsets V and $\mathcal{C}' \subset \mathcal{C}$ such that $|\mathcal{C}'| < m^{\epsilon}$ and at least one literal of every subset $C \in \mathcal{C}'$ is in V, $common(V,\mathcal{C}') < k(1-\gamma)|\mathcal{C}'|$.

Proposition A.8 *For all H functions, ISLUR gives up on unsatisfiable, sparse k-CNF formulas, $k \geq 3$, with the number of occurrences of any particular variable bounded from above by $m^{1/4}$.*

A.1.3 Pure implicational formulas

Falsifiability for pure implicational formulas (only the implies operator $\rightarrow$ is allowed) is surprisingly rich in providing hierarchies of subclasses of progressively increasing hardness, eventually reaching subclasses that are NP-complete [27]. This richness may be extended to CNF formulas.

Let $A = \{a_1, a_2, ..., a_n\}$ be a set of n atoms.

Definition A.5 The set of *pure implicational formulas* is defined by the following recursion:

1. Each atom $a \in A$ is a pure implicational formula.

2. If F_1, F_2 are pure implicational formulas, then so is $(F_1 \rightarrow F_2)$.

When parentheses are omitted, they are always assumed to associate to the right.

In what follows we assume that parentheses are omited from formulas when possible. Then every pure implicational formula can be written as $F_1 \rightarrow F_2 \rightarrow ... \rightarrow F_k \rightarrow z$ where all F_i, $1 \leq i \leq k$, are pure implicational formulas and z is an atom. The role of z is special when determining falsifiability for a pure implicational formula (does there exist a truth assignment that falsifies F?) as shown below.

Let F be a pure implicational formula such that the number of occurrences of every atom, except z, in F is no greater than two. Then, a recent result of Heusch [27] says that the problem of determining falsifiability for F is NP-complete. However, Heusch also shows that, if the number of occurrences of z in F is no greater than k, and all other atoms occur at most twice, then the falsifiability of F can be determined in time bounded from above by $|F|^k$. This result hints that there may be an infinite hierarchy $S_1 \subset S_2...$, where S_k is the set of pure implicational formulas with at most k occurrences of z and at most two occurrences of all other atoms, such that determining the falsifiability of formulas in S_k is increasingly harder with increasing k.

In our work we generalized Heusch's question somewhat. Let us call a formula *implicational* if its only connectives are $\rightarrow$ and $\mathbf{f}$, a constant proposition denoting **false**. (The recursive definition is the same as the one for pure implicational formulas except that $\mathbf{f}$ is also an implicattional formula.) It is well-known that every propositional formula is tautologically equivalent to an implicational formula. We studied both satisfiability and falsifiability of implicational formulas where each propositional variable occurs at most twice and $\mathbf{f}$ occurs at most k times.

Heusch observed that in the most difficult case for testing falsifiability for pure implicational formulas, the proposition letter z must be false. Accordingly, the difficult cases of his problem reduce to ours via a substitution of $\mathbf{f}$ for z. We also noted that testing falsifiability for implicational formulas and testing satisfiable formulas approximately the same complexity (up to at most an approximately linear factor).

We have investigated an algorithm **IMP** for determining satisfiability for implicational formulas that has the following characteristic.

Proposition A.9 *Given an implicational formula F with each atom (proposition variable) occurring in F at most twice, and with $\mathbf{f}$ occurring at most k times, algorithm* **IMP** *determines satisfiability of F in $O(k^k|F|)$.*

The algorithm is much too complex to be given here. The reader is referred to [20] for details.

The pure implicational logic results above are important for two reasons. First, we have shown that Heusch's hierarchy is fixed-parameter-tractable and as far as we know this is the first such hierarchy result for Satisfiability problems. Levels of a fixed-parameter tractable hierarchy are distinguished by a parameter k which in some sense measures the density of an instance; any algorithm to solve instances of length n for fixed k takes time bounded by some $c_k(1 + n^{e_k})$ where the value of e_k holds steady with increasing k [15]. Second, study of Heusch's hierarchy may reveal the nature of easy and hard problems and we believe we have found, roughly, the place in the hierarchy where the transition may occur, if at all. Regarding this point, it is interesting to note that the expressibility of *pure implicational* formulas is quite low and one wonders how such formulas can be rich enough to be hard. In any case, the lack of expressibility may be exploited to assist complexity investigations in other hierarchies.

It is easy to see that Heusch's pure implicational formulas carry over to CNF formulas quite naturally and all the results above apply to that CNF hierarchy. Such a CNF hierarchy might be useful in answering the following question. Why should the hierarchies of Gallo and Scutella [24], Dalal [14], or Kleine Büning [29] have $O(n^k)$ complexity when a complexity of $O(2^k n^2)$, say, is not inconsistent with any developed theory?

A.1.4 How large are these special subclasses?

We would like some measure of the "size" of subclasses such as hidden extended Horn, simple extended Horn, balanced, or SLUR. One way to do this is define a parameterized probabilistic model for generating random formulas and determine over what parameter subspace is a randomly generated formula a member of a particular subclass with probability tending to 1. Comparing regions of the parameter space where formulas likely belong to particular subclasses gives some idea of the size and scope of such subclasses. Such comparisons are found in [22] and summarized below.

We use the following probabilistic model, denoted $\mathcal{R}_{m,n,p}$, for generating random CNF formulas.

Definition A.6 A random formula generated according to $\mathcal{R}_{m,n,p}$ contains m clauses constructed independently from L as follows. For each clause $C \in \mathcal{C}'$, for all $l \in L$, admit l in C with probability p, independently of other literals and clause constructions.

We chose model $\mathcal{R}_{m,n,p}$ because it is one of the most popular models for analyzing the average-case performance of SAT algorithms and because it supports many of the classes we are interested in. A drawback of this model is that the probability that a null clause exists in a random instance tends to 1 if $pn < \log(m)$ (that is, the average number of literals in a clause is $O(\log(m))$).

The model $\mathcal{R}_{m,n,p}$, as well as most other models intending to be "unbiased," is symmetric and therefore does not favor non-symmetric subclasses such as Horn formulas. Thus, Horn formulas are generated with probability tending to 1 over only a small portion of the parameter space m, n, p. Surprisingly, this is true for simple extended Horn formulas as well. Let $\mathcal{C}_{m,n,p}$ be a CNF formula randomly generated from $\mathcal{R}_{m,n,p}$. We have the following.

Proposition A.10 *If $pn < 1/m^{1/2+\epsilon}$, $\epsilon > 0$, then $\mathcal{C}_{m,n,p}$ is a Horn formula with probability tending to 1.*

Proposition A.11 *If $pn > 1/m^{1/2+\epsilon}$, $\epsilon > 0$, then $\mathcal{C}_{m,n,p}$ is not a simple extended Horn formula with probability tending to 1.*

The parameter space representing all combinations of values of m, n, p is visualized in Figure 1. The vertical axis is pn, the average number of literals in a clause, and the horizontal axis is m/n, the ratio of clauses to variables. The region where formulas are Horn or simple extended Horn with high probability is below the horizontal line labeled "Simple Extended Horn & Horn." It may be puzzling that this region is contained in the half plane $pn < \log(m)$ (shown in Figure 1 below the horizontal line labeled "Null Clauses") where the probability that a null clause exists in $\mathcal{C}_{m,n,p}$ tends to 1. Actually, the number of null clauses in $\mathcal{C}_{m,n,p}$ for this half plane is approximately $m(1-2pn)$ which means about $2pmn$ clauses are not empty. Thus, when $pn = m^{-1/2}$, the number of non-null clauses is about $m^{1/2}$ and these clauses are Horn and simple extended Horn with high probability.

Our knowledge of stratified logic programs has influenced our understanding of extended Horn formulas to reveal a surprising connection to "cycles" in $\mathcal{C}_{m,n,p}$.

Definition A.7 A *cycle* in a CNF formula C is an ordered collection of clauses $\{C_0, C_1, ..., C_{k-1}\}$, $k \geq 2$, such that, for all $i = 0, .., k-1$, there is a variable common to C_i and $C_{i+1 \bmod k}$ (disregarding the variable's stature as a positive or negative literal).

The relationship between cycles and extended Horn formulas that we exploit is the following.

Proposition A.12 *If a CNF formula $\mathcal{C}$ does not have a cycle, then $\mathcal{C}$ is an extended Horn formula.*

The following result shows where cycles are not common.

Proposition A.13 *If $pn < n^{1-\epsilon}/m$ and $m/n \geq 1$ or $pn < \sqrt{n^{1-\epsilon}/m}$ and $m/n < 1$, $1 > \epsilon > 0$, then there exists no cycle in $\mathcal{C}_{m,n,p}$ with probability tending to 1.*

Since acyclic formulas are extended Horn, we have a lower bound on the region generating predominantly extended Horn sets. An upper bound may be found by considering a property not possible in any extended Horn set. An instance $\mathcal{C}$ is not extended Horn if there exists in $\mathcal{C}$ three clauses, C_1, C_2, C_3, and three positive literals l_1, l_2, l_3 such that C_1 has l_1 and l_2 but not l_3, C_2 has l_1 and l_3 but not l_2, and C_3 has l_2 and l_3 but not l_1. In this case, $\mathcal{C}$ is said to have a triplet of non-extendible Horn clauses, or a triplet for short. The following shows where formulas are not extended Horn, in probability.

Proposition A.14 *If $pn > \sqrt{n^{1+\epsilon}/m}$, $\epsilon > 0$, then the average number of triplets in $\mathcal{C}_{m,n,p}$ grows without bound as m and n tend to ∞.*

Thus, the region of Figure 1 below the diagonal lines labeled "Extended Horn" and "Hidden Extended Horn" seem to be where random formulas usually are extended Horn. Surprisingly, this region approximately coincides with the region for which no cycles exist with high probability. The reason is that, although most cycles are tolerated by extended Horn formulas, there are a few that are not. Because of the symmetry of $\mathcal{R}_{m,n,p}$, these few begin to be generated, in probability, when any cycle is. That is, $\mathcal{R}_{m,n,p}$ is unbiased and does not distinguish between "good" and "bad" cycles and when some good cycles are generated, then so are some bad cycles generated.

We can find the limits of balanced formulas by computing the average number of 2×2 submatrices that contain all 1's and do not sum to a multiple of 4. This average is roughly $\binom{m}{2}\binom{n}{2}p^4 = O(m^2n^2p^4)$. Formulas are balanced, in probability, only if this quantity tends to 0. Therefore,

Proposition A.15 *If* $pn > \sqrt{n^{1+\epsilon}/m}$, $\epsilon > 0$, *then* $C_{m,n,p}$ *is not a balanced formula with probability tending to 1.*

This means that balanced formulas are not frequently occurring where extended Horn formulas are not frequently occurring (that is, above the diagonal lines labeled "Extended Horn" and "Hidden Extended Horn").

The limits of the Hidden Horn class are obtained from the following.

Proposition A.16 *The probability that* $C_{m,n,p}$ *is Hidden Horn tends to 0 if* $pn \to 0$, *and* $pn > \sqrt{n^{1+\epsilon}/m}$, $\epsilon > 0$, *or if* $pn \to \infty$, *and* $pn > n^{1+\epsilon}/m$, $\epsilon > 0$.

This region is above the diagonal lines labeled "Hidden Horn?" and "Hidden Extended Horn" in Figure 1.

Finally, due to results in [18, 19] we can say the following.

Proposition A.17 *If* $p < 1$, *then with probability tending to 1,* $C_{m,n,p}$ *can be solved by SLUR without giving up.*

The region in which SLUR performs well, in probability, is below the diagonal line of Figure 1 labeled "SLUR."

A.2 Logic Programming Semantics

Logic programming began as the study of Horn clauses. However, the lack of natural expressibility of Horn clauses led to the study of various generalizations, particularly the generalization called *normal logic programming rules*, rules of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_k$$

where α is a positive literal and $\beta_1, \ldots, \beta_k$ are literals. During the 1980's a great deal of work went into finding the "right" semantics for such rules: semantics that captured the intuitions of negation as failure (e.g., as in PROLOG, such as the asymmetry between head and body of a rule) but were fairly logically clean (unlike PROLOG's *ad-hoc* patch to add in negative subgoals). Although research continues on finding the "right" semantics, two semantics have come into widespread favor, the Stable semantics [25, 26] and the Well-Founded semantics [43].

The hope is that these semantics, if they can be efficiently implemented, will restore the separation between logic and control that PROLOG promised but, in the end, abandoned. In fact, by now this research has matured far enough that a second generation of languages is beginning to be demanded.[2] A major difficulty in this implementation is simply the computational complexity of the problems and of the known algorithms. We limit attention here to propositional logic programs.

It is known that computing the stable semantics of a logic program is worst-case co-NP-complete [31]. Thus research on implementing the stable semantics has focused on finding methods that are reasonably efficient in a broad variety of circumstances. The standard algorithm for computing the well-founded semantics, on the other hand, is worst-case quadratic-time. Considering that the logic program whose well-founded partial model we need to find may be derived by instantiating a logic program over a large deductive database, quadratic time can be exceedingly expensive. A faster algorithm for computing the well-founded semantics, besides being useful in its own right, would be expected also to lead to faster algorithms for computing the stable semantics.

A.2.1 Definitions of the well-founded and stable semantics

The definitions of the well-founded and stable semantics in [43, 25, 26] discuss how these semantics are natural for logic programming. Here we give instead a characterization in terms of directed hypergraphs, one that we created (a small modification of

[2]The issues here are unrelated to the development of constraint logic programming languages.

the definitions in [1, 17, 35, 42, 43]) in order to provide a graph-theoretic view upon which to build an algorithm.

We considered a propositional logic program $\mathcal{P}$ as a directed hypergraph with edges labeled with DNF formulas (containing only negative literals). First, we modified the program $\mathcal{P}$ by adding a new proposition letter s, and for each rule with no positive literals in its body, we added s to the body. We also add a single rule $s \leftarrow$ (i.e., an assertion that s is true) to $\mathcal{P}$. This does not change the well-founded or stable semantics of $\mathcal{P}$ (except that it provides a new atom s always inferred to be true).

Now we convert the problem of finding the stable and well-founded semantics to hypergraph problems. The vertices of the hypergraph are the proposition letters of this altered $\mathcal{P}$. We treat the rule $s \leftarrow$ separately, identifying s as the *source* in the hypergraph. For each other rule

$$a \leftarrow b_1, b_2, \ldots, b_i, \neg c_1, \neg c_2, \ldots, \neg c_j$$

we create a directed hypergraph edge from $\{b_1, \ldots, b_k\}$ to a, labeled $\neg c_1 \wedge \neg c_2 \wedge \ldots \wedge \neg c_j$. Call this hypergraph $\mathcal{H}$.

The well-founded semantics is defined in terms of a 4-valued logic, with truth values t (true), $\bot$ (unknown), f (false), and $\top$ (contradiction). The value $\top$ will not actually arise here, but including it simplifies the mathematics. The truth tables for "and," "or," and "not" are quite intuitive if the value $\top$ is ignored:

$\wedge$	t	$\bot$	f	$\top$
t	t	$\bot$	f	$\top$
$\bot$	$\bot$	$\bot$	f	f
f	f	f	f	f
$\top$	$\top$	f	f	$\top$

$\vee$	t	$\bot$	f	$\top$
t	t	t	t	t
$\bot$	t	$\bot$	$\bot$	t
f	f	$\bot$	f	$\top$
$\top$	t	t	$\top$	$\top$

$\neg$	
t	f
$\bot$	$\bot$
f	t
$\top$	$\top$

With just Horn clauses, in the standard van-Emden Kowalski semantics [41], t means accessible from source s in $\mathcal{H}$, and f means inaccessible. For normal rules a more complicated definition is used.

Definition A.8 *Let τ be a 4-valued truth assignment on the vertices of $\mathcal{H}$.*

- *Let $\underline{\mathcal{H}}$ be the set of hyperedges of $\mathcal{H}$ whose labels evaluate to t or $\top$ under τ. Let $\overline{\mathcal{H}}$ be the set of hyperedges of $\mathcal{H}$ whose labels evaluate to t or $\bot$ under τ.* (The intuition is that $\underline{\mathcal{H}}$ is an under-approximation to the set of rules we'll finally use, and $\overline{\mathcal{H}}$ is an over-approximation.)

- *Define* $T = \{v \in \mathcal{H} : v$ *is accessible from* s *in* $\underline{\mathcal{H}}\}$ *and* $F = \{v \in \mathcal{H} : v$ *is not accessible from* s *in* $\overline{\mathcal{H}}\}$. (The intuition is that atoms in T are definitely provable, i.e., accessible from s, and the atoms in F are definitely unprovable, i.e., inaccessible.)

- *Define*

$$\tau(v) = \begin{cases} t & \text{if } v \in T \text{ and } v \notin F \\ f & \text{if } v \notin T \text{ and } v \in F \\ \bot & \text{if } v \notin T \text{ and } v \notin F \\ \top & \text{if } v \in T \text{ and } v \in F \end{cases}$$

- *If* $\tau = \tau'$ *then* τ *is a* 4-valued-stable *model of* $\mathcal{P}$.

Every logic program has a 4-valued-stable model τ *where* τ *never takes on value* $\top$ *and, among 4-valued stable models, the set of elements with truth value* $\bot$ *is maximal; this is the* well-founded (partial) model *of* $\mathcal{P}$.

If a 4-valued stable model is 2-valued, *i.e.,* τ *takes on only the values* t *and* f, *then* τ *is a* stable model. □

The standard calculation of the well-founded semantics is essentially the following. The proof that it finds the well-founded model is a corollary of the Tarski-Knaster theorem on fixed points of monotonic operators.

Algorithm A.1 *Let* τ_0 *be the constant* $\bot$ *interpretation. For each integer* $n \geq 0$, $\tau_{n+1} = \tau'_n$. *For* $\mathcal{P}$ *a finite propositional program, this sequence must reach a fixed point; that fixed point is the well-founded partial model.*

A.2.2 Computing the well-founded semantics faster

We investigated speeding up the computation of the well-founded semantics. Speedups had previously been found for programs obeying various strong restrictions on possible kinds of cycles. We found one other:

Proposition A.18 *Fix a natural number* k. *Let* $\mathbf{P}_k$ *be the set of all normal logic programs* $\mathcal{P}$ *for which there exists a set* A *of* k *atoms such that, if all rules with heads in* A *are deleted from* $\mathcal{P}$, *the* positive dependency *relation of the remaining rules is acyclic. There is an algorithm for constructing the well-founded semantics for all programs in* $\mathbf{P}_k$ *in time linear in the size of the program.*

But no more general speedups (speedups of worst-case behavior) were known. In fact, graph theoretic evidence [34] suggested that the standard algorithm might be optimal. To wit, during the course of running the algorithm above, the graph $\underline{\mathcal{H}}$ grows monotonically and the graph $\overline{\mathcal{H}}$ shrinks monotonically until the fixed point is reached. An obvious speedup is to maintain information about accessibility in both directed hypergraphs dynamically. Now it is easy to maintain accessibility information efficiently in a growing directed hypergraph, but the obvious generalization does not work for a shrinking hypergraph. Reif's work suggests that, even for a shrinking digraph, maintaining accessibility seems generally to take quadratic time. This makes the goal of speeding up the computation of the well-founded semantics sound implausible.

Thus if there is to be any faster algorithm of the well-founded semantics it must make use of special features of how the graph $\overline{\mathcal{H}}$ shrinks. Our speedup made use of one additional feature of the standard well-founded semantics algorithm: the deletions do not continue indefinitely; rather, as soon as a fixed point is reached, the algorithm halts.

Our speedup works for a very broad class of logic programs, with less stringent (and very different) syntactic constraints than previously studied classes. First, it requires that the size $|\mathcal{P}|$ of the program must be at least on the order of the square of the number $|\mathcal{A}|$ of proposition letters. Second, it requires that either (1) each rule of the program $\mathcal{P}$ have at most two positive subgoals — any number of negative subgoals is allowed — or that (2) each possible directed edge on the proposition letters be a directed subedge of fewer than $\mu = \sqrt[3]{|\mathcal{H}|/|\mathcal{A}|}$ hyperedges of the original $\overline{\mathcal{H}}$. Similar but weaker speedups hold if the bound μ is increased. Note that the bound is trivially satisfied if $\overline{\mathcal{H}}$ is a digraph, i.e., each rule of $\mathcal{P}$ has at most one positive subgoal.

The basic technique of the algorithm involves, before searching $\overline{\mathcal{H}}$ for inaccessible nodes, doing approximate searches. First it searches for vertices with in-degree 0 (in-degrees can be maintained efficiently); if any are found, it adds these to set F and returns to the basic algorithm loop. Next, it searches an approximation $\mathcal{H}'$ of $\overline{\mathcal{H}}$, where $\mathcal{H}'$ is constructed so that inaccessibility in $\mathcal{H}'$ guarantees inaccessibility in $\overline{\mathcal{H}}$. This approximation can be maintained efficiently as the algorithm runs. Approximation $\mathcal{H}'$ is smaller than $\overline{\mathcal{H}}$, so searching progresses faster. If inaccessible nodes are found, it adds these to set F and returns to the basic algorithm loop.

Only if no inaccessible vertices are found does the algorithm go on to searching all of $\overline{\mathcal{H}}$. If it does happen to search all of $\overline{\mathcal{H}}$, then either it finds that a fixed point is reached (and thus that the algorithm has terminated, giving a one-time linear cost) or else, as can be shown by combinatorial arguments, many inaccessible vertices are

found, reducing the per-vertex cost of the search.

The approximation $\mathcal{H}'$ is defined as follows for the case of the 2 positive subgoal per rule limitation: First, for every possible directed digraph edge e on the set of vertices, if e is a directed subedge of $\geq \mu = \sqrt[3]{|\mathcal{H}|/|\mathcal{A}|}$ hyperedges of $\overline{\mathcal{H}}$, replace all those hyperedges with the single directed edge e. Second, for each vertex a which is the head of $\geq \sqrt{\mu|\mathcal{H}||\mathcal{A}|}$ hyperedges of the first approximation, replace all those subedges with the single directed edge from s to a. Clearly, accessibility in $\overline{\mathcal{H}}$ implies accessibility in $\mathcal{H}'$.

Proposition A.19 *For any propositional normal logic program $\mathcal{P}$ obeying the syntactic restrictions above, our algorithm computes the well-founded partial model for $\mathcal{P}$ in time*

$$\mathbf{O}(|\mathcal{P}| + |\mathcal{A}|^2 + |\mathcal{A}|^{\frac{4}{3}}|\mathcal{P}|^{\frac{4}{3}}).$$

A.2.3 Easily Computed Special Cases in Logic Programming

In her Ph.D. dissertation under the supervision of John Schlipf, Jennifer Seitzer studied a number of special cases of logic programs under which computing well-founded and/or stable semantics can be done (relatively) quickly. A joint paper by Seitzer and Schlipf will be presented at the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning in Schloß Dagstuhl, Germany, in July 1997. The thrust of the paper was to investigate how severely limiting the number of times a variable can appear in the head or the body of a rule (and thus severely limiting the occurrence of cycles in the dependency relation of the program) simplifies the computation of the well-founded semantics and and the search for stable models. We include one pair of results below:

Recall that computation of the well-founded partial model of a normal logic program can be done in quadratic time, but it is not known whether a faster algorithm works in general; determining whether a normal logic program has a stable model is **NP**-complete.

A normal propositional logic program $\mathcal{P}$ is *uni-rule* if no two rules have the same head. The authors show that:

1. There is a linear-time algorithm which, given a uni-rule normal propositional program $\mathcal{P}$, computes its well-founded partial model.

2. Determining whether a uni-rule normal propositional program $\mathcal{P}$ has a stable model is **NP**-complete.

The second result can be taken to show that uni-rule programs, although a simple class of logic programs, definitely have non-trivial expressive power; this in turn shows that the speed-up given in the first result is significant.

A.3 Professional Activities and Invitations

A.3.1 Conferences & workshops

1. 3rd International Symposium on Mathematics and Artificial Intelligence. Ft. Lauderdale, Florida. January, 1994. Session chair and invited talk "Toward a good algorithm for determining unsatisfiability efficiently, in probability."

2. 15th International Symposium on Mathematical Programming. Ann Arbor, Michigan. August, 1994. Session chair and invited talk "Average case results for satisfiability under the random-clause-width model."

3. Logic and Complexity Theory. Hosted by Daniel Leivant with partial funding from the ONR. Indianapolis, Indiana. October, 1994. Invited participants.

4. Carnegie Mellon University Seminar. December, 1994. Invited talk "Probabilistic Analysis of Satisfiability Algorithms."

5. Timberline conference on the interface between OR and AI. Portland Oregon. May, 1995. "On finding solutions to extended Horn sets."

6. 3rd International Conference on Logic Programming and Nonmonotonic Reasoning. Lexington, Kentucky. June, 1995. "Computing the well-founded semantics faster."

7. Boolean mini-symposium. Jerusalem, Israel. July, 1995. Invited talk "On finding solutions to extended Horn and balanced formulas."

8. 14th European Symposium on Operations Research. Hebrew University, Jerusalem, Israel. July, 1995. Invited talk "Computing the well-founded semantics faster."

9. 7th ACM Symposium on Parallel Architectures and Algorithms. University of California, Santa Barbara. July, 1995. "On testing the consecutive-ones property in parallel."

10. INFORMS annual meeting. New Orleans, Louisiana. October, 1995. Invited talk "Computing the well-founded semantics faster."

11. 4th International Symposium on Mathematics and Artificial Intelligence. Ft. Lauderdale, Florida. January, 1996. Invited talk "An improved algorithm for determining the falsifiability of pure implicational logic."

12. DIMACS Workshop on Satisfiability. Rutgers University, New Brunswick, New Jersey. April, 1996. Invited talk "Special polynomial time subclasses of Satisfiability."

13. Siena Workshop on Satisfiability. University of Siena, Siena, Italy. May, 1996. "An algorithm for the class of pure implicational formulas."

14. Dagstuhl Seminar # 9627 on Disjunctive Logic Programming and Databases: Nonmonotonic Aspects. Schloß Dagstuhl, Saarland, Germany. July, 1996. Invited talk (TBA).

A.3.2 Workshop organized

Workshop on Satisfiability. Organized by H.K. Büning, J. Franco, G. Gallo, E. Speckenmeyer at Siena, Italy. May, 1996.

A.3.3 Journal articles (including refereed proceedings)

1. J. Franco and R. Swaminathan. To appear. Average case results for satisfiability algorithms under the random-clause-width model. *Annals of Mathematics and Artificial Intelligence* (invited).

2. J. Schlipf, F. Annexstein, J. Franco, R. Swaminathan. 1995. On finding solutions to extended Horn sets. *Information Processing Letters* **54**, pp. 133–137.

3. J. Franco and R. Swaminathan. To appear. Toward a good algorithm for determining the unsatisfiability of propositional formulas. *Journal of Global Optimization* (invited).

4. K. Berman, J. Schlipf, and J. Franco. 1995. Computing the well-founded semantics faster. *Proc. 3rd International Conference on Logic Programming and Non-Monotonic Reasoning. Lecture Notes in Artificial Intelligence* **#928**, A. Nerode, V. Marek, and M. Truszczyński, Editors. pp. 113–126.

5. F. Annexstein and R. Swaminathan. 1995. *Proc. 7th ACM Symposium on Parallel Architectures and Algorithms* (submitted to *Journal of Computer and System Sciences*).

6. R. Swaminathan, D. Giriraj, and D. Bhatia. 1995. The pagenumber of the class of bandwidth-k graphs is $k-1$. *Information Processing Letters* **55**, pp. 71–74.

7. J. Gu, P. Purdom, J. Franco, and B. Wah. Algorithms for the Satisfiability (SAT) problem: a survey. To appear in a *DIMACS* volume consisting of the proceedings of the "Workshop on Satisfiability," March, 1996.

8. J. Franco. Relative Size of Certain Polynomial Time Solvable Subclasses of Satisfiability. To appear in a *DIMACS* volume consisting of the proceedings of the "Workshop on Satisfiability," March, 1996.

9. J. Franco, J. Goldsmith, J. Schlipf, E. Speckenmeyer, and R. Swaminathan. An algorithm for determining falsifiability for a special class of pure implicational formulas. Submitted to *Discrete Applied Mathematics* special issue devoted to the Siena Workshop on Satisfiability. J. Franco, G. Gallo, H.K. Büning, E. Speckenmeyer, and C. Spera, editors.

10. J. W. Rosenthal, J. M. Plotkin, and J. Franco. The Probability of Pure Literals. Submitted.

11. J. Seitzer and J. Schlipf. Affordable classes of normal logic programs. To appear in the *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, 1997.

A.3.4 Popular literature

1. J. Franco. Network servers and Java. Invited submission to *IEEE Potentials.*

2. J. Franco. Coping with problems computers can't solve. Invited submission to *IEEE Potentials.*

A.3.5 Other relevant journal article

K. Berman, J. Franco, and J. Schlipf. 1995. Unique satisfiability for Horn sets can be solved in nearly linear time. *Discrete Applied Mathematics* **60**, pp. 77–91.

A.3.6 Articles in progress

1. J. Franco, J. Schlipf, and R. Swaminathan. Improved single look-ahead unit resolution.

2. J. Franco, and A. Van Gelder. On the size of certain polynomial time subclasses of Satisfiability.

3. V. Marek, J. Schlipf, and M. Truszczyński. The well-founded semantics for database revision programs.

A.3.7 Relevant visits

1. John Hooker, GSIA, Carnegie Mellon University. John Franco visited CMU in December, 1994.

2. Ewald Speckenmeyer, Computer Science, University of Cologne, Germany. August 15 - September 9, 1995. Prof. Speckenmeyer visited the Computer Science Theory Group at the University of Cincinnati.

3. Peter Hammer, RUTCOR, Rutgers University. John Franco visited RUTCOR for eight days in August, 1996.

A.3.8 External Ph.D. thesis committee

John Franco. RUTCOR, Rutgers University. May, 1995. Ondřej Čepek, "Structural Properties and Minimization of Horn Boolean Functions."

A.3.9 Editing

The proceedings of the Siena Workshop on Satisfiability will be published as a special issue of *Discrete Applied Mathematics*. Editors will be the workshop organizers.

A.3.10 Future projects

The following invited projects are expected to be initiated over the next two years.

1. Book on Satisfiability Algorithms, Cambridge University Press. Co-authors are Jun Gu, Paul Purdom, John Franco, Ben Wah. This will be a much expanded version of the survey paper by the same authors.

2. IEEE Tutorial on Satisfiability. Same co-authors as above.

3. Article for special issue of *Communication of the ACM* devoted to Satisfiability. Co-authors yet to be determined.

4. Monograph on Satisfiability for a special SIAM series edited by Peter Hammer. Exact topic to be determined.

A.4 Collaboration Under ONR Sponsorship

Several colleagues and students have contributed to results reported here and continue to assist the principle investigators on new and unreported results. We acknowledge such contributions here. **Kenneth Berman** was a joint researcher on computing the well-founded semantics faster. We are currently writing a joint paper with **Victor Marek** and **Mirosław Truszczyński** of the University of Kentucky on well-founded "revision semantics," a semantics for maintaining database dependencies during database updates. The work on implicational formulas was inspired by and joint with **Ewald Speckenmeyer** of the University of Köln, Germany. **Judy Goldsmith** of the University of Kentucky also contributed significantly to this work, particularly making us aware of the literature on fixed-parameter-tractable hierarchies. Nearly all Satisfiability results were obtained in collaboration with **R. Swaminathan**. Moreover, R. Swaminathan, while funded, made invaluable contributions to two major works with Fred Annexstein on parallelizing an algorithm for determining the consecutive ones property of matrices and with D. Giriraj and D. Bhatia on showing that the pagenumber of the class of bandwidth-k graphs is $k-1$. The results on single look-ahead unit resolution were obtained with **Fred Annexstein**. The observation that single look-ahead unit resolution can be achieved in linear time is due to **Klaus Trümper** of the University of Texas, Dallas. Discussions with **Endre Boros** of Rutgers University and **John Hooker** of Carnegie Mellon University about single look-ahead unit resolution were helpful in seeing our results from a different perspective. Probabilistic results pertaining to the size of certain polynomial time solvable subclasses of SAT, under the fixed-width model, are joint with **Allen Van Gelder** of the University of California, Santa Cruz. In particular, Allen suggested an analysis of the matching algorithm. Students **Jennifer Seitzer** and **Chris Giannella** have experimented with unusual methods for determining stable semantics. Student Chris Giannella has also run experiments to help determine conditions under which algorithms exhibit a "threshold" property. Student **Eric Luczaj** is exploring polynomial subclasses of Satisfiability. Student **Bryan Bayley** has experimented with an algorithm for verifying unsatisfiability. The following are among those who participated in our Workshop on Satisfiability in Siena in May, 1996: **Endre Boros**, **Peter Hammer**, **David Mitchell**, **John Schlipf**, **Allen Van Gelder**, and **Jinchang Wang**.

References

[1] C. Baral, and V. S. Subrahmanian. 1991. Dualities between alternative semantics for logic programming and non-monotonic reasoning. *Logic Programming and Non-monotonic Reasoning: Proc. of the First International Workshop*, A. Nerode, W. Marek, and V. S. Subrahmanian, editors. MIT Press, Cambridge, Massachusetts, pp. 69–87.

[2] Bollobás, B., and and A. G. Thomason. "Threshold Functions." *Combinatorica* 7, 1987, pp. 35-38.

[3] E. Boros. 1995. Personal Communication.

[4] E. Boros, Y. Crama, P. L. Hammer, and M. Saks. A complexity index for satisfiability problems. *SIAM Journal on Computing* 23:45–49, 1994.

[5] A. Broder, A. M. Frieze, and E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. *Proceedings of the 4th annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, 1993, pp. 322-330.

[6] V. Chandru and J. N. Hooker. 1991. Extended Horn sets in propositional logic. *J. ACM* **38**, pp. 205–221.

[7] M. Chao and J. Franco. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM J. Comput.*, 15:1106-1118, 1986.

[8] M. Chao and J. Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristic for the k-satisfiability problem. *Information Sciences*, 51:289-314, 1990.

[9] Chen, W., and D. S. Warren. "Domputation of stable models and its integration with logical query processing." Technical report. Available by ftp from ftp.ms.uky.edu:pub/lpnmr,chen2.ps. 1994.

[10] Cholewinski, P., V. W. Marek, and M. Truszczynski. "Default Reasoning System DeReS." *Proc. of Knowledge Representation* 1996, to appear.

[11] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. ACM* 35:759–768, 1988.

[12] V. Chvatal and B. Reed. Mick gets some (the odds are on his side). *Proceedings of the 33rd annual Symposium on Foundations of Computer Science*, IEEE, Los Alamitos, California, 1992.

[13] M. Conforti and G. Cornuéjols. 1992. A class of logical inference problems solvable by linear programming. *FOCS* **33**, pp. 670–675.

[14] M. Dalal, and D. W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing letters* **44**:173–180, 1992.

[15] R. G. Downey, and M. R. Fellows. Fixed parameter tractability and completeness.

[16] Eshghi, K. and R. A. Kowalski. "Abduction Compared with Negation by Failure." *Proc. 6th International Conference on Logic Programming*, 1989, p. 234.

[17] M. Fitting. 1985. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming* **2**(4), pp. 295–312.

[18] J. Franco. 1986. On the probabilistic performance of algorithms for the Satisfiability problem. *Information Processing Letters*, **23**, pp. 103–106.

[19] J. Franco and Y. Ho. 1988. Probabilistic performance of a heuristic for the satisfiability problem. *Discrete Applied Mathematics* **22**, pp. 35–51.

[20] J. Franco, J. Goldsmith, J. Schlipf, E. Speckenmeyer, and R. Swaminathan. In Progress. An algorithm for the class of pure implicational formulas.

[21] J. Franco, J. Schlipf, and R. Swaminathan. In Progress. Improved single look-ahead unit resolution.

[22] J. Franco and R. Swaminathan. To appear. Average case results for satisfiability algorithms under the random-clause-width model. *Annals of Mathematics and Artificial Intelligence.*

[23] A. M. Frieze and S. Suen. Analysis of simple heuristics for random instances of 3-SAT. To appear.

[24] G. Gallo and M. G. Scutella. 1988. Polynomially solvable satisfiability problems. *Information Processing Letters* **29**, pp. 221–227.

[25] M. Gelfond. 1989. Autoepistemic Logic and Formalization of Commonsense Reasoning. In *Non-Monotonic Reasoning*, M. Reinfrank, J. de Kleer, M. L. Ginsberg and E. Sandewall, editors. Lecture Notes in Artificial Intelligence, 346. Springer-Verlag, pp. 176–186.

[26] M. Gelfond and V. Lifschitz. 1988. The stable model semantics for logic programming. *Proc. 5th Internationl Conference on Logic Programming.*

[27] P. Heusch. 1995. The Complexity of the Falsifiability Problem for Pure Implicational Formulas. *Proc. 20th Int. Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, J. Wiedermann, P. Hajek (Eds.), Prague, Czech Republic. *Lecture Notes in Computer Science* (LNCS **969**), Springer-Verlag, Berlin, pp. 221–226.

[28] M. M. Halldórsson. Approximating k-Set Cover and complementary graph coloring. Tech Report, Science Institute, University of Iceland, Reykajavik, Iceland.

[29] H. Kleine Büning. On generalized Horn formulas and k resolution. *Theoretical Computer Science* **116**, pages 405–413, 1993.

[30] H. Kleine Büning, and Theodor Lettmann. Aussagenlogik: Deduktion und Algorithmen. B.G. Teubner, Stuttgart, 1993. English version to appear in 1996.

[31] W. Marek and M. Truszczyński. 1991. Autoepistemic logic. *Journal of the ACM* **38**(3), pp. 588–619.

[32] Nerode, A., and V. S. Subrahmanian. "Implementing Stable Semantics by Linear Programming." *Logic Programming and Non-monotonic Reasoning: Proc. of the 2nd International Workshop*, L. M. Pereira and A. Nerode, eds., MIT Press, 1993, pp. 23-42.

[33] Niemelä, I., and P. Simons. "Efficient Implementation of the Well-founded and Stable Model Semantics." Technical Report 7/96, Fachbereite INFORMATIK, Universität Koblenz-Landau (Germany), 1996.

[34] J. Reif. A topological approach to dynamic graph connectivity. *Information Processing Letters* **25**(1), pp. 65–70.

[35] J. Schlipf. 1991. Representing epistemic intervals in logic programming. In *Logic Programming and Non-monotonic reasoning: Proceedings of the First International Workshop*, A. Nerode, W. Marek, and V. S. Subrahmanian, editors. MIT Press, Cambridge, Massachusetts.

[36] J. Schlipf, F. Annexstein, J. Franco, R. Swaminathan. 1995. On finding solutions to extended Horn sets. *Information Processing Letters* **54**, pp. 133–137.

[37] R. P. Swaminathan and D. K. Wagner. 1995. The arborescence-realization problem. *Discrete Applied Mathematics*, **59**, pp. 267–283.

[38] C. Tovey. 1984. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics* **8**, pp. 85–89.

[39] K. Trümper. 1978. On balanced matrices and Tutte's characterization of regular matroids, preprint.

[40] K. Truemper. In preparation, expected 1997. Effective Logic Computation.

[41] M. H. van Emden and R. A. Kowalski. 1976. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM* **23**(4), pp. 733–742.

[42] A. Van Gelder. 1989. The alternating fixpoint of logic programs with negation. In *8th ACM Symposium on Principles of Database Systems*, pp. 1–10.

[43] A. Van Gelder, K. A. Ross, and J. S. Schlipf. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* **38**(3), pp. 620–650.