Indefinite Models and Parallel Positive Resolution for Disjunctive Stratified Logic Programs

Marek A. Suchenek
California State University – Dominguez Hills
Carson, CA 90747 U.S.A.
e-mail addr: Suchenek@csudh.edu

Abstract

The subject of this paper is the semantics of propositional disjunctive logic programs and indefinite deductive data bases. The concept of the indefinite model of a disjunctive logic program is introduced. The algorithm of computing the indefinite model and an algorithm of evaluation of clauses in the indefinite model, both using a parallel positive variant of resolution, are presented. The evaluation algorithm is extended to cover the stratified programs. The complexity of these algorithms and their scope of applicability is analyzed.

Key words: *hierarchically-minimal semantics, indefinite deductive data bases, complexity of minimal entailment, prioritized circumscription,*

1 Introduction

In this paper we investigate a resolution based method for evaluation of queries to propositional indefinite data bases by means of, what we call, an *indefinite modeling*. Our approach extends the well known results of research in semantics of negation, particularly the closed-world assumption, circumscription, and minimal model theory (cf.[Lif85a, Lif85b, YH85, Suc89, SS90, Suc90, Suc93, Suc94]). Its generalization over non-propositional data bases (i.e., with universal quantifier $\forall$) is the subject of forthcoming [Suc97].

Under the minimal-model semantics, a query q to a disjunctive deductive data base p is answered positively iff q is true in all minimal models of p. This fact is usually articulated by $p \vdash_{min} q$ (p minimally entails q). The minimal-model semantics has the following neat characterization in terms of first-order provability (propositional, in this paper) of positive (i.e., negation-free) sentences:

$$p \vdash_{min} q \textbf{ iff } Cn_{Pos}(p \wedge q) \subseteq Cn(p) \qquad (1)$$

where $Cn_{Pos}(X)$ is the set of all positive logical consequences of X, and Cn is the set of all logical consequences of X (see [YH85] for the proof of the case of q being a disjunction of negative ground literals, and [Suc89] for the proof of the general case with universal quantifier). Intuitively, the right-hand side of (1) means that every positive consequence of $p \wedge q$ is provable from p itself. In particular, if q is a positive sentence then $p \vdash_{min} q$ is equivalent to $p \vdash q$. Consequently, characterization (1) yields the following corollary: p and p' have the same positive consequences iff for every clause q, $p \vdash_{min} q$ and $p' \vdash_{min} q$ are equivalent.

The above fact gives rise to the following scheme of deciding $p \vdash_{min} q$. If p' is a formula equivalent to the conjunction of all positive consequences of p then decide $p' \vdash_{min} q$ instead of $p \vdash_{min} q$. It is being hoped that, at least in certain cases, one can find p' for which the decision problem $p' \vdash_{min} q$ is easier than $p \vdash_{min} q$. Of course, p' may be assumed to be a positive formula. In this paper we consider two forms of such alternate p': a positive reduced *disjunctive form* (DF), and a positive reduced *conjunctive form* (CF). The former constitutes, in fact, the familiar set of minimal models of p, and allows for straightforward algorithm of deciding $p' \vdash_{min} q$. We call the latter an *indefinite model* of p, and develop for it a method of deciding $p' \vdash_{min} q$ which uses the criterion (1) without actually computing all the positive consequences of $p' \wedge q$. This method involves a transformation of data base p into suitable DF or CF, using a *parallel positive* variant of resolution, complete with respect to positive conclusions. Although in many cases, computing an indefinite model p' of p a program (also a CF-formula) seems quite easy, the problem $p' \vdash_{min} q$ for p' in a conjunctive form is far from being trivial.

Our method of indefinite modeling extends over strat-

ified data bases whose semantics is given by the hierarchically-minimal entailment $\vdash_{Hmin}$ of [SS90] (or equivalently, by prioritized circumscription of [Lif85a, Lif85b]). This stratified case and its semantics constitutes the central point of interest in this paper, and is briefly introduced below.

The very form of a *stratified* program p (specifically, the occurrences of negation in the bodies of its clauses) defines a hierarchy $\mathcal{L}$ of first-order languages L_i which extend alphabets of one another, and partitions p onto *strata* σ_i. The concept of hierarchically-minimal model of p differs from the regular (that is, non-hierarchical) one in that its construction follows a series of model minimizations, one for each stratum σ_i. The "basis step" in this inductive process consists of the construction of a minimal model of the innermost stratum σ_0 of p in its smallest language L_0. The "inductive step" proceeds as follows. Let the language L' of stratum σ' be the closest extension of the language L of stratum σ in the hierarchy $\mathcal{L}$. The hierarchically-minimal model of σ' in L' is obtained by expansion of hierarchically-minimal model of σ in L to a minimal model of $\sigma \cup \sigma'$ in L'. This step requires minimization of a model of $\sigma \cup \sigma'$ in L', where the only predicates that are allowed to vary are those in $L' \setminus L$, and is possible for every stratified program p. Finally, the hierarchically-minimal entailment $p \vdash_{Hmin} q$ is defined as the truthfulness of q in all hierarchically-minimal models of p.

Computing a hierarchically-minimal model of a stratified logic program p is, generally, much more complex than computing just a minimal model of p, and makes the direct verification of q in all hierarchically-minimal models of p impractical. In Section 5 we present an alternative decision algorithm for $p \vdash_{Hmin} q$ based on the concept of *the weakest precondition* which, in a way characteristic to abductive reasoning, allows for backward evaluation of q in the indefinite model of p without actually computing models for relevant strata of p. It turns out that our approach, by appropriate generalization of the concept of stratification, eliminates the need of clauses with negation in the body.

Since this paper describes a research in progress, the results and ideas presented here are preliminary and have no pretenses to completeness. Moreover, **most of the contents of Sections 2 and 3 is known from elsewhere (albeit in a different notation) and has been included for comparison. New ideas and results are offered in Sections 4 and 5**.

2 Notation, etc.

We confine ourselves to a first-order language L without equality and variables (and, therefore, without quantifiers), with usual Boolean connectives $\vee, \wedge, \neg$ (we treat all other connectives as appropriate abbreviations), finitely many predicate symbols $A, B, ...$, and finitely many constants $1, 2, ..., M$. We denote formulas of L by $p, q, r, ...$. We call atomic formulas of L propositional variables (or simply variables) and place their arguments in subscripts rather than in parentheses (e.g., we write $B_{3,7}$ instead of $B(3,7)$). L has only finitely many distinct propositional variables; we denote their number by N. If φ is a set of formulas then $\wedge\varphi$ stands for $\wedge_{p\in\varphi}p$, with $\wedge 0$ interpreted as *true*, and dually for $\vee\varphi$, with $\vee 0$ interpreted as *false*. $\neg\varphi$ means $\{\neg p | p \in \varphi\}$. We extend the above convention over sets of sets of formulas in an obvious way; e.g. $\wedge\{\varphi_1, ..., \varphi_n\}$ means $\{\wedge\varphi_1, ..., \wedge\varphi_n\}$.

A *clause carrier* is a set of literals with every propositional variable occurring at most once. A *program carrier* is a set of clause carriers. We use small Greek letters to denote clause carriers, and capital Greek letters for program carriers. Both program and clause carriers are finite since L has only finitely many distinct variables. A *clause* is a formula of the form $\vee\varphi$, where φ is a clause carrier. A *program* (for simplicity we use this term instead of *indefinite deductive data base*) is a formula of the form $\vee\Phi$, where Φ is a program carrier, that is, a program is a finite set of clauses. Since the meaning of $\vee\Phi$ is the same as $\wedge\vee\Phi$, we will use $\vee\Phi$ and $\wedge\vee\Phi$ interchangeably. If X is a carrier, or a program or a clause, $|X|$ denotes the size of X measured as the number of occurrences of propositional variables in X, while $card(X)$ denotes the number of *elements* of X.

A *term* is a formula of the form $\wedge\varphi$, and is called a *minterm* if φ contains occurrences of all propositional variables of L. A conjunctive form formula (abbr: CF) is a formula of the form $\wedge\vee\Phi$. Similarly, a disjunctive form formula (abbr: DF) is a formula of the form $\vee\wedge\Phi$. If all elements of $\wedge\Phi$ are minterms then $\vee\wedge\Phi$ is called a disjunctive normal form formula (abbr: DNF). A set Y of subsets of X is called an *anti-chain* in X if no two distinct elements of Y are subsets one of another. If Φ is an anti-chain then $\wedge\vee\Phi$ is called a *reduced conjunctive form* formula (abbr: RCF), and $\vee\wedge\Phi$ a *reduced disjunctive form* formula (abbr: RDF).

Adjective *positive* refers to formulas and carriers without occurrences of negation, *negative* refers to those equivalent to negated positive ones. For instance, $A \vee B$ is a positive formula, while $\neg(\neg A \wedge \neg B)$ and $A \supset B$ are not ($A \supset B$ is an abbreviation for $\neg A \vee B$). Therefore, in the sense of this convention, positive clauses (or programs) are elsewhere called bodiless clauses (or programs). Bodiless definite programs constitute the easy case in traditional logic programming; they coincide with their minimal Herbrand mod-

els. Indefinite bodiless programs are much less trivial, and will serve as *indefinite models*. If φ is a clause carrier then φ^{Pos} denotes the set of all positive literals of φ. If Φ is a program carrier then $\Phi^{Pos} = \{\varphi \in \Phi | \varphi \text{ is positive}\}$ and $\Phi_{Pos} = \{\varphi^{Pos} | \varphi \in \Phi\}$. For instance, $\{\{A, \neg B\}, \{C\}\}^{Pos} = \{\{C\}\}$, while $\{\{A, \neg B\}, \{C\}\}_{Pos} = \{\{A\}, \{C\}\}$. We use similar convention for Φ^{Neg} and Φ_{Neg}.

Minterms $\wedge\varphi$ may be interpreted as propositional models of formulas of L. For instance, minterm $\wedge\varphi$ is a model of clause $\vee\psi$ iff $\varphi \cap \psi$ is non-empty, which, of course, is equivalent to $\wedge\varphi \vdash \vee\psi$. Therefore, DNF-formulas $\vee \wedge \Phi$ constitute sets of propositional models of formulas of L. Because every element of a DNF-formula $\wedge\Phi$ is a minterm, Φ is unambiguously determined by its positive fragment Φ_{Pos}; therefore we will think of Φ_{Pos} as of the set of models. Perhaps abusing the notation, we use CWA to denote the function which reconstructs Φ from Φ_{Pos}. For instance, if L contains only two variables A and B, $CWA(\{A\}, 0, \{A, B\}) = \{\{A, \neg B\}, \{\neg A, \neg B\}, \{A, B\}\}$. We denote by $Mod(p)$ the set of all models of p. If Φ is the set of models of p (that is, Φ is positive, $\vee\wedge CWA(\Phi)$ is a DNF-formula, and $\vee \wedge CWA(\Phi) \equiv p$ is a propositional tautology) then the set $min(\Phi)$ of all $\subseteq$-minimal elements of Φ is the set of minimal models of p, which we also denote by $Mod_{min}(p)$. $Mod_{min}(p)$ is a positive anti-chain, and for every positive anti-chain Ψ, there is a formula p of L (e.g., $\vee \wedge \Psi$) with $\Psi = Mod_{min}(p)$. Therefore sets of minimal models may be identified with positive RDF-formulas. It follows from (1) that a positive anti-chain Ψ is the set of minimal models of formula p iff $\vee \wedge \Psi$ is logically equivalent to the conjunction of all positive clauses which are logical consequences of p (there are only finitely many of them).

Relation of minimal entailment $\vdash_{min}$ is defined by: $p \vdash_{min} q$ iff $Mod_{min}(p) \subseteq Mod(q)$, that is, every minimal model of p is a model of q. Question "Does $p \vdash_{min} q$?" makes a formal articulation of query "q?" to program p. Of course, $p \vdash_{min} \wedge\varphi$ iff for every $q \in \varphi$, $p \vdash_{min} q$. Therefore rather than allowing arbitrary programs as queries, we will restrict ourselves to clauses. By the above observation, this will cover all reasonable forms of queries to deductive data bases, at least in the propositional case.

It follows that for every DF-formula $\vee \wedge \Phi$, the following are equivalent: $\vee \wedge \Phi \vdash_{min} q$, $\vee \wedge (\Phi_{Pos}) \vdash_{min} q$, $\vee\wedge(min(\Phi_{Pos})) \vdash_{min} q$, $\vee\wedge(CWA(min(\Phi_{Pos}))) \vdash_{min} q$, and $\vee \wedge (CWA(min(\Phi_{Pos}))) \vdash q$. The last statement reduces to a finite number of inclusions for any clause q. Therefore for deductive data bases in disjunctive form, deciding $\vdash_{min}$ is straightforward and can be carried out in polynomial time with respect to the size of a data base and its query.

Sets of positive RCF formulas $\wedge \vee \Phi$ define *indefinite models* of formulas of L: $\vee\Phi$ is an indefinite model of formula p (notation: $Ind(p)$) iff $\wedge \vee \Phi$ is logically equivalent to the conjunction of all positive logical consequences of p (and therefore, to $\vee \wedge Mod_{min}(p)$). It follows from (1) that for any formulas p and q,

$$p \vdash_{min} q \textbf{ iff } Ind(p) \vdash_{min} q. \tag{2}$$

3 Minimal Modeling

In this section we discuss some tractable cases of deciding $p \vdash_{min} q$ by direct verification of q in all minimal models of p. We call this verification process a *minimal modeling*.

The set $Mod(p)$ of models of program p is a positive carrier Φ, such that the $DNF-$formula $\vee \wedge CWA(\Phi)$ is logically equivalent to p. The set $Mod_{min}(p)$ is the $\subseteq -dense$ anti-chain Ψ in Φ. Given a positive carrier Φ, verifying if all elements of $\wedge\Phi$ are models of a clause $\vee\psi$ is easy because of simplicity of $CWA(\wedge\Phi)$: for any $\varphi \in \Phi$, φ is a model of $\vee\psi$ iff $(\psi^{Pos} \cap \varphi) \cup (\psi^{Neg} \setminus \varphi)$ is non-empty. (A straightforward generalization of this scheme can be used for equally easy verification of a more general case of ψ being a set of any clauses, not just a set of literals; recall that the problem $\vee CWA(\wedge\Phi) \vdash q$ is in P for arbitrary positive carrier Φ and formula q.) Obviously, $Mod_{min}(p) \subseteq Mod(p)$, which implies, in the case when $p \vdash q$ holds, that the direct verification of $p \vdash_{min} q$ is never more complex than the direct verification of $p \vdash q$. If $p \vdash q$ is *not* the case then in some cases deciding $p \vdash_{min} q$ may be more costly than the direct verification of $p \nvdash q$, since one can (incidentally) hit a non-minimal model of p which is not a model of q right at the very beginning of the verification process. However, the ratio of the average size of $Mod(p) \setminus Mod_{min}(p)$ to the average size of $Mod_{min}(p)$ is approx. $\sqrt{2N} - 1$. Therefore, hunting for an element of $Mod(p) \setminus Mod_{min}(p)$ which is not a model of q is very likely to be more costly than checking if q is true in all elements of $Mod_{min}(p)$.

This observation gives rise to a hope that $\vdash_{min}$ is more often feasibly verifiable than $\vdash$ is. For instance, a program $\varphi = \{A_1 \vee ... \vee A_n, B_1 \vee ... \vee B_n\}$, where A_i's and B_j's are distinct variables, has only n^2 minimal models, but it has $(2^n - 1)^2$ models. Thus the direct verification of $\wedge\varphi \vdash_{min} q$ is tractable in this case, while direct verification of $\wedge\varphi \vdash q$, in the worst case, is not. And, of course, any *definite* program has only one min-

imal model (the least fixed-point of the operator T_p), which makes the direct verification of $\wedge\varphi \vdash_{min} q$ even easier.

The process of verification of minimal entailment $\vdash_{min}$ by minimal modeling consists of two phases. First, given program $\vee\Phi$, the set $Mod_{min}(\vee\Phi)$ has to be constructed. Second, given a query $\vee\psi$, the non-emptiness of $(\psi^{Pos} \cap \varphi) \cup (\psi^{Neg} \setminus \varphi)$ for every minimal model $\varphi \in Mod_{min}(\vee\Phi)$ must be verified. The latter process can be done in $O(|Mod_{min}(\vee\Phi)| \times |\psi|)$ steps in the worst case. The former may be more costly; however the time spent on its execution will be spread over presumably a large number of queries to the same program $\vee\Phi$. Below, we discuss two cases when this process leads to a feasible computation.

Case 1. If reducing of $\vee\Phi$ to a $DNF-$formula $\vee \wedge Mod(\vee\Phi)$ can be done quickly (in polynomial time), then $Mod_{min}(\vee\Phi)$ may also be built in a polynomial time; having $Mod(\vee\Phi)$ one can purge all the non-minimal elements of $Mod(\wedge\varphi)$, which obviously can be done in $O(|Mod(\vee\Phi)|^2 \times N^2)$, (where N is the number of variables in L).

Case 2. If there are only few (polynomially many), say K, structures for L which do not include minimal models of $\vee\Phi$ then $Mod_{min}(\vee\Phi)$ may be built in polynomial time, according to the following algorithm. Think of relation of inclusion between sets of variables of L as of a digraph G with the empty set on the top and the set of all variables on the bottom, with edges going top down, connecting sets with their least proper supersets. A straightforward modification of breath-first search traversal of G with concurrent checking if the current vertex v is a model of $\vee\Phi$ (in $O(N^2)$ worst-case time) and cutting of its children in the case v *is* a model of $\vee\Phi$, will take $K + |Mod_{min}(\vee\Phi)|$ steps. Because $|Mod_{min}(\vee\Phi)| \leq 2K$, the entire process will stop in $O(N^2 \times K)$ worst-case time, which of course is polynomial.

Minimal modeling has several obvious drawbacks. First of all, the biggest cardinality of $Mod_{min}(p)$ is $\begin{pmatrix} N \\ \lceil \frac{N}{2} \rceil \end{pmatrix}$ (cardinality of the longest anti-chain in a power set of an N-element set), or approximately, $\sqrt{\frac{2}{\pi}} \times \frac{2^N}{\sqrt{N}}$, which makes its computation infeasible except for small values of N. For instance, let us consider a well known (cf. [CR79, Hak85, BT88]) hard case for resolution proofs, the pigeonhole principle (PHP). Let PHP^- be

$$\{\{A_{1,1}, ..., A_{1,n}\},$$
$$\vdots$$
$$\{A_{n+1,1}, ..., A_{n+1,n}\}\}$$

and PHP^+ be

$$\{\{A_{1,1}, A_{2,1}\}, \{A_{1,1}, A_{3,1}\}, ..., \{A_{n,1}, A_{n+1,1}\},$$
$$\vdots$$
$$\{A_{1,n}, A_{2,n}\}, \{A_{1,n}, A_{3,n}\}, ..., \{A_{n,n}, A_{n+1,n}\}\},$$

where all $A_{i,j}$'s are distinct variables with the intentional interpretation of "i-th pigeon sits in the j-th hole." Obviously, $\wedge \vee PHP^- \vdash \vee \wedge PHP^+$ holds: it states that if there are $n+1$ pigeons occupying n pigeonholes then at least one pigeonhole must accommodate at least two pigeons. Because PHP^+ is a positive carrier, $\wedge \vee PHP^- \vdash \vee \wedge PHP^+$ and $\wedge \vee PHP^- \vdash_{min} \vee \wedge PHP^+$ are equivalent. Hence $\vee \wedge PHP^- \vdash_{min} \wedge \vee PHP^+$ holds as well. $\wedge \vee PHP^-$ has $N = n \times (n+1)$ variables and roughly 2^N models, which makes a direct verification of $\wedge \vee PHP^- \vdash \vee \wedge PHP^+$ (as well as its proof by resolution) intractable. Number of *minimal* models of $\wedge \vee PHP^-$ is n^{n+1}. Although a considerable improvement over $\vdash$ (with ratio approximately $(\frac{2}{\sqrt[n]{n}})^N$), this size of $Mod_{min}(\wedge \vee PHP^-)$ leaves direct verifiability of $\wedge \vee PHP^- \vdash_{min} \vee \wedge PHP^+$ out of the question.

Second, even relatively easy queries will involve their verification in all minimal models of the program in question. For instance, a program $\vee\Phi = \{A_1 \vee ... \vee A_n, B_1 \vee ... \vee B_n\}$ has n^2 minimal models, therefore direct verification of straightforward $\Phi \vdash_{min} \vee \vee \Phi$ will involve $\Omega(n^3)$ steps (plus the time to compute $Mod_{min}(\Phi)$).

Third, adding new clauses to a program φ may considerably change $Mod_{min}(\varphi)$ (in fact, $Mod_{min}(\varphi)$ and $Mod_{min}(\varphi')$, where $\varphi \subseteq \varphi'$, may be disjoint). In this case, the costly process of computation of $Mod_{min}(\varphi)$ must be repeated.

4 Indefinite Modeling

As we have seen, converting a program $\vee\Phi$ into its positive RDF-formula $\vee \wedge Mod_{min}(\vee\Phi)$ may be very costly, as the size of $Mod_{min}(\vee\Phi)$ can be exponential in the size of $\vee\Phi$. It may also happen that converting a program $\vee\Phi$ into an equivalent DNF-formula is an exponentially lengthy process, even if $|Mod_{min}(\vee\Phi)|$ is only polynomial in $|\vee\Phi|$. However, because a program is a CF-formula itself, converting it into a positive RCF-formula $\wedge \vee Ind(\vee\Phi)$ should be much easier, at least in some cases. For instance, if $\vee\Phi$ is a positive program then $Ind(\vee\Phi)$ is a result of purging from $\vee\Phi$ all the clauses which are subsumed by other clauses of $\vee\Phi$, which can be done in $O(|\vee\Phi|^2 \times N^2)$ time. Existence of cases like this gives rise to use of *indefinite modeling* instead of minimal modeling. In this section we present and analyze algorithms for converting a

program $\vee\Phi$ into its positive *RDF*-formula $Ind(\vee\Phi)$, and for deciding $Ind(\vee\Phi) \vdash_{min} q$. Both algorithms are based on the following *parallel positive* variant of resolution:

$$\frac{q \vee \neg R_1 \vee ... \vee \neg R_n \mid p_1 \vee R_1 \mid \ldots \mid p_n \vee R_n}{q \vee p_1 \vee ... \vee p_n}, \quad (3)$$

where $n > 0$, R_i's are propositional variables, and q and p_i's are distinct positive clauses with no occurrences of variables $R_1, ..., R_n$.

Both algorithms use the operator of direct positive consequence $\mathcal{C}$, which assigns to a set of clauses $\vee\Phi$ the set $\mathcal{C}(\vee\Phi)$ of all positive clauses which can be derived from $\vee\Phi$ by subsumption and a *single* application of an instance of (3). Given a program $\vee\Phi$, Algorithm 4.1 computes $Ind(\vee\Phi)$.

Algorithm 4.1

$K := min(\vee\Phi)$; $I{:=}K^{Pos}$;$J{:=}K \setminus I$;
<u>while</u> $\mathcal{C}(I \cup J)$ is not subsumed by I <u>do</u>
$I := min(I \cup \mathcal{C}(I \cup J))$;
{At this point, all positive consequences }
{of $\vee\Phi$ are subsumed by I }
{Therefore, $I = Ind(\vee\Phi)$ }
return(I);

Here, $min(X)$ is the set of all clauses of X which are not subsumed by other clauses of X. The following lemma guarantees partial correctness of Algorithm 4.1 (and, in fact, the completeness of the parallel positive variant of resolution with respect to positive consequences).

Lemma 4.2 Let Φ be a program carrier. If there is a positive clause q which is a logical consequence of $\vee\Phi$ but is not subsumed by any of the clauses of $\vee\Phi^{Pos}$ then $\mathcal{C}(\vee\Phi)$ is not subsumed by $\vee\Phi^{Pos}$.

Proof. Let $\Psi = \Phi \setminus \Phi^{Pos}$. Assume that $\mathcal{C}(\vee\Phi)$ is subsumed by $\vee\Phi^{Pos}$, that is, for every $\psi \in \vee\Psi$, $\mathcal{C}(\vee\Phi^{Pos} \cup \{\psi\})$ is subsumed by $\vee\Phi^{Pos}$. Hence, $Mod(\vee\Phi^{Pos}) = Mod(\vee\Phi^{Pos} \cup \{\psi\} \cup \mathcal{C}(\vee\Phi^{Pos} \cup \{\psi\}))$, that is, $Mod_{min}(\vee\Phi^{Pos}) = Mod_{min}(\vee\Phi^{Pos} \cup \{\psi\} \cup \mathcal{C}(\vee\Phi^{Pos} \cup \{\psi\}))$. On the other hand, straightforward verification yields $Mod_{min}(\vee\Phi^{Pos} \cup \{\psi\}) = Mod_{min}(\vee\Phi^{Pos} \cup \{\psi\} \cup C(\vee\Phi^{Pos} \cup \{\psi\}))$. From this we conclude that $Mod_{min}(\vee\Phi^{Pos}) = Mod_{min}(\vee\Phi^{Pos} \cup \{\psi\}) = Mod_{min}(\vee\Psi^{Pos} \cup \vee\Psi) = Mod_{min}(\vee\Phi)$, that is, $\vee\Phi^{Pos} \vdash_{min} \wedge\vee\Phi$. Therefore, by thm. 4.3 in [Suc89], $\wedge\vee\Phi \in cwa_S(\vee\Phi^{Pos})$, that is, by definition of cwa_S (def. 3.1 in [Suc89]), all positive consequences of $\vee\Phi$ are provable from Φ^{Pos}. □

To evaluate the complexity of Algorithm 4.1, let us consider its following variant which calculates an "unpurged" version $Ind^*(\vee\Phi)$ of $Ind(\vee\Phi)$.

$K := \vee\Phi$; $I{:=}K^{Pos}$;$J{:=}K \setminus I$;
<u>while</u> $\mathcal{C}(I \cup J)$ is not subsumed by I <u>do</u>
$I := I \cup \mathcal{C}(I \cup J)$;
return(I);

We have: $Ind(\vee\Phi) = min(Ind^*(\vee\Phi))$. Obviously, the number of iterations of the *while* loop is the same for both algorithms. Because this number cannot be greater than the number of elements in $Ind^*(\Phi)$ (which is finite; actually, it can be shown that the number of iterations is not greater then $|\Phi|$), Algorithm 4.1 halts, and therefore, is totally correct, that is, it returns the indefinite model $Ind(\vee\Phi)$ of $\vee\Phi$.

Purging out non-minimal (with respect to subsumption) clauses form $I \cup \mathcal{C}(I \cup J)$ can be done as a part of checking whether I does not subsume $\mathcal{C}(I \cup J)$, which takes $O(|I| \times |\mathcal{C}(I \cup J)|)$ time in the worst case, that is, $O(|Ind^*(\vee\Phi)| \times |\mathcal{C}(\vee\Phi \cup Ind^*(\vee\Phi))|)$. Therefore, the entire algorithm executes in $O(|\vee\Phi| \times |Ind^*(\vee\Phi)| \times |\mathcal{C}(\vee\Phi \cup Ind^*(\vee\Phi))|)$ worst-case time. If the size of $\mathcal{C}(\vee\Phi \cup Ind^*(\vee\Phi))$ is polynomial in the size of Φ (say, $O(|\Phi|^c)$) then the worst-case running time of Algorithm 4.1 is polynomial in $|\Phi|$ too ($O(|\Phi|^{2c+1})$). There are two kinds of hard cases for Algorithm 4.1: when $|Ind(\Phi)|$ is large (e.g., exponential) in $|\Phi|$, and when $|Ind(\Phi)|$ is small (e.g., polynomial) in $|\Phi|$ but $|Ind^*(\vee\Phi)|$ is large in $|\Phi|$. For instance, $|Ind(\{A_1 \vee ... \vee A_n, B_1 \vee \neg A_1, ..., B_n \vee \neg A_n\})| = 2^n$, which illustrates the first case. Since Algorithm 4.1 is based on resolution and $|Ind(\vee PHP^- \cup \vee\neg PHP^+)| = 0$ (because $\vee PHP^- \cup \vee\neg PHP^+$ is inconsistent), PHP necessarily constitutes the second kind of a hard case (otherwise Algorithm 4.1 would yield a resolution based proof of PHP with only polynomially many steps, which is impossible; cf. [CR79]).

Now, we turn to the decision algorithm for $\vdash_{min}$.

Theorem 4.3 For every positive program $\vee\Phi$ and every clause q, $Ind(\vee\Phi) \vdash_{min} q$ **iff** $Ind(\vee\Phi)$ subsumes $\mathcal{C}(Ind(\vee\Phi) \cup \{q\})$.

Proof. We inter from Lemma 4.2 that $\mathcal{C}(Ind(\vee\Phi) \cup \{q\})$ subsumes all positive clauses provable from $\vee\Phi \cup \{q\}$. By def 3.1 and thm. 4.3 in [Suc89], $Ind(\vee\Phi) \vdash_{min} q$ iff all positive clauses provable from $Ind(\vee\Phi) \cup \{q\}$ are subsumed by $Ind(\vee\Phi)$. From this we conclude the thesis. □

Having calculated $Ind(\vee\Phi)$, the problem of $\vee\Phi \vdash_{min} q$

reduces by (2) to $Ind(\vee\Phi) \vdash_{min} q$, and by Theorem 4.3 to the following question:

$$\text{Is } \mathcal{C}(Ind(\vee\Phi) \cup q) \text{ subsumed by } Ind(\vee\Phi)? \qquad (4)$$

which may be answered in $O(|Ind(\vee\Phi)| \times |\mathcal{C}(Ind(\vee\Phi) \cup \{q\})|)$ worst-case running time.

We conclude this section with examples of easy cases for indefinite modeling. For instance, allowing for at most one negation per clause makes $Ind(\vee\Phi)$ easy to calculate. Below is another example which allows for fast computation of $Ind(\vee\Phi)$.

Theorem 4.4 If the length of every positive clause of $\vee\Phi$ is in $O(1)$, and every literal occurring negatively in Φ has the total number of occurrences in Φ at most 2 then Algorithm 4.1 runs in polynomial worst-case time.

Proof follows from the fact that there may be at most $2^\lambda \times N$ applications of a positive instance of resolution during the execution of Algorithm 4.1, where λ is the maximum length of a positive clause in $\vee\Phi$. □

If $card(Ind(\vee\Phi))$ is in $O(1)$ then simple application of distributive law yields $Mod_{min}(\vee\Phi)$ and decision $Ind(\vee\Phi) \vdash_{min} q$ in polynomial time. Analysis of the one-step parallel positive resolution yields a number of characterizations of other easy cases for $\vee Ind(\vee\Phi) \vdash_{min} q$. For instance, for each variable R, $Ind(\vee\Phi) \vdash_{min} \neg R$ iff R does not occur in Φ. Here is another one.

Theorem 4.5 If the number of negative literals of q having more than 1 positive occurrence in $Ind(\vee\Phi)$ is $O(1)$ then deciding (4) can be performed in $O(|Ind(\vee\Phi)|^2 \times |q|)$ worst-case time.

Proof. Generating $\mathcal{C}(Ind(\vee\Phi) \cup \{q\})$ takes $O(|Ind(\vee\Phi)| \times |q|)$ in the worst-case (all possible applications of parallel positive instances of resolution involving only one non-positive formula q), and then checking subsumption takes $O(|Ind(\vee\Phi)|^2 \times |q|)$. □

Theorem 4.5 allows for speeding up Algorithm 4.1 for programs Φ with $O(1)$ occurrences of negated literals (which seems to be a typical case for deductive data bases).

Algorithm 4.6

$I := min(\vee\Phi^{Pos})$;
for $\pi \in perm(min(\Phi \setminus \Phi^{Pos}))$ do
    for $q \in \pi$ do
        $I := min(I \cup \mathcal{C}(I \cup q))$;
return(I);

where $perm(X)$ is the set of all enumerations of X.

Theorem 4.7 If Φ has $O(1)$ occurrences of negated literals then Algorithm 4.6 computes $Ind(\Phi)$ in $O(|Ind(\Phi)|^2)$ worst-case running time.

Proof. Running time is evaluated by application of Theorem 4.5, letting $|q| \in O(1)$. Partial correctness follows from analysis of genealogical graph of any positive formula in $Ind(\Phi)$ computed by Algorithm 4.1: it follows that positive clauses p which are matched in (3) with their non-negative ancestors produce children which are subsumed either by ancestors of p or by siblings of p. □

Obviously, if Φ contains no negations and q is a positive clause then $\Phi \vdash_{min} q$ can be decided in $O(|\Phi| \times |\psi|)$ worst-case time. Below is a summary of easy cases.

i. Computation of $Ind(\vee\Phi)$ is polynomial in $|\Phi|$ if:
   (a) every clause of $\vee\Phi$ is $O(1)$ long, and each of its negative literal occurs at most twice (positively or negatively);
   (b) Φ has $O(1)$ occurrences of negative literals.

ii. Decision of $Ind(\vee\Phi) \vdash_{min} q$ is polynomial in $|Ind(\vee\Phi) \cup \{q\}|$ if:
   (a) $card(Ind(\vee\Phi))$ is in $O(1)$;
   (b) there are only $O(1)$ negative literals of q with more than one positive occurrence in $Ind(\vee\Phi)$;
   (c) (a special case) there are only $O(1)$ negative literals in q;
   (d) (a special case) $|q|$ is in $O(1)$.

iii. Decision of $Ind(\vee\Phi) \vdash_{min} q$ is polynomial in $|\vee\Phi \cup \{q\}|$ if:
   (a) $\vee\Phi \cup \{q\}$ has $O(1)$ occurrences of negative literals;
   (b) any other combination of subcases in (i) and (ii) holds.

5 Stratification

The method of indefinite modeling generalizes nicely over stratified programs. Quite surprisingly, the concept of indefinite model does not need any modifications in this case. This is a considerable advantage of indefinite modeling over minimal modeling which requires recomputation of the set of minimal models after stratification has been imposed or changed.

In this Section we analyse the case of programs with two strata, which covers the "inductive step" of the

definition of hierarchically-minimal model (end of Section 1). Since the "basis step" of that definition reduces to the minimal-model semantics, a routine induction over any tree hierarchy $\mathcal{L}$ of first-order languages extends our approach over programs with arbitrary finite number of strata. We distinguish a nonempty sublanguage L' of L, which partitions every program $\vee\Phi$ onto two strata: $\vee\Phi'$ and $\vee\Phi''$, where $\Phi' = \Phi \cap \mathcal{P}(L')$, and $\Phi'' = \Phi \setminus \mathcal{P}(L')$. $\vee\Phi'$ consists of all clauses of $\vee\Phi$ which do not contain occurrences of literals from outside L'; $\vee\Phi''$ contains the remaining clauses of $\vee\Phi$. Stratum $\vee\Phi'$ is given a higher priority during model minimization than stratum $\vee\Phi''$. Imposing this priority implies that certain minimal models of $\vee\Phi$ are no longer considered minimal for stratified $\vee\Phi$ (the converse is true, though).

Rather then using a relatively strong assumption of stratifiability (cf. [Apt88]) of $\vee\Phi$, we content ourselves with a weaker postulate, requiring that $\vee\Phi$ be a minimally conservative extension of $\vee\Phi'$, that is, that every minimal model (in the usual, non-stratified sense) of $\vee\Phi'$ has an expansion to a minimal model of entire $\vee\Phi$. We call every such an expansion a *hierarchically-minimal model* of $\vee\Phi$, and the entailment $\vdash_{Hmin}$ induced by hierarchically minimal semantics a *hierarchically-minimal entailment*. Any program which satisfies the above requirement is called a *layered* program. One can verify with ease that hierarchically-minimal models are exactly the models of prioritized circumscription. It is also easy to check that every stratified program is layered, but not vice versa.

For each clause q of L', $\vee\Phi \vdash_{min} q$ and $\vee\Phi \vdash_{Hmin} q$ are equivalent, as they are for negated literals q of L. For other clauses from $L \setminus L'$ $\vdash_{Hmin}$ behaves differently than $\vdash_{min}$. For instance, for positive clauses q of $L \setminus L'$, $\vee\Phi \vdash_{Hmin} q$ and $\vee\Phi \vdash q$ need not be equivalent. In particular, there are layered programs $\vee\Phi$ and positive clauses q of $L \setminus L'$ with $\vee\Phi \vdash_{Hmin} q$ but not $\vee\Phi \vdash q$. For example, if R is a variable of L' and S is a variable of $L \setminus L'$ then $\{\neg R \supset S\} \vdash_{Hmin} S$. (Recall, that $\neg R \supset S$ is an abbreviation for $R \vee S$.) We will characterize $\vee\Phi \vdash_{Hmin} q$ in terms of $Ind(\Phi)$, $\vdash_{min}$, and q. To that end, we need the following concept.

Definition 5.1 Let $\vee\Phi$ be a program and let q be a formula of L. The *weakest precondition* $WP(\vee\Phi, q)$ for q relative to Φ in L' is a set of clauses of L' which satisfies two conditions:

i. $\vee\Phi \cup WP(\vee\Phi, q) \vdash q$;

ii. for every sentence r of L', if $\vee\Phi \cup \{r\} \vdash q$ then $\{r\} \vdash \wedge WP(\vee\Phi, q)$.

□

Intuitively, the weakest precondition is equivalent to the weakest formula of the smaller language L' which together with $\vee\Phi$ implies q, that is, $\wedge WP(\vee\Phi, q)$ is logically equivalent to $\vee\{r \in L' | \vee\Phi \cup \{r\} \vdash q\}$. In particular, if q is a formula of L' then $WP(\vee\Phi, q) = \{q\}$. The following Lemma relates $\vdash_{Hmin}$ to $\vdash_{min}$ via WP.

Lemma 5.2 For every positive layered program Φ and every positive formula q of L,

$$\vee\Phi \vdash_{Hmin} q \textbf{ iff } \vee\Phi' \vdash_{min} \wedge WP(\vee\Phi, q).$$

Proof. $\vee\Phi \vdash_{Hmin} q$ **iff** [by [SS90], thm. 5.5] $q \in cwa_S(\vee\Phi \cup cwa_S(\vee\Phi'))$ **iff** [by [Suc89], thm. 4.3] $\vee\Phi \cup cwa_S(\vee\Phi') \vdash_{min} q$ **iff** [by positiveness of q and thm. 2.7 of [Suc90]] $\vee\Phi \cup cwa_S(\vee\Phi') \vdash q$ **iff** [by definition 5.1 of WP] $cwa_S(\vee\Phi') \vdash \wedge WP(\vee\Phi, q)$ iff [by [Suc89], thm. 4.3] $\vee\Phi' \vdash_{min} \wedge WP(\vee\Phi, q)$. □

The following lemma extends applicability of Lemma 5.2 over non-positive formulas q.

Lemma 5.3 For every positive program $\vee\Phi$ and every clause q,

$$\vee\Phi \vdash_{Hmin} q \textbf{ iff } \vee\Phi \vdash_{Hmin} \wedge\mathcal{C}(\vee\Phi \cup \{q\}).$$

Proof. ($\Rightarrow$) is obvious. ($\Leftarrow$) $\vee\Phi \vdash_{Hmin} \wedge\mathcal{C}(\vee\Phi \cup \{q\})$ **implies** [by [SS90], thm. 5.5] $\wedge\mathcal{C}(\vee\Phi \cup \{q\}) \in cwa_S(\vee\Phi \cup cwa_S(\vee\Phi'))$ **implies** [by thm. 4.3 of [Suc89]] $\vee\Phi \cup cwa_S(\vee\Phi') \vdash_{min} \wedge\mathcal{C}(\vee\Phi \cup \{q\})$ **implies** [by positiveness of $\wedge\mathcal{C}(\vee\Phi \cup \{q\})$ and thm 2.7 of [Suc90]] $\vee\Phi \cup cwa_S(\vee\Phi') \vdash \wedge\mathcal{C}(\vee\Phi \cup \{q\})$ **implies** [by positiveness of $\vee\Phi$ and Lemma 4.2] $\vee\Phi \cup cwa_S(\vee\Phi')$ proves all positive consequences of $\Phi \cup \{q\}$ **implies** [by [Suc89], thm. 4.3] for every minimal model $\mathcal{M}$ of $\vee\Phi \cup cwa_S(\vee\Phi')$ there exists a minimal model $\mathcal{N}$ of $\vee\Phi \cup \{q\}$ with $\mathcal{N} \subseteq \mathcal{M}$ **implies** [by [SS90], thm 5.4, every model of $cwa_S(\vee\Phi')$ is a minimal model of $\vee\Phi'$, therefore, $\mathcal{M} \mid\grave{}\, L'$ is a minimal model of $\vee\Phi'$; also, $\mathcal{N} \mid\grave{}\, L' \subseteq \mathcal{M} \mid\grave{}\, L'$, and $\mathcal{N} \mid\grave{}\, L' \models \vee\Phi']$ $\mathcal{N} \mid\grave{}\, L' = \mathcal{M} \mid\grave{}\, L'$ **implies** $\mathcal{N} \mid\grave{}\, L'$ is a model of $cwa_S(\vee\Phi')$ **implies** for every minimal model $\mathcal{M}$ of $\vee\Phi \cup cwa_S(\vee\Phi')$ there exists a minimal model $\mathcal{N}$ of $\vee\Phi \cup \{q\} \cup cwa_S(\vee\Phi')$ with $\mathcal{N} \subseteq \mathcal{M}$ **implies** [by [Suc89], thm. 4.3] $\vee\Phi \cup cwa_S(\vee\Phi')$ proves all positive consequences of $\vee\Phi \cup cwa_S(\vee\Phi') \cup \{q\}$ **implies** [by [Suc89], df. 3.1] $q \in cwa_S(\vee\Phi \cup cwa_S(\vee\Phi'))$ **implies** [by [SS90], thm. 5.5] $\vee\Phi \vdash_{Hmin} q$. □

For every program $\vee\Phi$ and clause q,

$$\vee\Phi \vdash_{Hmin} q \textbf{ iff } Ind(\vee\Phi) \vdash_{Hmin} q. \qquad (5)$$

This property follows from the fact, that if p and r have the same minimal models then they also have the same hierarchically-minimal models. Hence the following theorem.

Theorem 5.4 For every program $\vee\Phi$ and clause q, $\vee\Phi \vdash_{Hmin} q$ **iff** $Ind(\vee\Phi)'$ subsumes $\mathcal{C}(Ind(\vee\Phi)' \cup WP(Ind(\vee\Phi), \wedge\mathcal{C}(Ind(\vee\Phi) \cup \{q\})))$.

Proof. $\vee\Phi \vdash_{Hmin} q$ **iff** [by (5)] $Ind(\vee\Phi) \vdash_{Hmin} q$ **iff** [by Lemma 5.3] $Ind(\vee\Phi) \vdash_{Hmin} \wedge\mathcal{C}(Ind(\vee\Phi) \cup \{q\})$ **iff** [by Lemma 5.2] $Ind(\vee\Phi)' \vdash_{min} WP(Ind(\vee\Phi)), \wedge\mathcal{C}(Ind(\vee\Phi) \cup \{q\}))$ **iff** [by Theorem 4.3] $Ind(\vee\Phi)'$ subsumes $\mathcal{C}(Ind(\vee\Phi)' \cup WP(Ind(\vee\Phi), \wedge\mathcal{C}(Ind(\vee\Phi) \cup \{q\})))$. □

Given a procedure for computing WP, Theorem 5.4 yields a straightforward decision algorithm for $\vdash_{Hmin}$. Because $WP(\vee\Phi, p \wedge q) = WP(\vee\Phi, p) \cup WP(\vee\Phi, q)$, the following theorem indicates an equally straightforward algorithm to compute WP for positive programs and positive clauses.

Theorem 5.5 Let $\vee\Phi$ be a set of positive clauses, let q be a positive clause in $L \setminus L'$, and let $s_1 \vee p_1, ..., s_n \vee p_n$ be all the clauses of $\vee\Phi$ such that each s_i is in $L \setminus L'$, each p_i is in L', and each s_i subsumes q. Then

$$WP(\vee\Phi, q \vee r) \equiv \{\neg p_1 \vee ... \vee \neg p_n \vee r\}.$$

(If n=0 then, by convention, $\neg p_1 \vee ... \vee p_n \vee r = r$).

Proof. Let us first prove that $\vdash \neg p_1 \vee ... \vee \neg p_n$ is the weakest precondition for q. Obviously, $\vee\Phi \cup \{\neg p_1 \vee ... \vee \neg p_n\} \vdash q \vee r$. Therefore, it suffices to demonstrate that for every ground clause x in L', if $\vee\Phi \cup \{x\} \vdash q$ then $\vee\Phi \cup \{x\} \vdash \neg p_1 \vee ... \vee p_n$. Suppose to the contrary, that $\vee\Phi \cup \{x\} \nvdash \neg p_1 \vee ... \vee p_n$, that is, $\vee\Phi \cup \{x\} \cup \{p_1, ..., p_n\}$ is consistent. Because $\{p_1, ..., p_n\}$ subsumes $\{s_1 \vee p_1, ..., s_n \vee p_n\}$, $\vee\Phi \cup \{x\} \cup \{p_1, ..., p_n\}$ had the same models as $\vee\Phi' = \vee\Phi \setminus \{s_1 \vee p_1, ..., s_n \vee p_n\} \cup \{p_1, ..., p_n\}$. In particular, $\vee\Phi'$ is consistent. Let $\mathcal{M}$ be a model of $\vee\Phi'$. Because $\vee\Phi$ is positive, $\mathcal{M}' = \mathcal{M} \cup \{A | A$ is a positive literal of $L \setminus L'\}$ is also a model of $\vee\Phi'$. By the definition of $\{s_1 \vee p_1, ..., s_n \vee p_n\}$, all occurrences of q in $\vee\Phi'$ are in positive clauses involving literals of $L \setminus L'$ not in q, and all these literals are satisfied by $\mathcal{M}'$. Therefore, $\mathcal{M}'' = \mathcal{M}' \setminus \{B | B$ is a literal of $q\}$ is a model of $\vee\Phi'$. $\mathcal{M}'' \models \neg q$. Hence $\vee\Phi' \cup \{\neg q\}$ is consistent, and therefore, $\vee\Phi \cup \{x\} \cup \{p_1, ..., p_n\} \cup \{\neg q\}$ is consistent. From this, we conclude that $\vee\Phi \cup \{x\} \cup \{\neg q\}$ is consistent - a contradiction. Thus $\vee\Phi \cup \{x\} \vdash \{\neg p_1 \vee ... \vee \neg p_n\}$, and $\{\neg p_1 \vee ... \vee \neg p_n\}$ is the weakest precondition for q.

Now, let $\vee\Phi \cup \{x\} \vdash q \vee r$. We get $\vee\Phi \cup \{x \wedge \neg r\} \vdash q$, that is, $\vee\Phi \cup \{x \wedge \neg r\} \vdash \neg p_1 \vee ... \vee \neg p_n$, so $\vee\Phi \cup \{x\} \vdash \neg p_1 \vee ... \vee \neg p_n \vee r$, that is, $\neg p_1 \vee ... \vee \neg p_n \vee r$ is the weakest precondition for $q \vee r$. □

It follows from Theorem 5.5 that if $|\mathcal{C}(Ind(\vee\Phi) \cup \{q\})|$ is polynomial in $|\Phi|$ then $|WP(Ind(\vee\Phi), \mathcal{C}(Ind(\vee\Phi) \cup \{q\}))|$ is polynomial in $|\Phi|$ either. If, moreover, the number of occurrences of negation in $WP(Ind(\vee\Phi), \mathcal{C}(Ind(\vee\Phi) \cup \{q\}))$ is $O(1)$ in $|\Phi|$ then the question $\vee\Phi \vdash_{Hmin} q$ may be solved in polynomial time (proofs are straightforward). More general evaluation of its complexity requires further study.

Example 5.6 Let $L' = \{A, B\}, L = \{A, B, C, D\}, \vee\Phi = \{A \vee B, A \vee C \vee D, B \vee C \vee D\}$, and $q = C \vee D$. We have: $Ind(\vee\Phi) = \vee\Phi$; $\vee\Phi' = \{A \vee B\}$; Φ is a minimal conservative extension of $\vee\Phi'$; $\mathcal{C}(Ind(\vee\Phi) \cup \{q\}) = \{C \vee D\}$; $WP(\vee\Phi, C \vee D) = \{\neg A \vee \neg B\}$; $\mathcal{C}(\vee\Phi' \cup \{\neg A \vee \neg B\}) = B \vee A$; Φ' subsumes $\mathcal{C}(\vee\Phi' \cup \{\neg A \vee \neg B\})$. Hence, $\Phi \vdash_{Hmin} q$. □

Complexity of modeling of definite stratified programs was analyzed in [AB91].

6 Speed-ups

The pigeonhole principle PHP, as it was demonstrated in [CR79], leaves no hope for a complete and fast resolution-based derivation algorithm. Consequently, the resolution-based algorithms we presented so far must perform at least exponentially badly in the worst case. In this section we briefly discuss to what extent the indefinite modeling may be faster than the minimal modeling and vice versa.

First, let us notice that for every anti-chain Φ in the set of sets of variables of L, $\vee\Phi$ is its own indefinite model, and $\wedge\Phi$ is the set of minimal models of $\vee \wedge \Phi$. Therefore, if there are programs for which the indefinite model is smaller than the set of minimal models, there must be equally many others for which the converse is true. This fact suggests that a parallel application of both methods while deciding $\vee\Phi \vdash_{min} q$ may result in a considerable speed-up in some cases, while the slow-down factor, if any, will not exceed 2. For instance, as we noted in Section 3, $|Mod_{min}(\vee PHP^-)|$ is exponential in $|\vee PHP^-|$, while $|Ind(\vee PHP^-)| = |\vee PHP^-|$. Also the converse is true: there are programs $\vee\Phi$ with $|Mod_{min}(\vee\Phi)|$ polynomial in $|\vee\Phi|$ but $|Ind(\vee\Phi)|$ exponential in $|\vee\Phi|$. (Even if both are polynomial, any resolution-based conversion of one onto another may require exponential number of steps). This parallel strategy, however, cannot improve the worst-case performance of these methods, since there are programs $\vee\Phi$ for which both $|Mod_{min}(\vee\Phi)|$ and

$|Ind(\vee\Phi)|$ are exponential in the size of $|\Phi|$. For instance, take a union of a hard example for minimal modeling (PHP) with a hard example for indefinite modeling (any one from Section 4 will do). Moreover, the carrier of $Mod_{min}(\vee\Phi)$ coincides with the carrier of $Ind(\vee\Phi)$ in worst cases, which means that the improvement achieved by parallelization of the two method is limited to some non-worst cases.

Example 6.1 Let $N = 2n - 1$ and let $\Phi = \{\varphi \mid \varphi$ is a set of n distinct literals of $L\}$. Φ is an anti-chain. Obviously, $Ind(\vee\Phi) = \vee\Phi$. Also, $Mod_{min}(\vee\Phi) = \wedge\Phi$. The size of Φ is maximal: $\left(\begin{array}{c} N \\ \lceil \frac{N}{2} \rceil \end{array} \right)$ elements with $\lceil \frac{N}{2} \rceil$ literals each, which totals in $\lceil \frac{N}{2} \rceil \times \left(\begin{array}{c} N \\ \lceil \frac{N}{2} \rceil \end{array} \right) \approx \frac{2^N}{\sqrt{2N}}$ occurrences of literals. □

We conclude this section with rough estimation of the average ratio $R^{D,C}_{avg}$ of the sizes of the indefinite model and of the set of minimal models of a program with N variables. For that purpose, we will use the *Shannon's counting argument*. Let $\mathcal{M}$ be a set of m elements, $\mathcal{N}$ be the set of n shortest sequences of elements of $\mathcal{M}$, and k be the length of the longest element of $\mathcal{N}$. For every $i < k$, $\mathcal{N}$ contains m^i sequences of length i and at most m^k sequences of length k. Therefore,

$$\sum_{i=0}^{k-1} m^i < n \leq \sum_{i=0}^{k} m^i.$$

The total length l of all sequences of $\mathcal{N}$ is

$$\sum_{i=1}^{k-1} i \times m^i + k \leq l \leq \sum_{i=1}^{k} i \times m^i.$$

Elementary calculations show that the average length $\frac{l}{n}$ of an element of $\mathcal{N}$ satisfies

$$log_m \frac{n}{4} < \frac{l}{n} < log_m n$$

From this we conclude that the average length of a sequence in any finite set of n (not necessarily shortest) sequences of elements of $\mathcal{M}$ is greater than $\frac{log_2 n - 2}{log_2 m}$.

Let κ be the number of indefinite models in N variables (which, as we noted above, is the same as the number of sets of minimal models) and let $\gamma = \left(\begin{array}{c} N \\ \lceil \frac{N}{2} \rceil \end{array} \right)$ be the cardinality of the longest antichain with N elements. Because a subset of an anti-chain is an anti-chain, $\kappa \geq 2^\gamma$. Using the Shannon's counting argument we conclude that the average size of a set of minimal models (each of them being a sequence of up to N literals and curly braces { and }) is at least $\frac{log_2 \kappa - 2}{log_2(N+2)} \geq \frac{\gamma - 2}{log_2(N+2)}$. Hence

$$R^{D,C}_{avg} = \frac{1}{\kappa} \times \sum_{i=1}^{\kappa} \frac{d_i}{c_i} \geq \frac{1}{\kappa} \times \frac{\sum_{i=1}^{\kappa} d_i}{\lceil \frac{N}{2} \rceil \times \gamma} =$$

$$= \frac{1}{\lceil \frac{N}{2} \rceil \times \gamma} \times \frac{\sum_{i=1}^{\kappa} d_i}{\kappa} \geq \frac{\gamma - 2}{N \times log_2(N+2) \times \gamma} \approx$$

$$\approx \frac{1}{N \times log_2 N},$$

where c_i's and d_i's are the sizes of all indefinite models and sets of their minimal models. Symmetric argument shows that the average $R^{C,D}_{avg}$ of the converse ratio has the same lower bound. Therefore, on average, the cost of minimal modeling and indefinite modeling are very close to each other. Their closer relationship, for instance, the average ratio $\frac{d_i}{c_i}$ with averaging restricted to cases when the smaller of c_i, d_i is polynomial in N, requires further study.

7 Credit to Others

A version of the right-hand side of criterion (1) for indefinite deductive data bases was proposed (in an equivalent form) by Minker [Min82] under the name of Generalized Closed-World Assumption. Liu and Sunderraman [LS87] introduced indefinite tuples in a relational model of a data base. The credit for stratification is due to Apt, Walker and Blair [ABW88], and Van Gelder [Gel88]. The idea of deciding $p \vdash_{min} q$ by direct verification is Minker's and his collaborators.

8 Acknowledgements

I would like to thank Henrietta Okeke and Johnny Chan for LaTeXing my hieroglyphic manuscript, and Selase Williams for his support of my research activities.

References

[AB91] Krzysztof R. Apt and Howard A. Blair. Arithmetic classification of perfect models of stratified programs. *Fundamenta Informaticae*, 14:339–344, 1991.

[ABW88] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *[Min88]*, pages 89–142. 1988.

[Apt88] Krzysztof R. Apt. Introduction to logic programming. Report TR-87-35, University of Texas, Department of Computer Sciences, Austin, Texas 78712, 1988.

[BT88] Samuel R. Buss and Győrgy Turán. Resolution proofs of generalized pigeonhole principles. *Theoretical Computer Science*, 62:311–317, 1988.

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

[Gel88] A. Van Gelder. Negation as failure using tight derivations for general logic programs. In *[Min88]*, pages 19–88. 1988.

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

[Lif85a] Vladimir Lifschitz. Closed-world databases and circumscription. *Artificial Intelligence*, 27:229–235, 1985.

[Lif85b] Vladimir Lifschitz. Computing circumscription. In *Proceedings of Eight International Joint Conference on Artificial Intelligence*, pages 121–127, Los Angeles, 1985.

[LS87] K.C. Liu and R. Sunderraman. An extension to the relational model for indefinite databases. In *Proceedings of the ACM-IEEE Computer Society Fall Joint Computer Conference, Dallas, TX*, pages 428–435. ACM-IEEE, October 1987.

[Min82] Jack Minker. On indefinite databases and closed world assumption. In *Proceedings of 6-th Conference on Automated Deduction*, Lecture Notes in Computer Science 138, pages 292–308, Berlin, New York, 1982. Springer Verlag.

[Min88] Jack Minker, editor. *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, Los Altos, 1988.

[SS90] Marek A. Suchenek and Rajshekhar Sunderraman. Minimal models for closed world data bases with views. In Zbigniew W. Ras, editor, *Methodologies for Intelligent Systems, 5*, pages 182–193, New York, 1990. North-Holland.

[Suc89] Marek A. Suchenek. A syntactic characterization of minimal entailment. In Ewing L. Lusk and Ross A. Overbeek, editors, *Logic Programming, North American Conference 1989*, pages 81–91, Cambridge, MA, October 16–20 1989. MIT Press.

[Suc90] Marek A. Suchenek. Applications of Lyndon homomorphism theorems to the theory of minimal models. *International Journal of Foundations of Computer Science*, 1(1):49–59, 1990.

[Suc93] Marek A. Suchenek. First-order syntactic characterizations of minimal entailment, domain minimal entailment, and Herbrand entailment. *Journal of Automated Reasoning*, 10:237–263, 1993.

[Suc94] Marek A. Suchenek. Preservation properties in deductive databases. *Methods of Logic in Computer Science An International Journal*, 1:315–338, 1994. An invited paper.

[Suc97] Marek A. Suchenek. Evaluation of queries under the closed-world assumption. *Journal of Automated Reasoning*, 18:357–398, 1997.

[YH85] A. Yahya and L. Henschen. Deduction in non-Horn databases. *Journal of Automated Reasoning*, 1:141–160, 1985.