PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

CAP -- A Three-Phase Query Processing Technique For Indefinite Databases

Shan Chi and Lawrence J. Henschen

EECS Department
Northwestern University
Evanston, Illinois 60208

ABSTRACT

A new method, called the compile-access-prove (CAP) algorithm, is proposed for query processing in indefinite databases. A database is logically represented as a set of clauses among which the non-Horn clauses represent indefinite information. Physically the database intension, containing view definitions, is compiled into access rules and the database extension, containing elementary facts, is stored as relations on disks. Each access rule is a procedure consisting of relational operations. In general, the indefinite elementary facts need to be processed with a theorem prover. By storing all elementary facts (including indefinite ones) into relations, it is possible to replace the theorem proving steps with more efficient relational operations. However, this process changes the semantics of the database. At query time, the related indefinite elementary facts are collected and sent to a thorem prover to recover the original semantics. The CAP algorithm has the following advantages: (a) it is capable of answering queries for recursive indefinite databases, (b) the theorem prover involves only the indefinite facts related to the query, (c) updating the database extension does not require the recompilation of the database, and (d) all techniques developed for Horn databases can be used in this algorithm.

1 INTRODUCTION

In this paper, we present the CAP (compile-access-prove) algorithm for answering queries in indefinite databases. A database is logically represented as a set of clauses and is said to be indefinite if it contains non-Horn clauses. Most of the research in deductive databases and logic programming focused on the computation of definite information represented by Horn clauses. Horn clauses are simple and expressive enough for many applications. They are, however, not expressive enough for those applications involving indefinite information. For example, the rule "if x is in CS department then his office must be in building A or B" is represented by "building(y,A) v building(y,B) :- dept(x,CS), office(x,y)." which is a non-Horn clause.

A conventional approach to indefinite information processing is extending the relation model to accept indefinite data, such as a null value. In this paper, we drop the relational model completely and use the first-order logic because the latter is more general and the theories are well-founded. Nontheless, the database is physically stored as tables and is processed with relational operators. Therefore, the CAP algorithm is compatible with relational databases.

1.1 Non-Horn Clauses versus Null Values

One approach to indefinite information processing is to allow the attribute values to be null. This approach is a natural extension of the relational model. There have been many studies in the construction of formal semantics and theories for null values (Biskup 1981, Codd 1979, Grant and Minker 1981, Lipski 1979, Yahya and Henschen 1985).

The null value approach is sometimes inadequate. Consider a battle command system that receives from an intelligent radar the information "detected(Mig-28) v detected(Mig-29)." The information is not precise but can be very useful. Such information can not be represented in a Horn system like PROLOG. One way to circumvent this problem is using null values: "detected(NULL)." However, it does not carry as much information as a non-Horn clause can. It is also very hard to draw any conclusions from the information containing null values. It is shown in this paper that definite information can be deduced from some indefinite information represented by non-Horn clauses.

1.2 Non-Horn Clauses versus Negative Subgoals

Transforming a non-Horn clause into a Horn clause with negative subgoals is not always desirable. For example, "detected(Mig-28) v detected(Mig-29)" will probably be transformed into "detected(Mig-28) :- not detected(Mig-29)." in PROLOG. A query "detected(Mig-28)" will then be answered "yes" while "detected(Mig-29)" will be answered "no." Therefore the interpretation is biased.

A more serious problem is that the logic programs (or deductive databases) become unstratified after such transformations. The query answers for this class of programs are not even well-defined. In general, the semantics of a non-Horn clause is changed by the transformation. If the non-Horn clause represents indefinite information then the transformation is inappropriate since it distorts the original meaning of the clause.

From the above discussion, we conclude that processing non-Horn clauses is sometimes unavoidable or even desirable. An algorithm is proposed in this paper to make the processing more manageable in the context of deductive databases. A brief review of the first order logic is given in section 2. The compilation approach is introduced in section 3. Minimal conditional answers and relevent theorems are derived in section 4. The CAP algorithm is presented in section 5. Some comparisons with related works are in section 6.

2 PRELIMINARIES

The first-order logic considered in this paper is function-free and quantifier-free. An atom is written as P(t1,...,tn) where P,t1,...,tn are symbols. P is the predicate symbol of the atom. Each ti is said to be a variable if it starts with u,v,w,x,y or z, and a constant otherwise.

A literal is an atom (positive literal) or an atom preceded by a negation sign - (negative literal). A clause is a disjunction of literals, written as a literal sequence (with the "or" connective "v" omitted). The variables in a clause are assumed to be universally quantified. A clause is positive if all literals in it are positive. Similarly, it is negative if all literals in it are negative. A clause is Horn if there is no more than one positive literal in the clause; otherwise, it is non-Horn. A unit clause is a single-literal clause. A clause is ground if it contains no variables. A subclause of a clause C is a disjunction of some literals in C.

We use a set of clauses to mean a conjunction of clauses. A set of clauses is Horn if every clause in it is Horn; otherwise, it is non-Horn. The Herbrand universe of a set S of clauses is the set of all the constants in S. The atom set of S is the set of ground atoms of the form P(t1,...,tn) for all n-place predicates P, where t1,...,tn are elements of the Herbrand universe of S. A ground instance of a clause C of a set S of clauses is a clause obtained by replacing variables in C by members of the Herbrand universe of S.

An interpretation I of a set S of clauses is a subset of the atom set of S. I is said to satisfy an atom A if A is in I; otherwise, it is said to falsify A. I falsifies (satisfies) the literal -A if it satisfies (falsifies) A. An interpretation satifies a ground instance of a clause if it satisfies at least one literal in that instance. It satisfies a clause if it satisfies all the ground instances of that clause. It is a model of S if it satisfies all the clauses in S. It is minimal if none of its proper subsets is a model of S. A set of clauses is consistent (satisfiable) if and only if it has a model. It is inconsistent (unsatisfiable) if it is not consistent.

A deduction tree of a clause C is a binary tree whose nodes are clauses such that each parent node is the resolvent of its two children and the root node is C. An S deduction tree, where S is a set of clauses, is one whose leaves are all from S. From the binary resolution theory, we know if there exists an S deduction tree of a clause C then S derives C and C is derivable from S.

3 QUERY COMPILATION

A database DB is the union of two disjoint sets of clauses, IDB and EDB, representing the intensional and the extensional database, respectively. The EDB consists of all the ground clauses, representing facts, and the IDB consists of other clauses, representing rules and view definitions. The EDB is further divided into two disjoint subsets, DEDB and IEDB, containing Horn and non-Horn clauses, respectively. The IEDB clauses can not be directly stored into relations. Instead each literal in the IEDB is stored as a tuple in a disk relation and is interpreted as a unit clause. The set of all such literals is denoted as IEDB'. On the other hand, an IEDB clause simply contains pointers to the tuples that represent the literals.

The following convention is adopted in this paper: Predicate symbols P, Q and R are used in IDB literals. A and B are used in EDB literals. A predicate symbol, such as Q or A, can be used to denote a literal. Clauses are represented by C, D and E. For example, QC represents a clause with an IDB literal Q and a subclause C. An EDB (IEDB, DEDB) literal is simply a literal occurring in the EDB (IEDB, DEDB) while an IDB literal is an instance of a positive literal in IDB. The union of a clause C and a set of clauses S, denoted as C U S, is the set obtained by adding the clause C to S (therefore, the brackets around C are omitted).

3.1 Horn Database Compilation

The simplest approach to query processing is to treat each query as a theorem to be proved, using a theorem prover (Figure 1). The theorem prover resolves clauses from the IDB and the EDB, disregarding the fact that EDB clauses are stored on disks. The performance of such systems is usually unacceptable due to the impedance mismatch between the theorem prover and the disk accesses. The performance can be improved significantly by using the techniques developed for conventional database systems. Based upon this idea, Reiter (1978) proposed the compilation approach to query processing (Figure 2).

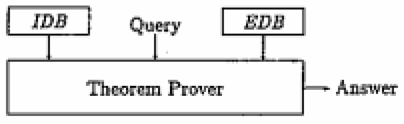

Figure 1: The Theorem Proving Approach

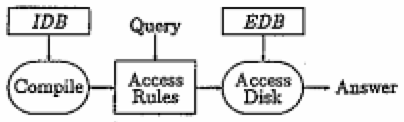

Figure 2: The Compilation Approach

In the compilation approach, the IDB is compiled into access rules. Each access rule represents a sequence of relational operations. (In practice, the access rules are stored as programs containing database calls. In this paper, they are represented as clauses for meta-theoretical derivations.) At query evaluation time, the related access rules are used to trigger corresponding database operations. There are at least two advantages of this approach: (a) relational operators are much more efficient than a theorem prover, and (b) due to the separation of compilation and query evaluation, the access rules can be well optimized at compile time. The compilation of Horn databases is based on the following:

<u>Theorem 3.1</u> If DB is a Horn database, then for any clause C derivable from DB, there exists a clause D such that IDB derives D and D U EDB derives C.

<u>Proof</u> If DB is Horn and derives C then there exists a DB deduction tree T of C. A new deduction tree T' is obtained from T by moving all the EDB clauses to the root. An IDB deduction tree of D is obtained by removing all the resolutions with the EDB clauses from T'. Therefore IDB derives D and D U EDB derives C. QED

<u>Example 3.1</u> If IDB contains -AP and -BP while EDB contains AB then DB derives P. However IDB does not derive any clause which, together with EDB, derives P.

The above example shows if the EDB is non-Horn then there might exist a clause derivable from the DB but not derivable by applying an access rule to the EDB. Therefore, the answers derived from the access rules and the EDB are not complete.

3.2 Indefinite Database Compilation

Example 3.1 shows the problem in directly applying the compilation technique to indefinite databases. Henschen and Park (1986) proposed the solution in which the IEDB clauses are included in the compilation (Figure 3). Therefore, at query time only Horn clauses remain in the EDB. This solution has several drawbacks: (a) updating the EDB requires a database recompilation, (b) the compiler generates too many access rules, each representing a sequence of database operations at query time, and (c) recursive databases can not be compiled.

The EDB represents the current state of the database, which varies with time. If every update triggers a compilation process, then the database is limited to retrieval-only applications.

When the database contains recursive rules, the non-Horn clauses may resolve with the recursive rules indefinitely. In general, the approach in (Henschen and Park 1986) can not deal with recursive indefinite databases. In the case when there are no recursive rules, the number of access rules generated still increases exponentially with the number of IEDB literals, as illustrated in the following example.

<u>Example 3.2</u> Consider the database with the following clauses:
C1 = R1(x,y)...-A(z,w)
C2 = R2(x,y)...-A(z,w)
...
C10= R10(x,y)...-A(z,w)
C11= A(a1,b1)...A(a10,b10)

To derive all the non-Horn clauses with R1 literals, we use C11 to resolve with C1. There are 10 choices of the literal to resolve upon. The resolvent can further resolve with one of C1,...,C10. The process continues until all the positive A literals are resolved away. More than 10^{10} different clauses (access rules) containing R1 can be derived. To answer a query R1(a,x) will then trigger that many access rules.

The CAP algorithm (Figure 4) uses a different approach in compiling indefinite databases. Each IEDB literal entered into the database is stored as a tuple in the corresponding disk relation. By applying relational operators to the relations at query time, each tuple is treated as if it were a unit clause. Therefore, the semantics of the database is temporarily distorted to facilitate the storage and processing of these indefinite tuples.

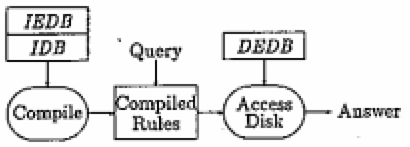

Figure 3: The Henschen-Park Algorithm

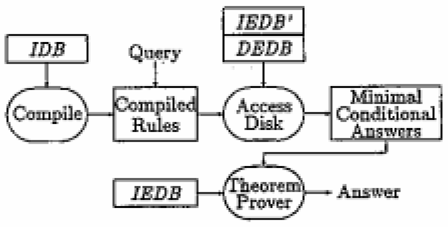

Figure 4: The CAP Algorithm

To recover the semantics of the database, in the third phase the original IEDB together with the minimal conditional answers (defined in the next section) is brought to memory and fed into a theorem prover. In this phase, only ground clauses are involved and they are relatively few in number. Thus the theorem proving process at query time should be fast enough in general.

4 MINIMAL CONDITIONAL ANSWER

In this section, we shall derive the theorems on which the CAP algorithm is based. To simplify the proofs of the theorems, we have the following assumption about the database: (a) the database contains no negative clauses, (b) every clause in the EDB is positive and minimal, (c) the predicate symbol of an IDB literal is distinct from that of any EDB literal, and (d) clauses are range-restricted, i.e., any variable occurring in a positive literal must also occur in a negative literal in the same clause.

A clause C subsumes a clause D if there exists a substitution θ such that $C\theta \subseteq D$. C is said to strictly subsume D if $C\theta \subset D$. A clause is minimal with respect to a set S if S does not derive any clause strictly subsuming the clause. It is locally minimal if it is minimal with respect to the set from which it is derived. It is globally minimal if it is minimal with respect to the database.

The CAP algorithm derives query answers according to the generalized closed world assumption (GCWA) (Minker 1982) which assumes a ground atom to be false only if it does not occur in any globally minimal positive clause. For example, if DB contains only P(a)P(b) and P(a), then P(b) is assumed to be false since the only globally minimal clause is P(a).

The head of a clause is defined to be the subclause with all positive IDB literals. The tail is the remaining subclause. An access rule is a clause derivable from DB with at least one positive IDB literal and no negative IDB literals. It is called access rule because its negative subclause can be used to access the database, as will be shown later. We use ACC to denote the set of all the access rules. ACC* denotes the set of their ground instances.

A positive ground clause can be used as a subscript for selection. For example, S_C means the set of clauses from S whose heads (or tails) contain C. $S_{C1,\dots,Cn}$ denotes the union of all the selections $S_{C1},\dots,S_{Cn}$. If C is not ground, then S_C denotes the union of the selections using the ground instances of C. Also $\leq C$ ($<C, \geq C, >C$) denotes all the heads or tails subsuming (strictly subsuming, subsumed by, strictly subsumed by) C.

<u>Definition</u> POS (MPOS) denotes the set of all (minimal) positive clauses derivable from DB. POS' (MPOS') denotes the set of all (minimal) positive clauses derivable from ACC U EDB.

Note that POS and MPOS imply the theorem proving approach to query answering while POS' and MPOS' imply the compilation approach. We know POS' and MPOS' are subsets of POS and MPOS, respectively (it is easy to show that POS' is a proper subset of POS). In order to have complete query answers from the compilation approach, we must have MPOS=MPOS'.

<u>Definition</u> A query is an atom. It is open if it contains variables; otherwise it is closed. The answer to a closed query Q is <u>true</u> if Q is in MPOS, i.e., if DB derives Q. It is <u>possible</u> if Q is not in MPOS but occurs in a clause in MPOS. Otherwise the answer is <u>false</u>.

A conjunctive query can be considered as a query and an IDB rule. For open queries, we collect all the ground insances which are answered true and those that are answered possible into two sets and call them definite answers and indefinite answers.

<u>Lemma 4.1</u> If M is a minimal model of a set S of clauses and Q is in M then there exists a ground instance QC of a clause in S such that M falsifies the subclause C.

<u>Proof</u> Suppose this is not true. Let QC1,...,QCn be all the ground instances containing Q of the clauses in S and M satisfies each of C1,...,Cn. Then M-{Q} is still a model of S---contradicting the minimality of M. QED

<u>Theorem 4.2</u> MPOS=MPOS'.

<u>Proof</u> We only need to prove MPOS⊆MPOS', i.e., every minimal positive clause C derivable from DB is also derivable from ACC U EDB. Obviously ACC U EDB does not derive any clause strictly subsuming C because C is minimal. Suppose ACC U EDB does not derive C. Then ACC U EDB U -C is consistent and has a minimal model M. Note that M falsifies C. We shall show that M is also a model of DB. Suppose it is not, then there exists a ground instance D of an IDB clause falsified by M. Let D=P1...Pm-Q1...-QnD', where D' is the subclause with negated EDB literals. In order to falsify D, M must contain all the Q atoms and no P atoms. Since M is a minimal model of ACC U EDB U -C, for each Qi there exists a ground instance of an access rule QiEi such that M falsifies Ei (Lemma 4.1). Resolving these ground instances with D will derive P1...PmE1...EnD, which is falsified by M and is a ground instance of an access rule (since all the IDB literals are positive)---contradicting that M is a model of ACC. Therefore M must be a model of DB and must satisfy C (as DB derives C)---a contradiction. Hence, ACC U EDB U -C must be inconsistent and ACC U EDB derives C. QED

<u>Definition</u> If C-A1...-An-B1...-Bm is a clause in ACC* such that C is the head, each A literal comes from IEDB' and each B literal, from DEDB, then C-A1...-An is said to be a minimal conditional answer. MICA denotes the set of all minimal conditional answers of DB.

<u>Theorem 4.3</u> (Completeness Theorem) If C∈MPOS then either C∈EDB or MICA U IEDB derives C.

<u>Theorem 4.4</u> (Soundness Theorem) If MICA U IEDB derives a locally minimal clause C then C∈MPOS.

The proofs of all the theorems in this paper can be found in Chi (1988). Phase 1 in the CAP algorithm involves a theorem prover with the IDB to generate the access rules. In phase 2, the access rules are applied to the DEDB to derive the related minimal conditional answers. In phase 3, this subset of MICA and the IEDB are used to derive the query answers. We have shown that the answers derived are sound and complete.

5 THE CAP ALGORITHM

In practice, there is no need to generate the entire MICA to answer a query. The following observations, based upon the assumptions about the database, are used to choose the relevent subset of MICA for processing queries: (a) Each clause in MICA contains a nonempty head and a negative tail. (b) Each clause in IEDB is minimal and positive and contains only IEDB literals. (c) If C∈MPOS contains only EDB literals then C is a clause from the EDB. (d) If C∈MPOS contains an IDB literal Q, then MICA U IEDB derives C and at least one clause from MICA containing Q is involved in the deduction. (e) If MICA U IEDB derives a positive clause C and Q is an IDB literal in a clause involved in the deduction, then Q is also in C.

5.1 EDB Query

Observation (c) makes processing queries with only EDB literals very simple. Consider a closed query Q(a). According to (c), if A(a)∈$\mathrm{MPOS}_{\geq A(a)}$ then A(a)∈EDB. Note that the EDB is physically stored as tables with marked indefinite tuples. If the tuple A(a) does not exist then the answer is false; else if the tuple is marked indefinite then the answer is possible; else the answer is true.

5.2 IDB Query

If the query is an IDB literal Q(a) then the answer depends upon whether $\mathrm{MICA}_{Q(a)}$ U IEDB derives Q(a).

Theorem 5.1 For any positive IDB literal Q, Q∈MPOS if and only if MICA_Q U IEDB derives Q.

This is a direct result from Theorems 4.3 and 4.4 by using the observations. In the following example, an indefinite recursive database is used.

Example 5.1 Let DB contain the following clauses:

```
C1 = R(x,y)-A(x,y)
C2 = R(x,z)-R(x,y)-A(y,z)
C3 = A(a,1)
C4 = A(a,3)
C5 = A(2,5)
C6 = A(4,7)
C7 = A(5,8)
C8 = A(6,9)
C9 = A(8,10)
C10= A(1,11)A(11,12)
C11= A(a,2)A(1,4)A(3,6)
C12= A(a,2)A(4,8)A(7,10)
C13= A(1,4)A(9,8)A(6,10)
```

Relation A		
a	1	
a	2	*
a	3	
1	4	*
1	11	*
2	5	
3	6	*
4	7	
4	8	*
5	8	
6	9	
6	10	*
7	10	*
8	10	
9	8	*
11	12	*

The physical storage of relation A is also shown. A tuple is marked * if it comes from the IEDB, i.e., if it is indefinite. Let the query be R(a,10). $\mathrm{MICA}_{R(a,10)}$ is obtained by applying the transitive closure algorithm (Chi and Henschen 1988) to relation A:

```
C14= R(a,10)-A(1,4)-A(7,10)
C15= R(a,10)-A(1,4)-A(4,8)
C16= R(a,10)-A(a,2)
C17= R(a,10)-A(3,6)-A(9,8)
C18= R(a,10)-A(3,6)-A(6,10)
```

$\mathrm{MICA}_{R(a,10)}$ U IEDB derives R(a,10) as shown below:

```
11,12,14 C19=R(a,10)A(a,2)A(3,6)A(4,8)
11,15,19 C20=R(a,10)A(a,2)A(3,6)
13,17,20 C21=R(a,10)A(a,2)A(1,4)A(6,10)
11,18,21 C22=R(a,10)A(a,2)A(1,4)
12,14,22 C23=R(a,10)A(a,2)A(4,8)
15,22,23 C24=R(a,10)A(a,2)
16,24    C25=R(a,10)
```

The numbers preceding each clause are the clauses involved in the resolution. Note that the definite fact R(a,10) is derived from some indefinite clauses. The query answer is true by Theorem 5.1.

5.3 Redundancy Elimination

Theorem 5.1 shows that the information in MICA_Q U IEDB is sufficient for computing MPOS_Q. However, for a non-unit head C the information in MICA_C U IEDB is in general insufficient for computing MPOS_C. We need a relevant subset of MICA to generate the query answers and some redundant resolutions should be avoided. The theorem developed below is used for such purpose.

Definition A set S of clauses is said

to be saturated if every clause it derives is subsumed by a clause in the IEDB.

Lemma 5.2 For any set of minimal conditional answers $MICA_C$, if $MICA_C$ U IEDB is saturated then every minimal model of the IEDB satisfies $MICA_C$.

Theorem 5.3 If $MICA_C$ U IEDB is saturated then every minimal positive clause derivable from MICA U IEDB is also derivable from MICA U IEDB - $MICA_C$.

The proof follows directly from Lemma 5.2 by showing that every minimal model of MICA U IEDB is also a model of MICA U IEDB - $MICA_C$. The CAP algorithm uses Theorem 5.3 to eliminate unnecessary resolutions in the process of finding all query answers. The steps are as follows:

1. Compute the ACC by resolving clauses in the IDB.
2. Let Q be the query and apply the access rules ACC_Q to disk relations to generate $MICA_Q$.
3. If $MICA_Q$ U IEDB derives Q then answer true and stop.
4. If $MICA_Q$ U IEDB is not saturated then answer possible and stop.
5. If there are no more access rules containing Q then answer false and stop.
6. Apply one access rule, say ACC_{QC}, to disk relations to generate $MICA_{QC}$.
7. If $MICA_{QC}$ U IEDB is saturated then go to 5.
8. If every positive clause derivable from $MICA_{QC}$ U IEDB is subsumed by a clause derivable from $MICA_{\leq C}$ U IEDB, then go to 5; else answer possible and stop.

Note that the saturation is tested by using a theorem prover. If the IDB contains only Horn clauses, then the first 4 steps are sufficient for finding all the answers.

6 RELATED WORKS AND CONCLUSION

Henschen and Park (1986) first introduced the compilation approach to non-Horn (indefinite) databases. The number of access rules generated in their approach increases exponentially with the number of IEDB literals. The theorem prover in the CAP algorithm can also involve many resolutions. However, the process is taking place in memory with only ground clauses involved, therefore is much faster.

Grant and Minker (1983) proposed a query processing method under the GCWA. It requires the generation and storage of all database models. Yahya and Henschen (1985) developed a deductive approach to query answering under the extended GCWA. A large group of clauses need to be proved at query time with this approach. Bossu and Siegel (1985) proposed an algorithm for answering queries based upon the so-called subimplication. The saturation (theorem proving) algorithm is applied to the entire database for answering a query. In these approaches, the physical storages of the IDB and EDB are not differentiated. Therefore, the possibility of replacing some deductions with relational operations was not explored.

We presented the CAP algorithm for processing queries in indefinite databases. The possibility of using a relational database and compilation techniques to process indefinite information is explored. The improvement in performance is based upon the following: (a) relational operators instead of a theorem prover is used for accessing disk relations, (b) redundant disk accesses are reduced to the minimum by applying the redundancy removal theorem and by delaying the indefinite information processing till the theorem proving phase, (c) only related indefinite information needs to be processed by the theorem prover, and (d) the theorem proving is taking place in main memory and involves only ground clauses. As in reality very few applications require deductions from a large amount of inter-related indefinite information, so the CAP algorithm should be efficient enough for practical use.

Our major contributions are (a) eliminate the need of using a theorem prover over the entire indefinite database, (b) remove the exponential growth of the number of disk accesses, and (c) maintain the compatibility with relational databases and recursive query processing techniques (Bancilhon and Ramakrishnan 1986, Chang 1981, Henschen and Naqvi 1984).

REFERENCES

1. Bancilhon, F. and Ramakrishnan, R., "An amateur's introduction to recursive query processing strategies" ACM SIGMOD Conference on Management of Data, (1986).

2. Biskup, J., "A formal approach to null values in database relations" Advances in Data Base Theory 1, H. Gallaire, J. Minker, and J.M. Nicolas, Eds., Plenum Press, New York, (1981), pp. 299-341.

3. Bossu, G. and Siegel, P., "Saturation, nonmonotonic reasoning and the closed-world assumption" Artificial Intelligence 25, (1985), pp. 13-63.

4. Chang, C.L., "On evaluation of queries containing derived relations" Advances in Data Base Theory 1, H. Gallaire, J. Minker, and J.M. Nicolas, Eds., Plenum Press, New York, (1981), pp. 235-260.

5. Chi, S., "A three-phase query processing technique for indefinite databases" Ph.D. dissertation, Northwestern University, (1988).

6. Chi, S. and Henschen, L.J., "Recursive query answering with non-Horn clauses" Conference on Automated Deduction, Argonne National Lab., (1988).

7. Codd, E.F., "Extending the database relational model to capture more meaning" ACM TODS 4, 4, (1979), pp. 339-434.

8. Grant, J. and Minker, J., "Answering queries in indefinite databases and the null value problems" University of Maryland, College Park, (1981).

9. Henschen, L.J. and Naqvi, S., "On compiling queries in recursive first order databases" JACM 31, 1, (1984), pp. 47-85.

10. Henschen, L.J. and Park, H., "Compiling queries in indefinite deductive databases under the GCWA" Ph.D. dissertation, Northwestern University, (1986).

11. Lipski, W. Jr., "On semantic issues connected with incomplete information databases" ACM TODS 4, (1979) pp. 262-296.

12. Minker, J., "On indefinite database and the closed world assumption" Lecture Notes in Computer Science 138, Springer Verlag, (1982), pp. 292-308.

13. Reiter, R., "Deductive question answering on relational data bases" Logic and Databases, H. Gallaire and J. Minker, Eds., Plenum Press, New York, (1978), pp. 149-177.

14. Yahya, A. and Henschen, L.J., "Deduction in non-Horn databases" Journal of Automated Reasoning 1, 2, (1985), pp. 141-160.