INDEFINITE AND GCWA INFERENCE IN INDEFINITE DEDUCTIVE DATABASES

Lawrence J. Henschen and Hyung-Sik Park

Northwestern University
Department of EECS
Evanston, Illinois 60201

ABSTRACT

This paper presents several basic results on compiling indefinite and GCWA(Generalized Closed World Assumption) inference in IDDB(Indefinite Dedutive Databases). We do not allow function symbols, but do allow non-Horn clauses. Further, although the GCWA is used to derive negative assumptions, we do also allow negative clauses to occur explicitly. We show a fundamental relationship between indefiniteness and indefinite inference. We consider three representation alternatives to separate the CDB(Clausal DB) from the RDB(Relational DB). We present the basic ideas for compiling indefinite and GCWA inference on CDB and evaluating it through the RDB. Finally, we introduce decomposition theorems to evaluate disjunctive and conjunctive queries.

I INTRODUCTION

The reader is assumed to be familiar with the logic approach to databases, especially with the concept of query compilation relative to an intensional database (IDB). This paper presents some basic results on compiling indefinite and GCWA inference, i.e. generating queries that will correctly answer questions like, "Is a ground formula q indefinite?" and "Can we assume a ground atom q to be false?", in an IDDB (Indefinite Deductive Database) under the GCWA(Generalized Closed World Assumption)[Minker, 1982]. The notion of indefinite and GCWA inference can be defined by using the semantics of minimal model: A ground formula q is indefinite with respect to IDDB iff it is true in some minimal model of IDDB and false in some minimal model of IDDB. Such a q is false with respect to IDDB under the GCWA iff it is false in every minimal model. An IDDB is a deductive database which does not allow function symbols, but does allow negative and non-Horn clauses in addition to Horn clauses. Since the volume of negative facts may be too huge to be explicitly represented, deductive databases have traditionally treated negative information implicitly. While the negation of a ground atom can be assumed to be true straightforwardly by negation as (finite) failure[Clark, 1978][Reiter, 1978-b] in a Horn database, a generalized metarule [Bossu and Siegel, 1985][Minker, 1982] must be used in a DB with non-Horn clauses. These metarules are much more difficult to compute. We introduce a compiling technique to help overcome the computational problems and also to separate the deduction from the data retrieval.

Three typical methods for dealing with the GCWA have been recently reported. First, Grant and Minker(GM)[Minker and Grant, 1981] developed an algebraic method which can answer a negative query in a generative database under the GCWA. Second, Yahya and Henschen(YH)[Yahya and Henschen, 1985] developed a deductive method which can answer a negative query in a non-Horn database under the extended GCWA. Third, Bossu and Siegel(BS)[Bossu and Siegel, 1985] developed a deductive method which can answer a query by subimplication. Subimplication is a generalization of GCWA that handles databases having no minimal model. It reduces to GCWA if the database has no function symbols. However, those methods have the following weak points: GM's method requires the system to generate all data base models. YH's method requires the query to be decomposed into several subqueries which must all be proved at query time. BS's requires many subsumption tests in the computation of characteristic clauses and characteristic formulas[Bossu and Siegel, 1985]. None of these methods seems practical enough for application to large databases.

A major difficulty is that ordinary resolution applied to an IDDB cannot distinguish between ground atoms that are indefinite and those that can be assumed false under GCWA. This will be illustrated by an example in section 2. In order to overcome this problem we will investigate the relationships between indefiniteness and indefinite inference in an IDDB. We will also discuss certain tradeoffs among three schemes for representing explicit negative data. We will then develop indefinite and GCWA inference engines by introducing a compilation technique for IDDBs similar in spirit to compilation for Horn databases [Chang, 1981] [Henschen and Naqvi, 1984] [Reiter, 1978-a].

II INDEFINITE DEDUCTIVE DATABASES AND THE GCWA

A deductive database is an extension of the proof theoretic relational DB [Reiter, 1984] in which new facts may be derived from the set of explicit facts, called the EDB (Extensional DB), by using the deductive general laws, called the IDB (Intensional DB). There are two kinds of deductive databases relevant to our study, DDDB (Definite Deductive Databases) and IDDB (Indefinite Deductive Databases), and their properties are quite different. A DDDB allows only function-free Horn definite clauses, while an IDDB allows function-free indefinite (non-Horn) clauses as well. For an extensive survey of deductive databases, refer to [Gallaire, Minker, and Nicolas, 1984]. In this paper, we use the notations "DB |- q" for "q can be derived from DB", "DB |X q" for "q cannot be derived from DB", "DB |-GCWA -q" for "-q can be assumed by GCWA", and "DB |XGCWA -q" for "-q cannot be assumed by GCWA". A clause is written as a list of literals without commas.

Since a typical database will have vast amounts of negative information, such information should be implicitly represented. To this end we distinguish two parts to a database - those formulas represented explicitly, e.g., IDDB or DDDB, and those parts not represented explicitly, for example, the negative facts that are to be assumed. Reiter [Reiter, 1978-b] developed the <u>closed world assumption</u> (CWA), for implicit negative information in a DDDB. CWA says that a negative ground unit clause, -p, can be assumed to be true if p cannot be derived from DDDB. However, CWA leads to inconsistencies when used with an IDDB. For example, let IDDB = {p q}. IDDB |X p. Hence IDDB |-CWA -p. Similarly for q. However, IDDB + {-p, -q} |- nil. Minker [Minker, 1982] suggested the <u>semantic</u> and <u>syntactic</u> <u>definitions</u> of the <u>GCWA</u>, which can be used to handle negative information implicitly in an IDDB, and showed that they are equivalent. It is based on the concept of minimal model. An interpretation is specified by listing the ground atoms that are to be true. A <u>minimal model</u> is a model of a database such that no proper subset of the true atoms still satisfies the database.

<u>Semantic Definition of GCWA</u>:
-P(<u>c</u>) can be assumed to be true with respect to IDDB iff P(<u>c</u>) is not in any minimal model of IDDB.

<u>Syntatic Definition of GCWA</u>:
-P(<u>c</u>) can be assumed to be true with respect to IDDB iff P(<u>c</u>) v C is not provable from IDDB for any C in S, where S is a set of all purely positive (possibly empty) clauses not provable.

<u>Example 1</u>
Let DB = {p q, r, -s}. Then, there are two minimal models of DB: M1 = {p, r} and M2 = {q, r}. Consider the following queries.

Q1 = r. This query is <u>true</u> in DB.
Semantic justification:
r is in M1 and M2.
Syntactic justification:
DB + (-r) |- nil,
and DB is consistent.
That is, DB |- r.

Q2 = p. This query is <u>indefinite</u> in DB.
Semantic justification:
p is in M1, but not in M2.
Syntactic justification:
DB + (-p) |X nil, and DB + (p) |X nil.
That is, DB |X p, DB |X -p,
and DB |XGCWA -p.

Q3 = s. This query is <u>provably false</u> in DB.
Semantic justification:
s is not in M1 or in M2.
Syntactic justification:
DB + (-s) |X nil, but DB + s |- nil.
That is, DB |- -s.

Q4 = t. This query can be <u>assumed false</u> in DB by <u>GCWA</u>.
Semantic justification:
t is not in M1 or in M2.
Syntactic justification:
DB + (-t) |X nil, and DB + t |X nil.
DB |X t C for any positive or empty C.
That is, DB|X t and DB |X -t,
but DB |-GCWA -t.

As shown in example 1, one difficulty in evaluating a query in an IDDB under the GCWA is that there is no difference between the indefinite and false cases when the ordinary approach (of trying to prove the query or its negation) is applied, as in Q2 and Q4. The difference arises only when the query literal is considered in conjunction with additional positive parts. Hence, we need to develop specialized inference engines for indefinite and GCWA answers.

III INDEFINITENESS AND INDEFINITE INFERENCE

We introduce the basic notions for analyzing indefiniteness as follows: PIGC is the set of minimal positive indefinite ground clauses implied by IDDB, where the notion of "minimal set" means that clauses in PIGC cannot be properly subsumed by any positive ground clause derivable from IDDB. If q is a ground atom, then PIGC[q] consists of members of PIGC that contain q. Recall that q is indefinite if it is true in some minimal model of DB and false in some minimal model of DB. We introduce three auxilliary functions.

Definition
True[q] = t if DB |- q
  f otherwise

Definition
Indef[q] = t if q is indefinite in DB
  f otherwise

Definition
GCWA[q] = t if True[q] = f
  and Indef[q] = f
  f otherwise

Notice that the above makes no distinction between provably false and false by assumption. Our main purpose is to determine which of the three values q has, true, false or indefinite. If it was important for a user to distinguish between the two false cases, additional tests would have to be made.

Example 2
Let DB1 = {p, p q}, and DB2 = {p q, r s}.
In DB1, PIGC[p] = {}, Indef[p] = f and GCWA[r] = t.
In DB2, PIGC[p] = {p q}, Indef[p] = t and GCWA[p] = f.

Lemma 1 (Minker's Lemma[Minker, 1982])
Every minimal model of C is a minimal model of CP, where C = CP union CNP, CP denotes a set of all positive clauses provable from DB, and CNP denotes a set of clauses provable from DB each of which includes at least one negative literal.

The lemma 1 says that Mcp = Mc rather than Mcp < Mc, where Mcp and Mc mean sets of minimal models of CP and C, respectively.

Lemma 2
C = q v C' is in PIGC iff q is indefinite in DB.

Theorem 1 (Indefiniteness Theorem)
Indef[q] = t if PIGC[q] is not empty
  f otherwise

Corollary 1.1
GCWA[q] = t if True[q] = f
  and PIGC[q] is empty
  f otherwise

The Indefiniteness Theorem says that PIGC characterizes the indefiniteness of an IDDB. It provides the theoretical basis for developing the indefinite inference. That is, it seems unavoidable to consider PIGC in some form or other for answering Indef[q]. However, since it is obviously unfeasible to derive PIGC in general, we should develop an appropriate mechanism to handle only PIGC[q], that is, the portion of PIGC relevant to the query at hand.

IV COMPILATION AND REPRESENTATION ALTERNATIVES

The goal of compiling is to separate the deductive process from the data retrieval process. For the problem at hand, namely determining PIGC[q] for a generic q, we need to find which resolvents of IDB clauses could lead to positive ground clauses containing q when data from the database is taken into consideration. Further, such a positive ground clause must not be subsumed by another positive clause. A simple example will illustrate the basic approach. Suppose the database contained the clauses

-P(x) -Q(y) -S(z) R(x,y,z) T(x,y,z),
-O(w) S(w) U(w), and
-M(v) T(v,10,u) U(u),

where P, Q, M and O are simple relations stored in the EDB. Suppose we had the query, "Is R(JOHN,10,TOYS) indefinite or false?" The resolvent, -P(x) -Q(y) -O(z) R(x,y,z) T(x,y,z) U(z), could produce a positive ground clause containing R(JOHN,10,TOYS) if the appropriate data were in the relations P, Q and O. On the other hand, if JOHN were in M, the third clause would derive a positive ground clause subsumming the one containing R(JOHN,10,TOYS); that is, R(JOHN,10,TOYS) T(JOHN,10,TOYS) U(TOYS) would not be in PIGC after all. Thus, we may answer false or indefinite after retrieving the appropriate data from P, Q, O and M and testing the resulting clauses for subsumption. Notice that if the third clause had contained T(v,25,u) instead, there would be no possibility for subsumption, and R(JOHN,10,TOYS) T(JOHN,10,TOYS) U(TOYS) would definitely be in PIGC. As with regular query compilation, the above kinds of analyses can be carried out on the basis of generic values for the attributes of R, and the deductive analysis separated from the data retrieval.

In order to carry out the above deductive analysis, we identify certain sets of clauses. The IIDB(Indefinite IDB) consists of indefinite general clauses.

The DIDB(Definite IDB) consists of definite general clauses. The IEDB(Indefinite EDB) consists of non-Horn ground clauses. The DEDB(Definite EDB) consists of positive unit ground clauses. As will be seen below, the precise details of compiling will depend on whether negative clauses and clauses in IEDB are used at compile time or are to be handled at retrieval time. Therefore, we call CDB (Clausal Database) the set of clauses that are to be used in the compile phase. Then, NH[q] is the set of minimal non-Horn clauses which contain a positive occurrence of predicate q and are derivable from the CDB. A clause C1 is said to potentially subsume another clause C2 iff there is a positive subclause of C1 obtained by deleting the negative EDB literals and the positive literals for which there are corresponding negative EDB relations that subsumes a ground instance of a positive subclause obtained similarly from C2. PSUB[nhi] denotes a set of clauses which potentially subsume a clause, nhi, in NH[q] and are derivable from CDB. The sets NH[q] and PSUB[nhi] can be used to generate PIGC[q] if the negative data is used properly.

To see why the negative data plays a crucial role, we consider three representation alternatives. We assume that #DEDB >> #-p > #-C > #IEDB > #DIDB >> #IIDB, where #X denotes the number of clauses or relational tuples in each database X, #-p the number of negative ground unit clauses explicitly occuring in DB, and #-C the number of negative nonunit clauses explicitly occuring in DB.

REPRESENTATION 1
(a) CDB = IIDB + DIDB
+ negative nonunit clauses
(b) RDB = IEDB + DEDB
+ negative unit ground facts

REPRESENTATION 2
(a) CDB = IIDB + DIDB + IEDB
+ negative nonunit clauses
(b) RDB = DEDB
+ negative unit ground facts

REPRESENTATION 3
(a) CDB = IIDB + DIDB + IEDB
+ negative clauses
(b) RDB = DEDB

Example 3
Let DB = { -P(x) Q(x), -U(x) S(x) V(x), -T(x) P(x) R(x) S(x), -Q(a), -S(a), T(a), U(a)}.

In representation 2,
CDB = {-P(x) Q(x), -U(x) S(x) V(x), -T(x) P(x) R(x) S(x)} and
RDB = {-Q(a), -S(a), T(a), U(a)}.
Then NH[P] = { nh1: -T(x) P(x) R(x) S(x)} and PSUB[nh1] = {-U(x) S(x) V(x), -T(x) Q(x) R(x) S(x)}.
Then NH[P(c1)] = {nh1: -T(c1) P(c1) R(c1) S(c1)} and PSUB[nh1] ={-U(c1) S(c1) V(c1), -T(c1) Q(c1) R(c1) S(c1)},
where c1 denotes a generic constant. Hence, PIGC[P(a)] is empty.

In representation 3,
CDB = { -P(x) Q(x), -U(x) S(x) V(x), -T(x) P(x) R(x) S(x), -Q(a), -S(a)} and RDB = {T(a), U(a)}.
Then, performing resolution on the CDB yields the following resolvents :
-T(x) R(x) S(x) Q(x), -P(a), -U(a) V(a), and -T(a) R(a).
Hence, NH[P(c1)] = {nh1: -T(c1) P(c1) R(c1) S(c1)} and PSUB[nh1] = {-T(a) R(a)}.
Hence, PIGC[P(a)] is empty.

In example 3, notice that the clause -U(x) S(x) V(x) should be in PSUB[nh1] for representation 2, since the literal V may be resolved with the negative data in RDB if there is a negative table for V, and these resolutions are made at query time, not at compile time. In representation 3, any negative information about V would have to be in the CDB and would therefore be resolved at compile time.

Theorem 2 (Representation Theorem)
1. In representation-1, PIGC[q] of DB is not equivalent to PIGC[q] of RDB + NH[q] + PSUB[nhi].

2. In representation-2, PIGC[q] of DB is equivalent to PIGC[q] of RDB + NH[q] + PSUB[nhi].

3. In representation-3, PIGC[q] of DB is equivalent to PIGC[q] of RDB + NH[q] + PSUB[nhi].

The representation theorem indicates that representation schemes 2 and 3 enable us to compile the CDB with respect to NH[q] and PSUB[nhi] before query time. In order to avoid extra overhead in the deduction at compile time, we may prefer representation 1. However, we have some difficulties with the algebraic manipulation of the IEDB in representation 1: First, the ordinary relational table is not adequate for storing indefinite clauses due to the variable length of these clauses. Second, the representation theorem indicates that it is very difficult to develop the interface for generating PIGC[q] between CDB and RDB. In order to avoid some combinatorial explosion due to indefinite ground clauses at query time, we may prefer representation 2 and 3. Representation 2 needs an additional RDB operation for handling the negative unit ground clauses, while representation 3 needs additional resolutions on the CDB. When the negative ground facts are updated into the IDDB, some modifications to the compiled program are needed in representation 3, but not in representation 2. However, assuming that the volume of explicit negative ground facts is not very large and updates to them are not frequent, representation 3 may be preferred, since it reduces the size of NH[q] and PSUB[nhi] and it incorporates the traditional relational DB as the RDB.

V COMPILING INDEFINITE INFERENCE IN A NON-RECURSIVE IDDB

The indefiniteness theorem and its corollary tell us that for answering Indef[q] and GCWA[q], we must calculate True[q] and PIGC[q]. The representation theorem tells us that we may have the following scenario for developing more practical indefinite and GCWA inference engines which consists of three major procedures, namely Compile[q], Eval[q], and Modify[C], in the following manner: (1) At DB design time, the procedure Compile[q] compiles generic queries with respect to the CDB. (2) At query time, the procedure Eval[q] evaluates True[q], Indef[q], and GCWA[q] for a closed query, q, by evaluating the compiled program through the RDB. (3) At update time, the procedure Modify[C] modifies the compiled programs with respect to the update of a clause C into CDB. Updates in RDB require no program modification.

The compilation of the CDB may be performed by various techniques such as linear resolution[Chang and Lee, 1973], connection graph[Kowalski, 1975] [McKay and Shapiro, 1981] [Sickel, 1976], a generalized version of the system graph[Lozinskii, 1985], etc., of which the effectiveness will depend upon the structure of IDDB. In this paper, we show a simple saturated resolution technique for compiling a non-recursive IDDB with a small CDB. However, even though the CDB of an IDDB consists of only a few clauses, a large volume of resolvents may be generated by simple saturation. We present a more effective compiling technique in [Henschen and Park, 1985] [Park, 1985] by introducing NH-reduction theorems. Furthermore, we present a basic idea on compiling queries in a recursive IDDB in [Park, 1985].

Compilation Phase
For True[q], perform resolution on the CDB until saturation occurs, i.e. no more resolutions are possible. Construct a set of Horn clauses, called PTRUE[q], of which the positive literal unifies with q and the negative part consists of only base relations. For PIGC[q], perform resolution on the CDB until saturation occurs, and construct NH[q] and PSUB[q] defined in the previous section.

Evaluation Phase
For True[q], evaluate the negative part of each Horn clause in PTRUE[q] by performing join operations through the RDB, until either True[q] = t or PTRUE[q] has been exhausted. For PIGC[q], evaluate the negative part of each clause in NH[q] and PSUB[nhi] and compute PIGC[q] and its potential subsuming clauses. Perform subsumption tests on each clause in PIGC[q], say pigc, by its potential subsuming clauses and relations in RDB relevant to pigc.

Example 4 illustrates the compilation and evaluation of Indef[q] and GCWA[q] in a non-recursive IDDB by resolution, using representation scheme 3. The given IDDB partially describes the blood type relationship between parents and children.

Example 4
Base Relations:
P(person, father, mother)
B(person, blood_type)

Virtual Relations:
FB(person, father_blood_type)
MB(person, mother_blood_type)
BP(person, possible_blood_type)

```
CDB:
P(x1,x2,x3) & B(x2,x5) --> FB(x1,x5)
P(x1,x2,x3) & B(x3,x5) --> MB(x1,x5)
FB(x1,A) & MB(x1,O)
        --> BP(x1,A) V BP(x1,O)
B(x4,x5) --> BP(x4,x5)
-BP(g,O)

RDB:
P(a,i,j)    P(b,m,n)    P(e,a,b)    P(f,a,b)
P(g,a,b) B(a,A) B(b,O) B(e,A)
```

Compilation:
```
PTRUE[BP(c1,c2)] =
 {h1: B(x4,x5) --> BP(x4,x5)}
NH[BP(c1,c2)] =
 {nh1: P(x1,x2,x3) & B(x2,A) & B(x3,O)
                --> BP(x1,A) V BP(x1,O)}
PSUB[nh1] =
 {B(x4,x5) --> BP(x4,x5),
  P(g,x2,x3) & B(x2,A) & B(x3,O)
                      --> BP(g,A)}
```

Evaluation:
For the query BP(e,A),
PTRUE[BP(e,A)] = {h1:B(e,A) --> BP(e,A)}
True[BP(e,A)] = t by resolving h1 in PTRUE and B(e,A) in RDB.
Hence, Indef[BP(e,A)] = f and
GCWA[BP(e,A)] = f.

For the query BP(f,A),
NH[BP(f,A)] = {nh1: P(f,x2,x3) & B(x2,A)
      & B(x3,O) --> BP(f,A) V BP(f,O)}

PSUB[nh1] = {B(f,A) --> BP(f,A)}
PIGC[BP(f,A)] = {BP(f,A) V BP(f,O)}
Since True[BP(f,A)]=f and PIGC[BP(f,A)] is not empty, Indef[BP(f,A)] = t and GCWA[BP(f,A)] = f.

For the query BP(g,A),
PIGC[BP(g,A)] is empty, since BP(g,A) subsumes BP(g,A) V BP(g,O).
That is, True[BP(g,A)] = t.
Hence, Indef[BP(g,A)] = f and GCWA[BP(g,A)] = f.

This example is relatively simple because the indefinite predicate, BP, does not occur as an hypothesis of any rule. We point out that the complexity of the simple saturation method grows very fast as more indefinite predicates occur as hypotheses and as the length of resolution chains stemming from an indefinite hypothesis grows.

VI QUERY DECOMPOSITION

We introduce the following decomposition theorem to evaluate disjunctive and conjunctive queries from their unit subqueries.

Theorem 3 (Decomposition Theorem)
Let CL1 and CL2 be different clauses. "*" denotes "don't care" and "x" denotes "t, f, or i(indefinite)".

1. Disjunctive decomposition

CL1	CL2	CL1 V CL2
t	*	t
f	x	x
i	i	i or t

2. Conjunctive decomposition

CL1	CL2	CL1 V CL2
f	*	f
t	x	x
i	i	i or f

Notice that the decomposition theorem shows a duality between disjunctive and conjunctive decomposition. In disjunctive decomposition, if all ground literals appearing in CL1 and CL2 are indefinite, CL1 V CL2 may be either indefinite or true. Let DB = {p r, q s, -p -q}. Then, the minimal models are M1 = {r, s}, M2 = {p, s}, and M3 = {q, r}. Let DB' = {p r, q s}. Then, M1' = {p, q}, M2' = {p, s}, M3' = {r, q}, and M4' = {r, s} are the minimal models. Let CL1 = -p and CL2 = -q. Then, both CL1 and CL2 are indefinite with respect to DB and DB'. However, CL1 V CL2 = -p V -q is true with respect to DB, while it is indefinite with respect to DB'.

Disjunctive queries can be evaluated as follows. Let Q = L1 V L2 V .... V Ln. Then, determine the value of each literal Li by utilizing the compiled program for it, and evaluate Q by using the disjunctive decomposition theorem. In case all Li are indefinite, there are two ways to proceed. First, we may look for a straightforward refutation of DB & -Q to infer the value of Q. If nil is derived, Q is true. Otherwise, Q is indefinite. Second, Q may be evaluated by utilizing the compiled program of unit queries as follows. Generate PIGC[Li] for each Li. Let pigc be a clause in PIGC[Li]. If Q is a positive clause and there is a pigc consisting of only ground atoms of Q, Q is true with respect to DB. Otherwise, it is indefinite. For example, let DB = {p r, q s} and Q = p V q V r. All p, q, and r are indefinite. Since we can generate p V r consisting of only ground atoms p and r appearing in Q, Q is true. Notice that Q may be compiled. Evaluation theorems for more complex queries including conjunctive queries are presented in [Park, 1985].

VII CONCLUSION

Our goal is to develop effective inference engines for indefinite databases. We have shown that PIGC is the key to determining when a positive ground literal is indefinite or can be assumed false under GCWA. Further, we have shown which sets of resolvents must be generated in a compile phase in order to separate deduction from data retrieval. We have shown that two of the three obvious representation schemes allow such clause sets to be generated in a separate compile phase. We have shown how conjunction and disjunction can be handled. Work beyond that described in [Henschen and Park, 1985][Park, 1985] is needed to improve the actual compilation, in particular the generation of just the right resolvents in an effective way. This is especially true for recursive IDDBs.

REFERENCES

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

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

[3] Chang, C.L. and R.C.T. Lee. Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York, 1973.

[4] Clark, K.L. "Negation as Failure." In Logic and Databases. H. Gallaire and J. Minker, Eds. Plenum Press, New York. (1978), pp. 293 -324.

[5] Gallaire, H., J. Minker, and J. Nicolas. "Logic and Databases: a deductive approach." ACM Computing Surveys. 10:2 (1984) pp. 153-185.

[6] Henschen, L.J. and S. Naqvi. "On compiling queries in recursive first-order databases." J.ACM 31:1 (1984) pp. 47-85.

[7] Henschen, L.J. and H.S. Park. "Compiling the GCWA in indefinite deductive databases." in preparation for Maryland Workshop on Deductive Databases and Logic Programming. Maryland, August, 1986.

[8] Kowalski, R. "A proof procedure using connection graphs." J.ACM 22:4 (1975) pp. 572-595.

[9] Lozinskii, E.L. "Evaluating queries in deductive database by generating." In Proc. IJCAI-85. pp. 173-177.

[10] Maier, D. The Theory of Relational Database. Computer Science Press, Maryland, 1983.

[11] McKay, D. and S. Shapiro. "Using active connection graphs for reasoning with recursive rules." In Proc. IJCAI-81. pp. 24-28.

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

[13] Minker, J. and J. Grant, J. "Answering queries in indefinite databases and the null value problems." University of Maryland, College Park, Maryland, July, 1981.

[14] Park, H.S. "Compiling queries in indefinite deductive databases under the generalized closed-world assumption." in preparation for Ph.D. dissertation, Department of EECS, Northwestern University, August, 1986.

[15] Reiter, R. "Deductive question answering on relational databases." In Logic and Data Bases. H. Gallaire and J. Minker, Eds. Plenum Press, New York, 1978-a, pp. 149-177.

[16] Reiter, R. "On closed world databases." In Logic and Databases. H. Gallaire and J. Minker, Eds. Plenum Press, 1978-b, pp 55-76.

[17] Reiter, R. "Towards a logical reconstruction of relational database theory." In On Conceptual Modelling. M.L. Brodie, J. Mylopoulos, and J.W. Schmit, Eds. Springer-Verlag, New York, 1984, pp. 163-189.

[18] Sickel, S. "A search technique for clause interconnectivity graphs." IEEE Transactions on Computer. C-25:8 (1976) pp. 823-834.

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