A Nonmonotonic Typed Multilevel Logic for Multilevel Secure Data / Knowledge Base Management Systems - II

Bhavani Thuraisingham

The MITRE Corporation, Bedford, MA 01730

ABSTRACT

The study of databases through formal logic has not only enabled efficient proof procedures to be developed for query evaluation and integrity checking, but it has also resulted in the development of more intelligent and powerful database management systems. Despite all these advances, a logic for reasoning in a multilevel environment or a logic programming system for multilevel environments does not exist at present. As a result, multilevel secure database management systems lack several features that have been successfully incorporated into conventional database management systems. These include constraint processing, deductive reasoning, and handling efficient proof procedures. Classical first order logic, being monotonic, is an inappropriate tool for formalizing concepts in multilevel databases. This is because, in a multilevel environment, it is possible for users at different security levels to have different views of the same entity. In other words, statements that are assumed to be true at one security level can very well be false at a different security level. Therefore, a special logic is needed for formalizing concepts in multilevel databases as well as to develop multilevel database systems.

In our earlier paper we described a logic called Nonmonotonic Typed Multilevel Logic (NTML) for multilevel database applications. We also described various approaches to viewing multilevel databases through NTML. In this paper we continue with our discussion of the applications of NTML. In particular, the use of NTML as a programming language, issues on handling negative information in multilevel databases, and approaches for integrity checking in multilevel database systems are described. Our work on NTML will be of significance to multilevel data/knowledge base applications in the same way logic programming has been to the development of data/knowledge base applications.

1. INTRODUCTION

Ever since Colmerauer and Kowalski pioneered the use of predicate logic as a programming language (see for example [KOWA74]), Mathematical Logic has been applied to various areas of computer science such as database systems [GALL78, REIT78, CLAR78, KOWA78, NIC078a, NIC078b, LLOY87, MINK88]. It has not only been used as a framework to study their properties, it has also been used as a basis for developing powerful intelligent database systems [BIBE89]. The first workshop on Logic and Databases held in France in 1977 [GALL78] discussed the formalism of first order logic for database systems, which subsequently led to the formalization of relational database concepts using the proof and model theoretic results of first order logic. Further research activities contributed significantly to the development of advanced logic programming languages, inference engines for database systems, treatment of integrity constraints, and in handling negative, partial, and uncertain information. As a result, complex deduction and decision making processes have been incorporated into commercial intelligent data/knowledge base management systems available today [COHE89].

In the meantime, the recommendations of the Air Force Summer Study [AFSB83] led to the design and development of multilevel secure relational database management systems [HINK75, GRAU84, DENN87, THUR87, KEEF89, STAC90, THUR90a]. In such database systems, users cleared at different security levels can access and share a database with data at different sensitivity levels without violating security. Despite these advances, logic programming language research and research activities in multilevel secure database management systems remained largely separate. That is, a logic for reasoning in a multilevel environment or a logic programming system for multilevel environments is not currently available. Thus, multilevel secure database management systems lack several important features that have been successfully incorporated into conventional database management systems. They include constraint processing, deductive reasoning, and handling efficient proof procedures.

An early attempt was made to view multilevel databases through first-order logic [THUR88]. Although not entirely successful, this approach helped gain an insight into utilizing formal logic to develop multilevel systems. That is, classical first order logic, being monotonic, was found to be an inappropriate tool for formalizing concepts in multilevel databases. This is because it is possible for users at different security levels to have different views of the same entity. In other words, statements that are assumed to be true at one security level can very well be false at a different security level. Another contention is that first-order logic deals with only one universe (or world). In a multilevel database environment, there is a world corresponding to each security level. In other words, the universe in a multilevel environment is decomposed into multiple-worlds, one for each security level. Considerations such as these have led us to believe that a special logic is needed for reasoning in a multilevel environment. From an

---

1 We define an intelligent database system to be a database system with deduction capability. A knowledge base management system is a system which cannot only make deductions, but it can also reason in the midst of incomplete, uncertain, imprecise, and imperfect information among others. We do not differentiate between a knowledge base management system and a knowledge-based system. Some view a knowledge-based system to be an expert system. In any case, there are no standard definitions for these systems. For a discussion on integrating artificial intelligence and database concepts, we refer to [BROD86].
examination of the various nonstandard logics described in the
literature [TURN84, FROS86], none appeared capable of being
used for multilevel systems. Therefore, we have developed a
logic for not only formalizing multilevel database concepts, but
also for developing intelligent multilevel database systems.

In our earlier paper [THUR91a] we described a logic that
we developed for multilevel databases called a Nonmonotonic
Typed Multilevel Logic (NTML). It extends typed first-order
logic to support reasoning in a multilevel environment. We
have also formalized multilevel databases using NTML. In
particular, the proof theoretic and model theoretic approaches
for viewing multilevel databases were described. In this paper we
continue with our discussion of the applications of NTML. In
particular, we describe the use of NTML as a programming
language, handling negative information in multilevel databases,
and the treatment of integrity constraints in multilevel database
management systems. The work reported in [THUR91a]
together with the discussion in this paper describe the use of
logic for multilevel data/knowledge base management systems.

The organization of this paper is as follows. In section 2
we provide some background information on NTML taken from
[THUR91a]. In section 3 we discuss the essential points towards
the use of NTML as a programming language. Approaches to
handling negative information in multilevel databases are described in section 4. Handling integrity constraints is discussed in section 5. The paper is concluded in section 6.

We assume that the reader is familiar with both logic and
database concepts as well as multilevel database concepts. Logic
and database concepts are documented in [GALL78, GALL84].
For a discussion on database concepts we refer to [ULLM88]. A
useful starting point for multilevel database concepts is the Air
Force Summer Study Report [AFSB83]. A knowledge of
elementary mathematical logic and logic programming is useful
in order to understand the various concepts discussed in this
paper. For a discussion on these topics we refer to [MEND79,
CLOC84, LLOY87]. An excellent exposition on viewing
data/knowledge base management systems through logic is given in
[FROS86].

2. BACKGROUND

In this section we describe the essential points of the logic
NTML and discuss how multilevel database could be viewed
through such a logic. For a more detailed discussion of this
logic we refer to our earlier paper [THUR91a].

NTML is a nonmonotonic typed multilevel logic which
extends first order logic for reasoning in a multilevel
environment. It has a syntax and a semantics. The syntax of
NTML is typed first-order logic with extensions to support
multilevel security. It consists of the primitive symbols, terms,
and formulas of the language. Security properties are enforced
for each symbol, term, and formula. Any NTML theory based
on this syntax must satisfy these properties.\textsuperscript{3} NTML semantics
consists of interpretations of the primitive symbols, terms, and
formulas with respect to security levels. For example, a formula
which is assigned a security level L has a truth value in each
world which dominates the level L.

As in any logic theory, an NTML theory has a set of
logical axioms, a set of proper axioms, and a set of inference
rules. The logical axioms of an NTML theory are analogous to
those of first-order logic with equality. The rules of inference
of an NTML theory are Modus Ponens (MP), Generalization (GEN)
and Deductions Across Security Levels (DASL). For example,
if a formula F is true at the Unclassified level and is false at the
Secret level, then it is assumed to be false at all levels which
dominate the Secret level unless it is explicitly specified
otherwise.

We described three approaches to viewing multilevel
databases through NTML. In this first approach, called the proof
theoretic approach, the perceived multilevel universe is
represented as a NTML theory. The proper axioms of the theory
are the general laws (which are also called integrity constraints)
and the multilevel database. The implicit information is the
theorems of the NTML theory. The actual multilevel universe is
an interpretation of the NTML theory. Whether the actual
universe is a model of the theory depends on how accurately the
actual universe fits the perceived universe.

In the second approach, called the model theoretic
approach, the set of elementary information is considered to be
an interpretation of an NTML theory. This is the approach that
is implicitly followed in a traditional relational database system.
The proper axioms of the NTML theory are the general laws.
All general laws are used as integrity rules and not as derivation
rules. That is, since certain general laws are the proper axioms
of an NTML theory, these laws must be satisfied by the
multilevel database.

The third approach, called the integrated approach, is a
mixture of the two previous approaches for handling the general
laws. That is, some of the general laws are taken as integrity
rules and the others as derivation rules. The database has two
components. One is the explicit extension which is a model of
an NTML theory whose proper axioms are the general laws
which are regarded as integrity rules. The other is the implicit
extension which is the set of all tuples which are derived from
the explicit extension by virtue of the general laws which are
used as derivation rules.

The proof theoretic, model theoretic, and integrated
approaches have not only enabled efficient proof procedures to be
developed for query evaluation and integrity checking in
relational database systems, intelligent (or deductive) database
systems have also been developed. It is envisaged that
formalizing multilevel relational databases using logic will not
only enable efficient proof procedures to be developed for query
evaluation and security/integrity checking, but will enable
multilevel secure intelligent database systems to be developed.

\textsuperscript{2} NTML evolved from NPML (Nonmonotonic Propositional
Multilevel Logic). NPML is a propositional logic for a
multilevel environment. A discussion of NPML is given in
[THUR90b].

\textsuperscript{3} In [THUR90b] we have formally specified these security
properties.
3. NTML AS A PROGRAMMING LANGUAGE

In this section we describe our investigation toward the development of a programming language based on NTML. In section 3.1 we discuss the motivation for such a language and in section 3.2 we describe a resolution rule.

3.1 MOTIVATION

The advent of logic programming languages during the 1970s enabled computing systems to exhibit a certain degree of intelligence or deductive reasoning—an important feature which until then, remained relatively elusive. In other words, logic programmers need specify only the logic component of an algorithm without having to write any procedures or sets of instructions for the computer. The latter task is often tedious and time-consuming and can be effectively handled by the deductive reasoning component of the logic programming system.

Logic programming languages, such as Prolog, are gaining in popularity and are being heralded as languages of the future [ICOT87]. Therefore, it is important and useful that a logic programming language be developed that is appropriate for knowledge-based military applications. That is, one that incorporates constructs for multilevel security. However, current logic programming languages cannot be used for knowledge-based multilevel secure applications as they require the programmer to specify the procedures, thus causing it to lose the main advantages of logic programming. In contrast, we have developed the initial features of a logic programming language, called NTML-Prolog for multilevel applications. NTML-Prolog is based on a subset of NTML. Such a development will be of significance to multilevel knowledge-based applications in the same way Prolog has been to the development of knowledge-based applications.

In section 3.2, we describe the essential points toward the development of a logic programming language that we call NTML-Prolog. In particular, we describe an NTML-Prolog Program and a resolution rule for NTML-Prolog. Much research needs to be done before an interpreter for NTML-Prolog can be implemented. However, producing a workable NTML-Prolog system will be significant in the development of multilevel intelligent data/knowledge base management systems in the same way Prolog has marked a significant milestone in the development of intelligent data/knowledge base management systems.

3.2 NTML-Prolog

In this section we discuss the essential points of NTML-Prolog. We assume that the reader is familiar with the programming language Prolog [CLOC84].

An NTML program consists of a set of program clauses. A program clause is of the form:

\[
(((A.L) \leftarrow (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n), L^*) \quad \text{or} \\
((NOT (A.L)) \leftarrow (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n)), L^*)
\]

where \(n \geq 0\) (note that if \(n = 0\), there are no \(B_1\)'s); \(L, L_1, L_2, \ldots, L_n, L^*\) are security levels; \(L, L_1, L_2, \ldots, L_n\) are all \(\leq L^*\) (if the security levels are constants); \(A, B_1, B_2, \ldots, B_n\) are (positive) atomic NTML formulas.

The second form is the negation of the first form. The first form is read as follows:

"To show that \(A\) is true at level \(L\), we need to show that \(B_1, B_2, \ldots, B_n\) are all true at levels \(L_1, L_2, \ldots, L_n\), respectively. Further, this entire statement is true at the level \(L^*\)."

A goal clause is of the form:

\[
\langle- (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n)
\]

where \(n \geq 1\), \(L_1, L_2, \ldots, L_n\) are security levels and \(B_1, B_2, \ldots, B_n\) are (positive) atomic NTML formulas.

Note that an NTML-Prolog program is based only on a subset of NTML. That is, we involve only the clauses with at most one negative NTML atom.\(^4\)

Next, we describe the algorithm which implements the resolution rule for NTML-Prolog. A query posed by a user at security level \(L\) is expressed as a goal associated with level \(L\). Let the goal be the following:

\[
\langle- (A_1, L_1), (A_2, L_2), \ldots, (A_m, L_m)
\]

The resolution principle is implemented as follows.

For each \(i (1 \leq i \leq m)\) do the following:

Unify \((A_i, L)\) with some program clause

\[
(((A^*L^*) \langle- (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n), L^*)
\]

where the following conditions are satisfied:

(a) \(L^* \leq L\) if \(L^*\) is a constant. (Note that \(L^*_i, L_1, L_2, \ldots, L_i\) all \(\leq L^*_\).

(b) It is not the case that there is a security level \(L^*_i\) where \(L^*_i < L^*\), such that

\[
(((NOT (A^*L^*)) \langle- (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n), L^*).
\]

Note, by unification we mean find the most general unifier for \(A_i\) and \(A^*\) which satisfies the conditions stated above.

If a unifier cannot be found, then there is no solution for the query. Return "failure" as the response.

If there is such a unifier, then it is included in the response being assembled. Try the procedure (i.e., unification) for \((A_{i+1}, L)\).

End (for each i).

Return the response to the query. If another solution has to be found, then repeat the same procedure, but exclude the previous

\(^4\) If we do not explicitly specify the level of an atomic formula, then we assume that its level is system-low.
responses in the unification process. That is, find only the new solutions.

This ends the algorithm for implementing the resolution principle.

The resolution principle as described here will ensure that appropriate responses at or below the user's level will be retrieved.  

4. HANDLING NEGATIVE INFORMATION

4.1 OVERVIEW

As stated in [NICO78a], negative information corresponds to the fact that a given tuple does not satisfy a relation and may have to be represented by a negative ground literal. In a logical system, positive and negative information are treated in the same way. That is, both types of information have to be explicitly specified or derived from other information. In database applications, negative information is represented implicitly. That is, the negation of any statement is assumed to be true if the statement is not asserted. Such an implicit representation of negative information has been investigated extensively by researchers of logic programming and databases. The most notable efforts are those of Reiter [REIT78a], Clark [CLAR78], and Nicolas and Gallaire [NIC078a]. Many of the other proposals for handling negative information [MINK82, SHEP84] can be regarded as an evolution of these earlier proposals.

In [THUR88] an attempt was made to extend Reiter and Clark's proposals for multilevel databases. However, as stated earlier, that effort is solely based on first-order logic, and, therefore, does not capture many of the essential features of multilevel reasoning and data processing. In this section, we consider Reiter's and Clark's proposals to handle negative information within the framework of NTML. Reiter's proposal, known as the closed world assumption (CWA), states that negative information does not cause any complications if there are no gaps in the knowledge. Reiter also shows that, when only ground literals are handled, CWA does not produce any inconsistencies. But when general laws are taken into consideration, and when a deduction process is involved, this is not always the case. Clark's proposal is concerned with interpreting negation as failure. He calls such a rule "the negation as failure rule" and develops a proof procedure for a Horn clause logic program which handles negation. In section 4.2 we discuss CWA, and in section 4.3 we discuss negation as failure within the framework of NTML.

Our discussion on viewing multilevel databases through NTML, given in [THUR91a] showed how Nicolas and Gallaire's proposal could be adapted for a multilevel environment. As part of that discussion, issues on handling negative information were also addressed. In particular, the following conclusions were reached for the three approaches to handle negative information. They are the following:

1. In the proof-theoretic approach, negative information must be made explicit.
2. In the model theoretic approach, it is implicitly assumed that a tuple which does not appear in the extension of a relation does not satisfy it.
3. In the integrated approach it is implicitly assumed that a tuple which does not appear in the extension of a relation does not satisfy it. But there is an additional problem here, as some of the laws are used as derivation rules, and, therefore, one can envisage information to be deduced from negative information implicitly assumed. This problem is overcome by not permitting certain rules to be derivation rules.

4.2 CLOSED WORLD ASSUMPTION FOR MULTILEVEL DATABASES

4.2.1 Overview

Reiter [REIT78] has distinguished two conditions under which queries could be evaluated; the open-world assumption (OWA) and the closed-world assumption (CWA). Under OWA, queries are evaluated using the first-order approach. That is, queries are expressed as formulas of first-order logic, and, proofs of the formula are attempted. Each proof will produce an answer to the query. Under CWA, if no proof of a positive ground literal exists, then its negation is assumed to be true. That is, certain answers to a query are permitted as a result of the failure to find proofs. Under CWA, the database constitutes all positive ground literals explicitly specified, plus the negation of the ground literals implicitly assumed to be true. For applications which utilize logic for knowledge representation, it may be necessary to evaluate queries under OWA. But Reiter points out that, for many database applications, it is reasonable to evaluate queries under the CWA.

Our intent is to examine the various issues involved in extending CWA to a multilevel environment. A preliminary discussion on this topic will be given in section 4.2.2. First, we discuss some terminology. We also explain OWA for a multilevel environment as it can then be compared with CWA.

We assume that queries are expressed as NTML formulas. The level L of the user posting the query is the level at which the formula is specified. A multilevel database (MDB) is a NTML-Prolog program (which contains no function signs). An answer to a query: $Q = \forall x : T_1 \exists y : T_2 W(x,y), L$ is a set of elements $(c_1, c_2, \ldots, c_n)$ if and only if the following conditions are satisfied:

1. Each $c_i$ is of type $T_1$.
2. Level($c_i$) $\leq L$.

Note that the level at which the formula is asserted dominates the inherent level of the formula.

Note that $x$ and $y$ could be tuples instead of individual elements. We assume that they are individual elements for convenience. The query requests to find all $x$ such that there is a $y$ for which $W(x,y)$ holds.

7 Note that the level at which the formula is asserted dominates the inherent level of the formula.
8 Note that $x$ and $y$ could be tuples instead of individual elements. We assume that they are individual elements for convenience. The query requests to find all $x$ such that there is a $y$ for which $W(x,y)$ holds.
The set of all such answers will form the response to a query. The notation \( \{c_1+c_2+c_3+...+c_n\} \) is also used to denote the answer \( \{c_1,c_2,c_3,...,c_n\} \). It should be noted that if \( \{c_1, c_2, ... c_m\} \) is an answer, then so will \( \{c_1, c_2, c_3, ... , c_n, c_{n+1}\} \) be an answer provided \( n+1 \) is of type \( T_1 \) and has level \( \leq L \). This suggests the need for a minimal answer. A minimal answer to \( Q \) is defined to be an answer such that no proper subset of it will be an answer. If an answer consists of a single element \( \{c\} \), then it is a definite answer. Otherwise it is an indefinite answer.

Evaluating \( Q \) under open world assumption amounts to finding all minimal answers to \( Q \). This is denoted by \( \| Q \|_{OWA} \). We could also attach the security level \( L \) and denote it by \( \| Q \|_{OWA}(L) \).

Consider the following example. We assume that there are only two security levels: Unclassified and Secret. Let MDB consist of the following set \( E \) of employees and set \( D \) of departments. We use \( U \) to denote Unclassified and \( S \) to denote Secret.

\[
E = \{e_1(U), e_2(U), e_3(U), e_4(S)\}
\]
\[
D = \{d_1(U), d_2(U), d_3(S)\}
\]

Let the integrity constraints enforced at the Unclassified level be the following:

"Every employee must work in at least one department"

The constraint is expressed by the NTML formula:

\[
(\forall x / EMP \exists y / DEPT Works(x,y), U)
\]

Additional facts in the database are the following:

- \( (Works(e_1, d_1), U) \)
- \( (Works(e_2, d_3), S) \)

Suppose Unclassified and Secret users pose queries to find out where the employees are working. The queries will be expressed by the following NTML formulas:

- \( (\forall x / EMP \exists y / DEPT Works(x,y), U) \)
- \( (\forall x / EMP \forall y / DEPT Works(x,y), S) \)

The response (under OWA) to the query issued by the Unclassified user will be:

\[
\{(e_1, d_1), (e_2, d_1) + (e_2, d_2), (e_3, d_1) + (e_3, d_2)\}.
\]

The response (under OWA) to the query issued by the Secret user will be:

\[
\{(e_1, d_1), (e_2, d_3), (e_3, d_1) + (e_3, d_2) + (e_3, d_3),
(e_4, d_1) + (e_4, d_2) + (e_4, d_3)\}.
\]

The specific techniques employed to obtain these answers have been adapted from Reiter's work on query evaluation under OWA described in [REIT78b].

4.2.2 Closed World Assumption

We first illustrate query evaluation under CWA using a purely extensional multilevel database MDB. Such a database consists of ground literals of different security levels. We assume that there are only two security levels; Unclassified and Secret.

Consider the multilevel database shown in table 1. There are two domains EMP and DEPT. The MDB consists of one relation 'Works.' Note that, at the Unclassified level, \( e_1 \) works in \( d_1 \). At the Secret level, \( e_1 \) works in both \( d_1 \) and \( d_2 \). This could mean that, at the Secret level, \( e_1 \) works in both \( d_1 \) and \( d_2 \), or the tuple is polyinstantiated. For this discussion we assume that it is the former. That is, at the Secret level \( e_1 \) works in both departments \( d_1 \) and \( d_3 \) at the Secret level.

Table 1. A Multilevel Purely Extensional Database

<table>
<thead>
<tr>
<th>EMP</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_1</td>
<td>d_1</td>
</tr>
<tr>
<td>e_2</td>
<td>d_2</td>
</tr>
<tr>
<td>e_3</td>
<td>d_3</td>
</tr>
</tbody>
</table>

Table 2. Representing Negative Information

<table>
<thead>
<tr>
<th>EMP</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_1</td>
<td>d_1</td>
</tr>
<tr>
<td>e_2</td>
<td>d_2</td>
</tr>
<tr>
<td>e_3</td>
<td>d_3</td>
</tr>
</tbody>
</table>

9 In general the extensional portion of a multilevel database MDB is denoted by MEDB. In this example, we assume that MDB and MEDB are the same.

10 If we want to assume at the Secret level that \( e_1 \) does not work in department \( d_1 \), then as discussed in the model theoretic approach to multilevel databases, we need to duplicate every tuple that holds in the Unclassified level at the Secret level also. We will see that if negative information is explicitly specified, then negations at the Secret level can also be explicitly specified. That is, the tuple \( (e_1, d_1) \) can be included in the relation \( \neg Works. \)
Consider the following query posed at both the Unclassified and Secret levels. 
(∀x / EMP 1 Works(x, D2), U)
(∀x / EMP 1 Works(x, D2), S)

The query requests all those employees who do not work in department D2. Under OWA, the response to this query is NULL at both security levels. That is, if Q(OWA(U) = ∅ and Q(OWA(S) = Empty set.

What we actually want is the response (e1, e2) at the Unclassified level and the response (e2, e4) at the Secret level. Under OWA, this response cannot be obtained as first-order logic and does not implicitly assume that is not present in the database. Therefore, in order to obtain the response that we want, we need to explicitly specify the relation ¬ Works (usually denoted by Works) in the database. Table 2 shows the relation ¬ Works. The relation ¬ Works (U) has all the negative information at the Unclassified level and the relation ¬ Works (S) has the negative information at the Secret level. The database consists of the negative information of the complement of MEDB and is denoted by Comp-MEDB. The database. Therefore, in order to evaluate a query, the database MDB and Comp-MEDB (the complement of its extensional component) have to be examined.

Table 3. A Multilevel Deductive Database

| EMP | (c1, c2), (c3, c4), (c5, c6) |
| CSCI | (c1, c2), (c3, c4), (c5, c6) |
| DEPT | (c1, c2), (c3, c4), (c5, c6) |

Law 1: ∀x EMP V y/DEPT Works(x, y) A Subdept(x, y) 1 Works(x, y)
Law 2: ∀x EMP V y/DEPT Works(x, y) A Subdept(x, y) 1 Works(x, y)
Law 3: ∀x EMP V y/DEPT Works(x, y) A Subdept(x, y) 1 Works(x, y)

We now formally define query evaluation under CWA for an arbitrary multilevel database MDB under consideration. Consider the MDB shown in table 3. There are three domains: EMP (for employees), CSCI (for computer science departments), and DEPT (for departments). The elements of the various domains are also shown in the table. Note that a computer science department is also a department. We assume that there are only two security levels, Secret and Unclassified.

We illustrate query evaluation under CWA for an arbitrary multilevel database with an example. Consider the MDB shown in table 3. There are three domains: EMP (for employees), CSCI (for computer science departments), and DEPT (for departments). The elements of the various domains are also shown in the table. We assume that there are only two security levels, Secret and Unclassified.

The extensional MDB consists of two relations, Works and Subdept (the subdepartment relation). The intensional MDB consists of three general laws, which are represented as NTML formulas at the Unclassified level. These laws state the following:

Law 1: Any employer who works in a department works in all its sub-departments.
Law 2: Employee e2 works in all computer science departments.
Law 3: The 'Subdept' relation is transitive.

Part of Comp-MEDB for the MDB under consideration is shown in table 4. We assume that negative information at the Unclassified level is also valid at the Secret level provided the positive counterpart cannot be derived either implicitly or explicitly at the Secret level.

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on April 16,2010 at 18:06:32 UTC from IEEE Xplore. Restrictions apply.
Table 4. Representing Negative Information for a Multilevel Deductive Database

<table>
<thead>
<tr>
<th>Sect.</th>
<th>Dept.</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Works</td>
<td></td>
</tr>
<tr>
<td>e1</td>
<td>e3</td>
<td>U</td>
</tr>
<tr>
<td>e4</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td>e1</td>
<td>U</td>
</tr>
<tr>
<td>e2</td>
<td>e2</td>
<td>S</td>
</tr>
<tr>
<td>e1</td>
<td>e2</td>
<td>S</td>
</tr>
<tr>
<td>e0</td>
<td>d1</td>
<td>U</td>
</tr>
<tr>
<td>d2</td>
<td>d2</td>
<td>U</td>
</tr>
<tr>
<td>e2</td>
<td>d3</td>
<td>S</td>
</tr>
<tr>
<td>e3</td>
<td>d3</td>
<td>S</td>
</tr>
<tr>
<td>e4</td>
<td>e3</td>
<td>U</td>
</tr>
<tr>
<td>e1</td>
<td>e1</td>
<td>U</td>
</tr>
<tr>
<td>e2</td>
<td>e2</td>
<td>S</td>
</tr>
<tr>
<td>e3</td>
<td>e3</td>
<td>S</td>
</tr>
<tr>
<td>e4</td>
<td>e4</td>
<td>S</td>
</tr>
<tr>
<td>e0</td>
<td>d1</td>
<td>U</td>
</tr>
<tr>
<td>d3</td>
<td>d3</td>
<td>U</td>
</tr>
<tr>
<td>e2</td>
<td>d3</td>
<td>S</td>
</tr>
<tr>
<td>e3</td>
<td>d2</td>
<td>S</td>
</tr>
<tr>
<td>e4</td>
<td>e2</td>
<td>S</td>
</tr>
<tr>
<td>e1</td>
<td>e1</td>
<td>U</td>
</tr>
</tbody>
</table>

The following comments are in order:

1. Under CWA there are no gaps in our knowledge. That is, nothing is uncertain. This is because, for each predicate P, constant c, and level L, either Pc can be deduced from MDB or ¬Pc can be deduced from Comp-MEDB with respect to L. Further, since the multilevel database is taken to be MDB U Comp-MEDB, it will definitely be the case that either Pc or ¬Pc can be deduced from MDB U Comp-MEDB. Therefore, intuitively, it seems that all answers under CWA with respect to any security level are definite. Reiter has formally proved this for a nonmultilevel database. It appears that this proof can be extended to include a multilevel database also.

2. It can be seen that the negative information that has to be represented is very large. For any practical application it may not be possible to efficiently represent all the negative facts. Reiter has proved that, for many cases, the negative information need not be taken into consideration for query evaluation under CWA. That is, he shows that, for many cases, query evaluation coincides both under OWA and CWA. However, for some cases this does not work, and, therefore, negative information has to be explicitly specified. Reiter also shows that when the database is a non-Horn database, inconsistencies can arise with respect to CWA. However, if the database is a Horn database, then it is consistent under CWA. Future work should include investigating the applicability of Reiter's results for query evaluation under CWA in a multilevel database.

4.3 NEGATION AS FAILURE RULE FOR MULTILEVEL DATABASES

4.3.1 Overview

Clark [CLAR78] has defined a query evaluation process for a logic database (i.e., a deductive database) consisting of a set of clauses to be essentially Horn Clause theorem proving with a special inference rule which he calls the negation as failure rule in order to deal with negation. The negation as failure rule (denoted NF) simply states that ¬P can be inferred if every proof of P fails. That is:

From ⊢ ¬I: P infer ⊢ ¬P.

The proof that P is not provable is an exhaustive but unsuccessful search for at least one proof of P.

It is useful to compare the proposals of Reiter (CWA) and Clark (NF) to handle negation. The information ¬A can be inferred under CWA if A cannot be proven to be a consequence of the logic database. As stated in [LLOY87], in logic programming terminology, this amounts to the following: If A is not in the success set of the database DB, then ¬A is inferred. For A not to be in the success set of DB, either A is in the SLD finite failure set or the SLD-tree has at least one infinite branch. If A is in the SLD-finite failure set, then under CWA, we can assume ¬A. If there is an infinite branch in the SLD-tree, then unless there is a mechanism for detecting infinite branches, it will not be possible to show that A is not a logical consequence of the database DB.

In logic programming terminology,13 if the logic database is a set of Horn clauses, then the NF states that if A cannot be proven to be a consequence of the logic database DB, then ¬A is inferred. We can see that NF is weaker than CWA. However, as stated in [LLOY87], in practice, implementing anything beyond NF is difficult.

In section 4.3.2 we discuss how the NF rule could be applied for multilevel databases. We regard a multilevel database as an NTML-Prolog program. Query evaluation in such a database is resolution theorem proving. The resolution rule is the one that we have described in section 3 with negated literals evaluated by a failure proof. That is, we augment the resolution rule that we have designed with the rule NF for multilevel databases.

4.3.2 NF Rule for Multilevel Databases

The multilevel database consists of a set of NTML-Prolog clauses. Note that an NTML-Prolog clause could be a positive clause or it could be the negation of a positive clause at a different security level. We assume that each clause has a distinguished positive literal. The positive literal is what the clause defines and is placed at the head. For example, a clause is either of the form:

((R(t1,t2,...,tn), L*) < (K1, L1) Λ (K2, L2) Λ (K3, L3) .... (Kn, Ln)), L)

or

(¬((R(t1,t2,...,tn), L*) < (K1, L1) Λ (K2, L2) Λ (K3, L3) .... (Kn, Ln)), L)

13 For the logic programming terminology that we have used we refer to [LLOY87].
We first describe the query evaluation algorithm, and then illustrate it with an example.\(^\text{14}\)

**RESOLUTION RULE AUGMENTED WITH NF FOR MULTILEVEL DATABASES**

**Begin**

Let the query to be evaluated be \(<: (K_1, L) \land (K_2, L) \land (K_3, L) \land \ldots \ldots (K_n, L))\). Call it the current query \(Q\).

Until an empty query is derived and the evaluation succeeds, do the following:

Select a literal from the current query so that a negative literal is selected only if it does not have any variables.

**Case 1:**

If \(k_i\) is a positive literal \(R(t_1, t_2, \ldots, t_m)\).

Nondeterministically, choose a database clause

\(((R(t_1', t_2', \ldots, t_m), L^+) <: (K'_1, L_1) \land (K'_2, L_2) \land \ldots \ldots (K'_n, L_n), L^+)\) such that the level \(L^+\) is dominated by \(L\), and the clause is not negated at any level between \(L^+\) and \(L\).

Attempt to unify \(K'_i\) with \(R(t_1', t_2', \ldots, t_m)\). If \(K_i\) does not unify with this clause, then try another clause. If there is no other clause, the query fails. If \(K'_i\) does unify with \(R(t_1', t_2', \ldots, t_m)\), find the most general unifier \(\phi\) and replace the current query by

\(<: (K_1, L) \land (K_2, L) \land \ldots \ldots (K_i-1, L) \land (K'_i, L_1) \land (K'_2, L_2) \land \ldots \ldots (K'_n, L_n), L) \land (K_i+1, L) \land (K_i+2, L) \land \ldots \ldots (K_n, L)) \land \phi\).

**Case 2:**

If \(k_i\) is a negative ground literal \(-P\). Negative literals are proved by showing that all proofs of its positive counterpart fail. Therefore, nondeterministically enter the query evaluation process with \(<: P, L)\) as a query. If the evaluation of this query succeeds, then \(P\) has been shown to be a logical consequence of the database with respect to level \(L\). Therefore \(-P\) cannot be assumed to be true. This means that the evaluation of the original query \(Q\) fails.

If the evaluation of \(<: P, L)\) fails for every path of its nondeterministic evaluation, this means that all proofs of \(P\) fail, and therefore \(-P\) can be assumed to be true. Hence, the current query can be replaced by the query:

\(<: (K_1, L) \land (K_2, L) \land \ldots \ldots (K_i-1, L) \land (K_i+1, L) \land (K_i+2, L) \land \ldots \ldots (K_n, L))\). \(\Box\)

**End.**

We now illustrate the query evaluation process with an example. Consider the following NTML-Prolog program which consists of the clauses C1 - C25.

\begin{align*}
\text{C1}: & \quad ((\text{Works}(x, d1), U) \leftarrow \ldots, U) \\
\text{C2}: & \quad ((\text{Works}(x, d3), S) \leftarrow \ldots, S) \\
\text{C3}: & \quad ((\text{Works}(x, c4), S) \leftarrow \ldots, S) \\
\text{C4}: & \quad ((\text{Works}(x, z), L) \leftarrow (\text{Works}(x, y), L) \land (\text{SubDept}(x, y), L)) \\
\text{C5}: & \quad ((\text{Works}(x, e), L) \leftarrow (\text{Works}(x, y), L)) \\
\text{C6}: & \quad ((\text{EMP}(x), U) \leftarrow \ldots, U) \\
\text{C7}: & \quad ((\text{EMP}(x), U) \leftarrow \ldots, U) \\
\text{C8}: & \quad ((\text{EMP}(x), S) \leftarrow \ldots, S) \\
\text{C9}: & \quad ((\text{CSIC}(x), U) \leftarrow \ldots, U) \\
\text{C10}: & \quad ((\text{CSIC}(x), U) \leftarrow \ldots, U) \\
\text{C11}: & \quad ((\text{CSIC}(x), U) \leftarrow \ldots, U) \\
\text{C12}: & \quad ((\text{CSIC}(x), S) \leftarrow \ldots, S) \\
\text{C13}: & \quad ((\text{DEPT}(x), U) \leftarrow \ldots, U) \\
\text{C14}: & \quad ((\text{DEPT}(x), U) \leftarrow \ldots, U) \\
\text{C15}: & \quad ((\text{DEPT}(x), S) \leftarrow \ldots, S) \\
\text{C16}: & \quad ((\text{DEPT}(x), U) \leftarrow (\text{Works}(x, y), L)) \\
\text{C17}: & \quad ((\text{DEPT}(x), U) \leftarrow (\text{Works}(x, y), L)) \\
\text{C18}: & \quad ((\text{DEPT}(x), L) \leftarrow (\text{CSIC}(x), L)) \\
\text{C19}: & \quad ((\text{DEPT}(x), L) \leftarrow (\text{CSIC}(x), L)) \\
\text{C20}: & \quad ((\text{DEPT}(x), L) \leftarrow (\text{SubDept}(x, y), L)) \\
\text{C21}: & \quad ((\text{DEPT}(x), L) \leftarrow (\text{SubDept}(x, y), L)) \\
\text{C22}: & \quad ((\text{SubDept}(x, y), L) \leftarrow (\text{SubDept}(x, y), L)) \\
\text{C23}: & \quad ((\text{SubDept}(x, y), L) \leftarrow (\text{SubDept}(x, y), L)) \\
\text{C24}: & \quad ((\text{SubDept}(x, y), L) \leftarrow (\text{SubDept}(x, y), L)) \\
\text{C25}: & \quad ((\text{SubDept}(x, y), L) \leftarrow (\text{SubDept}(x, y), L)) \\
\end{align*}

Let the query posed by an Unclassified user be the following:

\(((\text{Works}(x, d1), U) \leftarrow (\text{SubDept}(d1, d2), U)).\)

This query requests to find all those who work in department \(d1\) provided \(d1\) is not a subdepartment of \(d2\). The query evaluation algorithm will nondeterministically select a literal to be evaluated. Suppose the negative literal \(-\text{SubDept}(d1, d2)\) is selected. Since this literal is ground, there is no problem. In order to evaluate a negative literal, it has to be shown that all proofs of its positive counterpart fail. Therefore, the query to be evaluated is:

\(<:: (\text{SubDept}(d1, d2), U), U)\).

Examining clauses C5, C22, C23, C24 and C25, it can be shown that the goal \(<:: (\text{SubDept}(d1, d2), U), U)\) is not satisfied. Therefore \(-\text{SubDept}(d1, d2)\) can be asserted as a lemma at the Unclassified level.

Next, the algorithm will evaluate \text{Works}(x, d1).\) If all solutions are to be found, then the algorithm is executed repeatedly assuming that \(-\text{SubDept}(d1, d2)\) is a lemma until no more answers are obtained. It can be shown that the answer to the query is \(\{e1\}\).

We have stated the essential points in applying NF rule for a multilevel database. Future research should include the investigation of the soundness and completeness of the proof procedure that we have developed. This would also involve investigating issues on the completion of multilevel databases.\(^\text{15}\). It should be noted that we need to first investigate

---

\(^{14}\) Compare this algorithm with Clark's SLDNF proof procedure.

\(^{15}\) Completion of databases are discussed in [CLAR78]
the soundness and completeness of the resolution rule that we
developed in section 2 for multilevel databases before
investigating these issues for the augmented resolution rule
described in this section.

5. INTEGRITY CHECKING IN MULTILEVEL
DATABASES

5.1 OVERVIEW

In this section we describe how integrity constraints may
be handled in multilevel databases. Integrity constraints are also
known as general laws. Two types of general laws have been
studied in the past for database systems. They are state laws and
transition laws. State laws are laws which refer only to one state
of the world. Transition laws are relative to world evolution.

In [THUR91a] we describe how general laws may be
handled by the three approaches to formalizing multilevel
database concepts. In the proof theoretic approach, all general
laws are regarded as derivation rules. That is, these laws are used
to deduce new information. In the model theoretic approach, all
general laws are treated as integrity rules. That is, these laws
must be satisfied by the multilevel database. In the integrated
approach, some laws are treated as derivation rules and others as
integrity rules.

For multilevel databases, the general laws also include the
security constraints which are used to assign security levels to
the data. As described in section 4, some of the security
constraints may be used as integrity rules and others as derivation
rules. For example, consider the following security constraint
enforced at the Unclassified level:

"A name of an employee is Secret if his salary is greater than
60K."

If this constraint is used as an integrity rule, then the following
formula must be satisfied by the multilevel database:

\( \text{Level}(\text{Name}) = \text{Secret V Salary-value} \leq 60K, \text{Unclassified}. \)

That is, if tuple level classification is provided, then the tuple
(John, 70K) must be classified at the Secret level, whereas the
tuple (James, 50K) could be classified at the Unclassified level.

If this security constraint is used as a derivation rule, then it is
used to deduce that any name whose salary is more than 60K is
Secret. For example, if (John, 70K) is stored at the Unclassified
level, then the derivation rule can be used during query time to
deduce that John's level is actually Secret.

Some laws are better treated as integrity rules, while
others are better treated as derivation rules. More research needs
to be done before a characterization of security constraints can be
obtained which will determine whether a constraint should be
treated as a derivation rule or as an integrity rule.\(^{16}\)

In section 5.2 we describe the issues involved in treating
general laws either as integrity rules or as derivation rules. We

only address state laws. Handling transition laws is a subject of
future research. Our treatment of general laws examines the
techniques proposed in [NICO78b] and adapts them for
multilevel databases. Since Nicolas’ work on integrity checking
in databases [NICO78b], several enhancements to various

\(^{16}\) Lock Data Views [STAC90] is designed in such a way that
some security constraints are treated as derivation rules and
others as integrity rules. However, that effort did not attempt
to characterize the security constraints. A more recent
approach to treating integrity constraints in multilevel
databases is given in [THUR91b].

5.2 HANDLING INTEGRITY CONSTRAINTS OR
GENERAL LAWS

In this section, we describe techniques for handling general
laws. We focus only on state laws.

5.2.1 Integrity Rules for Multilevel Databases

Integrity rules are expressed as NTML formulas. They
could be activated when data is updated.\(^{17}\) Efficient techniques
need to be developed in order to determine which of the integrity
rules need to be activated during a database update operation.

The first step is to rewrite an integrity rule in Skolem
form. This can be done as in the case of a first-order formula,
except that the security level is also attached to the formula in
Skolem form. Any NTML Skolem formula is a conjunct of
clauses. Therefore, for the entire formula to be validated under an
interpretation, each clause must be validated.\(^{18}\) Note that a
clause

\( (P_1 \lor \cdots \lor P_n \lor Q, L) \)

where \( m \) and \( n \) are not null at the same time. Any update to the
database at level \( L \) or higher could activate such an implication
clause if certain conditions are satisfied.

The following rules may be used to determine the
implication clauses that need to be activated during database
updates. Let the clause be of the form \( (P \implies Q, L) \) where
\( P \) is

\( P_1 \lor \cdots \lor P_n \lor Q, L \).

that activates the following rules:

\( (P_1 \implies Q, L) \)

\( (P_2 \implies Q, L) \)

\( (Q_1, L) \)

\( (Q_2, L) \)

\( \cdots \)

\( (Q_m, L) \).

(a) If a tuple is inserted at level \( L^* \geq L \) into a multilevel
relation \( R \), then the integrity rule

\( (P \implies Q, L) \)

is activated if:

\( (P_1 \implies Q, L) \) is not negated at \( L^* \).

\( R \) is one of the \( P_i \)'s.

(b) If a tuple is deleted at level \( L^* \geq L \) from a multilevel
relation \( R \), then the integrity rule

\( (P \implies Q, L) \)

is activated if:

\( (P_1 \implies Q, L) \) is not negated at \( L^* \).

\( R \) is one of the \( Q_i \)'s.

(c) If a tuple is modified at level \( L^* \geq L \) from a multilevel
relation \( R \), then the integrity rule

\( (P \implies Q, L) \)

is activated if:

\( (P_1 \implies Q, L) \) is not negated at \( L^* \).

\( R \) is either one of the \( P_i \)'s or one of the \( Q_i \)'s.

\(^{17}\) An update operation is either an insert, delete, or modify. A
modify operation can be regarded as a delete followed by an
insert.

\(^{18}\) An interpretation in this case is the multilevel database.
If an integrity rule is falsified, possible actions to be taken include the following:

1. Deny the operation. The user can also be told of the integrity rule that was falsified. He could then decide whether to accept the denial or whether to take some action. For example, he could try other updates first so that the denied update could be accepted later, or he could negate the integrity rule at his security level so that the update is accepted.

2. The second option is to automate some of the actions of the user as described in the first option. That is, the system could negate the integrity rule that was just violated at a higher level, or it could perform some other operations so that the integrity rule will not be violated later.

A problem is to determine which operations to perform in order for the rule not to be violated later. For a given rule ((P1 A P2 A ... A Pn) → Q1 A Q2 A ... A Qm), L, if an update to a P1 falsifies this rule, then the reason for this is that P1 A P2 A ... A Pn evaluates to True and Q1 A Q2 A ... A Qm evaluates to False. Any operation that would either make P1 A P2 A ... A Pn false or Q1 A Q2 A ... A Qm true would ensure that the integrity rule is satisfied. Note that if the request that was denied was attempted at level L* > L, and if any of the operations that have to be performed are at level L+ where L ≤ L* ≤ L*, then the user at level L* cannot perform these operations.19 In this case, the user could probably log in at the lower level L+ in order to perform the operation.20

5.2.2 Derivation Rules for Multilevel Databases

5.2.2.1 Overview

In this section, we discuss derivation rules for multilevel databases. We assume that they coexist with the integrity rules. That is, some of the laws are treated as derivation rules and some others as derivation rules.

Nicolas and Gallaire [NIC078a] showed that if the derivation rules are Horn clauses with one positive literal, then they can be safely used for deduction. In [THUR91a] we showed that this was also the case for multilevel databases viewed through NTML. We could exploit the derivation rules either during the query operation or during the update operation. That is, the derivation rules are divided into two groups. Those derivation rules that are handled during the query operation are called the Query-Derivation rules, and those that are handled during database updates are called the Generation-Derivation rules.

Query-Derivation rules are used to deduce implicit information which is then included in the response to a query. Generation-Derivation rules are used to deduce information and make it explicit during database updates. For example, if the rule (P(x) → Q(x), S) is a query derivation rule, and the database is ((P(b), S), (Q(a), U)), then if a Secret user requests to find all values for Q, the rule is used to deduce Q(b). Therefore, the response is (Q(a), Q(b)). If this rule is used as a generation-derivation rule and the database is (Q(a), U), when (P(b), S) is inserted, the rule is used to make (Q(b), S) explicit. We discuss both types of derivation rules in the next two subsections.

5.2.2.2 Query-Derivation Rules

We discuss how the query-derivation rules are handled during query, insert and delete operations.

1. Query operation

Let a query be posed at level L. A deduction process is required to run at level L during the query operation in order to deduce the implicit information that can be derived using the query-derivation rules. If the derivation rules are handled in a manner to be stated below during the insert and delete operations, then the level of all the information derived should in general, be dominated by L. The derived information will be part of the response.

If, however, the query-derivation rules are not handled appropriately during the insert and delete operations, then it could be possible for users to infer information from the legitimate responses that they receive. In this case, a deduction process should run at each security level L* ≥ L. The deduction process running at level L* should determine whether a user at level L* can infer information to which he is not authorized by seeing the response released at level L. If so, the response is not released at level L. Note that users could use several other inference strategies such as inductive reasoning, analogical reasoning and analogical reasoning in order to deduce information to which they are not authorized.21 Such inferences cannot be handled by an NTML-based deduction process. Furthermore, such inferences are almost always made as a result of the query operation. Therefore, it is necessary to have an inference controller which handles such inferences as part of the query processor.

2. Insert Operation

When inserting a tuple at level L, the explicit extensions of a relation will be modified. If any of the integrity rules are violated, then appropriate actions have to be taken as described earlier. Now, inserting a tuple could also modify the implicit extensions of other relations. For example, if (P(a), U) is inserted and (P(x) → Q(x), S) is a derivation rule, then the implicit extension of Q is affected at level S because (Q(a), S) can be deduced. If there is an integrity rule (~Q(a), U), then this rule is violated. Therefore, when a tuple is inserted at level L, deduction processes have to run at each level L* ≥ L. Each process should deduce the implicit information using the query-derivation rules. If any of the integrity rules are violated, then appropriate actions have to be taken.

Insertions can also cause redundancies. For example, if (Q(a), U) is asserted and (P(x) → Q(x), U) is a query-derivation rule, then inserting (P(a), U) will make (Q(a), U) redundant. Therefore (Q(a), U) could be deleted in order to avoid this redundancy. However, later if (P(a), U) is deleted and if (Q(a), U)

---

19 This will be ensured by the security policy enforced by the system.

20 In general, this is not a desirable solution.

21 The various inference strategies are identified in [THUR90c]
is still valid, then this information is lost. Therefore, unless some techniques can be devised so that (Q(a), U) is inserted when (P(a), U) is deleted, redundancies cannot be avoided.

(3) Delete Operation

When tuples are deleted, explicit or implicit extensions of relations may be affected. The relations that are affected need to be determined and appropriate integrity rules need to be activated.

Also, when a tuple is deleted, it could still be possible to derive this tuple using a query-derivation rule. For example, consider the database [(Q(a), U), (P(a), U)] and the derivation rule (P(x) -> Q(x), U). If (Q(a), U) is deleted, then it can still be derived. If this is the case, appropriate actions need to be taken. One possible action would be to determine all ways to deduce the deleted tuple and to request operations that will ensure that the deleted tuple cannot be derived.

5.2.2.3 Generation-Derivation Rules

We discuss how the generation-derivation rules are handled during insert, delete, and query operations.

(1) Insert Operation

When a tuple is inserted at level L, the generation-derivation rules are used to deduce implicit information. The deduced information is then made explicit. Note that information can be deduced at different security levels. Therefore, there has to be a deduction process running at each security level.

(2) Delete Operation

Since deduced information is made explicit during the insert operation, the tuple to be deleted could be a derived tuple, or it could be an original tuple.22 If original information is deleted, then information that is derived from it may not be valid anymore. For example, if (P(x) -> Q(x), U) is a generation-derivation rule and (P(a), U) is some original information, then (Q(a), U) will be derived. Deleting (P(a), U) will cause (Q(a), U) not to be valid unless (Q(a), U) is also explicitly asserted, or it can be derived some other way. Therefore, a deduction process running at level L* >= L (where L is the level of the delete operation) should identify which of the derived information cannot be deduced by other means. Such derived information should also be deleted.

If derived information is deleted, then the generation-derivation rule may not be valid anymore. For example, if the database is [(P(a), U)], (P(x) -> Q(x), U) is a generation-derivation rule, and (Q(a), U) is derived information. Then, if (Q(a), U) is deleted, the generation-rule is falsified. Therefore, (P(a), U) should also be deleted in order for the generation-rule not to be invalidated. A deduction process, which will determine the sequence of operations that have to be performed, is necessary in order for the generation-derivation rules not to be invalidated.

As stated earlier, we run into problems if the derivation rules and the database are at different security levels. For example, let ((P(a), U)) be the database, (P(x) -> Q(x), S) be a generation-derivation rule, and (Q(a), S) be derived information. Then, if (Q(a), S) is deleted, the generation-rule is falsified. Therefore, (P(a), U) should also be deleted in order for the generation-rule not to be invalidated. However, a Secret user cannot delete this information as it is classified at the Unclassified level. A solution to this problem would be for the Secret user to insert the assertion (~P(a), S) at the Secret level.

(3) Query Operation

The generation-derivation rules will make all the deduced information explicit during the insert operation. When the database is queried, all of the information will be present in the extensional database. Therefore, there is no need for a deduction process to operate during query processing.

6. CONCLUSION

In this paper we provided a brief overview of NLMT, the logic that we developed for multilevel databases as given in [THUR91a] and discussed three approaches to formalizing multilevel databases concepts using NLMT. Then we described applications of NTML. In the first application, we showed how a subset of NLMT could be used as a basis for a programming language which we called NTML-Prolog. A resolution rule for NTML-Prolog was also given. In the second application, we described how negative information could be handled in multilevel databases. Here, we examined the closed world assumption of Reiter [REIT79] and the negation as a failure inference rule of Clark [CLAR78], and we proposed analogous rules for multilevel databases. The third application was involved with approaches for handling integrity constraints in multilevel databases. By integrity constraints we mean the constraints enforced in traditional database systems and the security constraints which are used to assign security levels to the data. We discussed ways of handling integrity constraints as (1) integrity rules which must be satisfied by the data in the database and (2) derivation rules which are used to deduce new information. Much of our work on integrity constraints has been influenced by the work reported in [NICO78a, NICO78b].

In addition to the work reported in this paper, we are also developing various logics for multilevel knowledge base management systems. This is because NTML has been developed mainly for multilevel database systems. It does not provide the capability for reasoning with uncertain, imprecise and incomplete knowledge. Therefore, we are proposing extensions to NTML in order to support these features. The logics that we are developing are: Situational-NTML, Nonmonotonic-NTML, Fuzzy-NTML, Object-NTML, Temporal-NTML and Modal-NTML. Our work also includes an investigation of security issues of a system based on NTML. That is, we have defined security properties of NTML systems, and we are examining ways of showing that any system based on NTML is secure.

ACKNOWLEDGEMENT

We gratefully acknowledge the support of the Department of the Navy (SPAWAR) for sponsoring this work under contract F19628-89-C-0001.
REFERENCES


