Checking Classification Constraints for Consistency and Completeness

Selim G. Akl

Queen's University, Dept. of Computing and Information Science
Kingston, Ontario, Canada K7L 3N6

Dorothy E. Denning

SRI International, Computer Science Laboratory
333 Ravenswood Ave., Menlo Park, CA 94025

Classification constraints are rules for assigning access classes to data when they are entered into a database. In order that a given set of constraints specify meaningful classes, they should be consistent, that is, not define conflicting classes for the same data; and complete, that is, assign a class to all data. This paper gives algorithms for checking the consistency and completeness of a set of classification constraints defined on a database schema. The techniques use computational geometry to compute intersecting regions in a multidimensional space, where each region is defined by a classification constraint or integrity constraint.

1 Introduction

A *multilevel database system* is a database system that supports data having different *access classes* (security markings), where permission to access data is determined by the access class of the user requesting access along with the access class of the data. Classification can be applied at different levels of granularity in the database – for example, the relation, tuple, attribute, or element level.

In our earlier work [1], we introduced basic concepts for a multilevel-secure relational database system based on views. These concepts have since evolved into an informal policy for secure views [2], and we are presently developing a formal model that is consistent with the policy. The formal model is based on a multilevel relational data model [3] that consists of multilevel relations, with classification at the element level; a set of application-independent integrity constraints; a set of multilevel relational operators; and a method of decomposing all multilevel relations into single-level relations.

The security policy for the secure views model includes a labeling policy for determining the access class for new data when they are entered into a multilevel relation [1,2]. The policy states that the labels are to be determined according to rules called *classification constraints*, which are like integrity constraints except that they apply to classification values rather than to data values. In the multilevel relational data model, the classification label associated with a data attribute A is modeled as a classification attribute C associated with A, so that a classification constraint on A becomes an integrity constraint on C.

In order that a given set of constraints specify meaningful classes, they should be consistent and complete [1]. A set of constraints is said to be *consistent* if no two constraints, both of which must be satisfied simultaneously, define conflicting classes. A set of constraints is said to be *complete* if an access class is defined for each valid data element (i.e., one that satisfies the integrity constraints). Together, consistency and completeness mean that the classification constraints assign one and only one access class to each new data element.

There are two basic strategies for ensuring consistency and completeness. The first, which is applied at data entry time, essentially forces consistency and completeness as follows: When a new data element is entered, all constraints that could apply to the element are evaluated. If no constraint yields an access class for the element, then the element is considered to be invalid and rejected, thereby ensuring completeness. If multiple constraints yield different access classes, then a new constraint of higher priority is effectively defined that assigns the least upper bound of the classes to the element.

The second strategy is to analyze a set of constraints for consistency and completeness at schema definition time. This approach has the advantage of uncovering inconsistencies that might indicate inference problems [1] and of uncovering incomplete labeling rules.

This paper gives algorithms for checking the consistency and completeness of a set of classification constraints with respect to the database schema. The techniques use computational geometry to compute intersecting regions in a multidimensional space, where each region is defined by a classification constraint or integrity constraint. The algorithms are practical and efficient when the constraints are not too complex.

Denning and Morgenstern [4] have proposed a different method for checking consistency and completeness, which is based on logic programming. We do not wish to argue that either approach is better or worse than the other. Both merit further investigation and experimentation.

We shall first define classification constraints and integrity constraints as used by the algorithms, and then present the algorithms for checking consistency and competeness.

2 Constraints

Before defining classification and integrity constraints on multilevel relations, we first define a multilevel relation.

A *multilevel relation* R is modeled by a schema

$$R(A_1, C_1, \ldots, A_n, C_n) ,$$

where C_i is a classification attribute representing the access class labels associated with data attribute A_i.

We will assume that the domain of each data attribute A_i is defined on the set of all real numbers. Clearly, integer entries are a simple special case, and nonnumerical data can be (first mapped to and then) treated as integers. Thus, we lose no generality by this assumption.

Each C_i is defined by a minimum and maximum access class, which specifies a sublattice of the access class lattice [1]. We will write "class(A)" to denote the classification attributes associated with a single attribute A or a list of attributes, also denoted A. We will also write $R.A$ to denote attribute A in relation R, where the name A might otherwise be ambiguous.

2.1 Classification Constraints

A *classification constraint* S is a rule that specifies values for one or more of the classification attributes C_i. Formally, each rule S is a 4-tuple of the form:

$$S = (R, A, E, L) , \qquad (1)$$

where R is the name of a relation, A is a list of one or more data attributes in R, E is an optional expression, and L is an access class. The rule is interpreted as follows:

if E **then** class($R.A$) = L .

The class L is an expression, which is either a constant — e.g., "CONFIDENTIAL" or "SECRET", or contains one or more variables — e.g., "class(B)" or "class(B) $\oplus$ class(C)," where $\oplus$ denotes the least upper bound operator. Two classes L_1 and L_2 are considered to be equal if and only if they are identical expressions symbolically. Thus, we do not consider class(B) to be equivalent to SECRET even if class(B) always evaluates to SECRET for all valid data elements. This interpretation greatly simplifies the algorithm for checking consistency, and means that we do not have to interpret the expressions L. The only drawback is that we may flag constraints as being inconsistent that are actually consistent.

The expression E is a conjunction of one or more conditions to be satisfied by a collection of attributes in the database. An example of a complex expression is:

$$E = (R1.A \geq 0) \wedge (R1.A = R2.A) \wedge (0 \leq R1.B + R2.B \leq 100) .$$

We will not include disjunctions in E because disjunctions can be represented by separate constraints; for example, the constraint

$$(R, A, E_1 \vee E_2 \vee \cdots \vee E_k, L)$$

is equivalent to the set of constraints:

$$\begin{array}{c} (R, A, E_1, L) \\ (R, A, E_2, L) \\ \vdots \\ (R, A, E_k, L) \end{array}$$

Thus, because expressions in disjunctive normal form can be represented as multiple constraints, we have lost no generality by restricting each constraint to a conjunction.

There are four types of classification constraints [1]:

- *Type-dependent* — The expression E is absent and the class L is a constant, so that all elements associated with an attribute have the same class. This type of constraint defines a single-level attribute. Examples are:

  (R, A, , SECRET)
  (R, (A, B), , PROPRIETARY)

- *Value-dependent* — Either E is present or else L is a variable expression (or both), so that the access class associated with an element depends on the value of the element, or on the value or access class of other data in the database. Examples are:

  (R, A, $A = 1$, CONF)
  (R, A, $A = 2$, SECRET)
  (R, A, , class($R.B$))
  (R, A, $A = 1$, class($R.B$))

  In the first two examples, the class of an element associated with attribute A is determined by the value of the element. In the third example, it is determined by the class of attribute B for that tuple. In the fourth example, it is determined by both.

- *Source-level* — The access class L is the expression "class(user)," which is the access class of the user (subject) entering the element. The expression E may or may not be present. Examples are:

  (R, A, , class(user))
  (R, B, $B > 0$, class(user))

- *Source-label* — The access class L is the expression $\star$, which means that the subject entering the element supplies an access class for the element. The expression E may or may not be present. Examples are:

  $(R, A, , \star)$
  $(R, B, B > 0, \star)$

2.2 Integrity Constraints

We consider integrity constraints that are of the same form as the expressions E in classification constraints; that is, conjunctions of one or more conditions to be satisfied by attributes in the database.

Given a database with m integrity constraints, $I_1 \ldots I_m$, we define a *database state space*, denoted by D, as the intersection of all integrity constraints:

$$D = \cap_{k=1}^{m} I_k \,. \tag{2}$$

The state space D represents the set of values any instance of the database is allowed to take. We assume that D includes *domain integrity constraints* that limit the range of allowable values for an attribute — e.g., "$1 \leq A \leq 9999$," as well as *semantic integrity constraints* that relate the values associated with different attributes in one or more tuples — e.g., "$A+B = 100$," and "$(R1.A = R2.A) \wedge (R1.B+R2.B = 100)$."

We assume that the integrity constraints do not contain any inherent contradictions; that is, $D \neq \emptyset$. Furthermore, we assume that each expression E_i in a classification constraint does not contradict the integrity constraints; that is, $E_i \cap D \neq \emptyset$.

3 Consistency

A set of classification constraints is said to be *consistent* if and only if for every instance of the database in the state space D, no two constraints define conflicting access classes for the same data element.

Example: Consider the following classification constraints on attributes FLIGHTNO, DEST, and DEPART-TIME of a relation FLIGHT:

(FLIGHT, (FLIGHTNO, DEST, DEPART-TIME),
(DEPART-TIME $\leq$ 500), SECRET)

(FLIGHT, (FLIGHTNO, DEST, DEPART-TIME),
(1 $\leq$ DEST $\leq$ 2), TOP-SECRET)

(FLIGHT, (FLIGHTNO, DEST, DEPART-TIME),
(DEST > 2 $\wedge$ DEPART-TIME > 500),
class(user))

Suppose the integrity constraints permit a tuple with the following values to be inserted into FLIGHT:

(FLIGHTNO = 1735, DEST = 1,
DEPART-TIME = 450)

Then the constraints are inconsistent because the elements of the tuple satisfy both of the first two constraints, but the constraints do not assign identical access classes. However, the constraints would not be inconsistent under the following integrity constraint:

$$(1 \leq \text{DEST} \leq 2) \wedge (\text{DEPART-TIME} > 500)$$

(which rules out the preceding update) because then no two of the classification constraints are ever satisfied simultaneously. □

To determine whether a given set set of classification constraints is consistent, we need only determine whether each pair of constraints S_i and S_j is consistent. This is because the set as a whole is consistent if and only if it is pairwise consistent. A given pair of constraints S_i and S_j is consistent if any one of the following conditions is satisfied:

1. $L_i = L_j$ — that is, both constraints assign the same access class.
2. $A_i \cap A_j = \emptyset$ — that is, S_i and S_j place constraints on disjoint attribute sets.
3. $E_i \cap E_j = \emptyset$ — that is, both constraints cannot be satisfied simultaneously.
4. $E_i \cap E_j \cap D = \emptyset$ — that is, both constraints do not simultaneously satisfy all integrity constraints.

As mentioned earlier, we interpret the condition $L_i = L_j$ to mean that the expressions are identical symbolically. For this reason, the conditions are sufficient, but not necessary, for consistency. The main reason for using symbolic equivalence is to avoid the complication of determining whether two value-dependent constraints such as the following assign the same class:

$(R1, A, A = R2.B, \text{class}(R2.B))$
$(R1, A, A = R3.B, \text{class}(R3.B))$

In addition, it is not possible to define necessary conditions that can be determined from the schema alone. This is because a constraint can assign a class that is dependent on the user entering the data. For example, the following two constraints are consistent only if the user always enters the class SECRET:

$(R, A,$,SECRET)
$(R, A, , \star)$

Because the user could enter some class other than SECRET, it seems better to disallow such constraints. However, we do not regard our requirement for symbolic equivalence of access classes as a serious limitation of our method because constraints that are equal in value are redundant anyway.

The preceding conditions lead directly to the algorithm shown in Figure 1 for checking consistency:

Figure 1: Algorithm for Checking Consistency

Algorithm 1 (Consistency Checking)

for each pair of constraints
$S_i = (R, A_i, E_i, L_i)$, $S_j = (R, A_j, E_j, L_j)$
do
if $L_i = L_j$ **then** return(consistent)
else if $A_i \cap A_j = \emptyset$ **then** return(consistent)
else do
$E = E_i \cap E_j$;
if $E = \emptyset$ **then** return(consistent)
else do
$E = E \cap D$;
if $E = \emptyset$ **then** return(consistent)
else return(inconsistent)
od
od
od

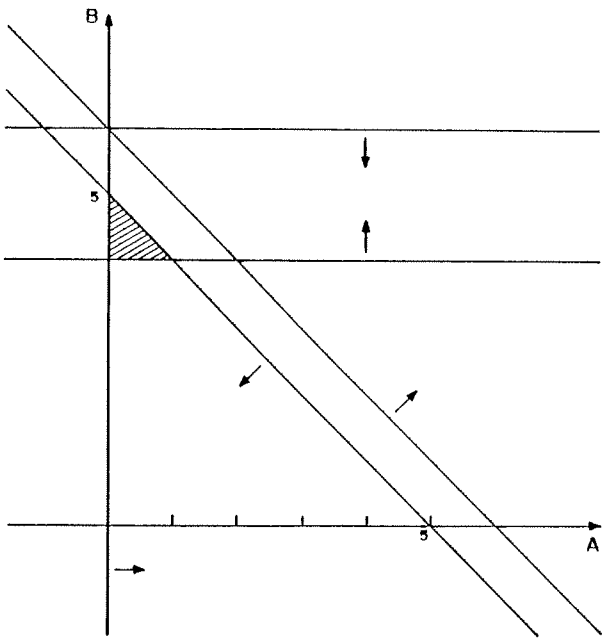
The algorithm involves computing intersections for pairs of constraints $E = E_i \cap E_j$. This is done as follows: Let d be the number of attributes that E_i and E_j have in common. Then E_i and E_j each defines a region in d-dimensional space. The intersection $E = E_i \cap E_j$ is thus evaluated by computing intersections of these regions using standard techniques for solving inequalities in computational geometry (e.g., see Preparata and Shamos [5]). Similarly, the intersection of E with the integrity constraints $I_k \in D$ is evaluated. One simple way of determining whether $E = \emptyset$ is to compute the intersections of all pairs of conditions involved in the expression. If at least one intersection point satisfies all the conditions, then E is nonempty.

Example: Consider the following two classification rules and integrity constraint:

$S_i = (R, A, ((4 \le R.B \le 6) \wedge (R.A \ge 0)),$ SECRET)
$S_j = (R, A, (R.A + R.B \le 5),$ TOP-SECRET)
$I_k = R.A + R.B \ge 6$

The intersection of $E_i = ((4 \le R.B \le 6) \wedge (R.A \ge 0))$ and $E_j = (R.A + R.B \le 5)$ is represented by the shaded triangle in Figure 2. Because this area is nonempty, we must turn to the integrity constraint, where we find that $E_i \cap E_j \cap I_k = \emptyset$. □

Figure 2: Intersecting Regions for Consistency Checking



To determine the complexity of the consistency checking algorithm, let a_{ijr}, e_{ijr}, and d_{ijr} be the number of steps required to compute $A_i \cap A_j$, $E_i \cap E_j$, and $E \cap D$, respectively, for each pair of constraints S_i and S_j and relation r. Then the number of steps required by Algorithm 1 is bounded by:

$$T1 = \sum_{i,j,r:i \neq j} a_{ijr} + e_{ijr} + d_{ijr} \; . \qquad (3)$$

In order to express this more concretely, let us assume that the database contains N relations, n classification constraints, and m integrity constraints; and that each relation is of degree g (i.e., contains g attributes). If the attributes in the lists A_i and A_j are sorted, then each intersection $A_i \cap A_j$ can be computed in linear time by comparing entries in the lists; thus, $a_{ijr} = O(g)$. Further, if all expressions E_i are *two-dimensional linear* — i.e., are a linear function of at most two attributes (as in the preceding example) — then $E_i \cap E_j$ is obtained by computing the intersection of two straight lines; hence, $e_{ijr} = O(1)$. Similarly, if all I_k are two-dimensional linear, then $d_{ijr} = O(m^2)$. Because n^2 pairs of constraints must be examined for each of N relations, the complexity of algorithm 1 is thus given by

$$T1 = O(Nn^2(g + m^2)) \; . \qquad (4)$$

If g is small compared to m, then Formula (4) reduces to $O(Nn^2m^2)$.

4 Completeness

Given a set of classification constraints that has been shown to be consistent, we now wish to determine whether it is

complete. A set of constraints is said to be *complete* if and only if for every instance of the database in the state space D, each element is assigned an access class by at least one constraint.

The first step in the process of checking completeness is to determine bounds on the values that each attribute A can take in the database space D. These can be obtained by determining the extreme corners of D along the A-axis. We let $D(A)$ denote this space.

Next, for each classification constraint $S_i = (R, A_i, E_i, L_i)$, we define $D_i(A)$ for each attribute A in A_i as the body in space obtained by intersecting the following attribute ranges:

- The range $D(A)$ of values that can be taken by A, and
- The entire ranges of all attributes B in E_i (except A, if $A \in E_i$) as defined by $D(B)$.

Then, for a given attribute A, $\cup_{i:A \in A_i} D_i(A)$ is the body within which all the values of A must be classified for completeness. If this region is the same as $\cup_{i:A \in A_i}(D_i(A) \cap E_i)$, then all valid values of A are classified by the constraints.

The algorithm for testing completeness is shown in Figure 3. For each attribute A, it determines whether the set of all classification constraints S_i that classify A are complete with respect to A.

Figure 3: Algorithm for Checking Completeness

Algorithm 2 (Completeness Checking)

for each attribute A **do**
  obtain all constraints $S_i = (R, A_i, E_i, L_i)$ where $A \in A_i$
  if no such S_i exists, **then** return(incomplete for A)
  else do
    for each S_i **do**
      $D_i(A) = D(A) \cap (\cup_{B:B \in E_i \wedge B \neq A} D(B))$;
      $H_i(A) = D_i(A) \cap E_i$
      od
    if $\cup_i H_i(A) \neq \cup_i D_i(A)$
      then return(incomplete for A)
    od
  od
return(complete)

Example: Consider two attributes, A and B, with the following ranges:

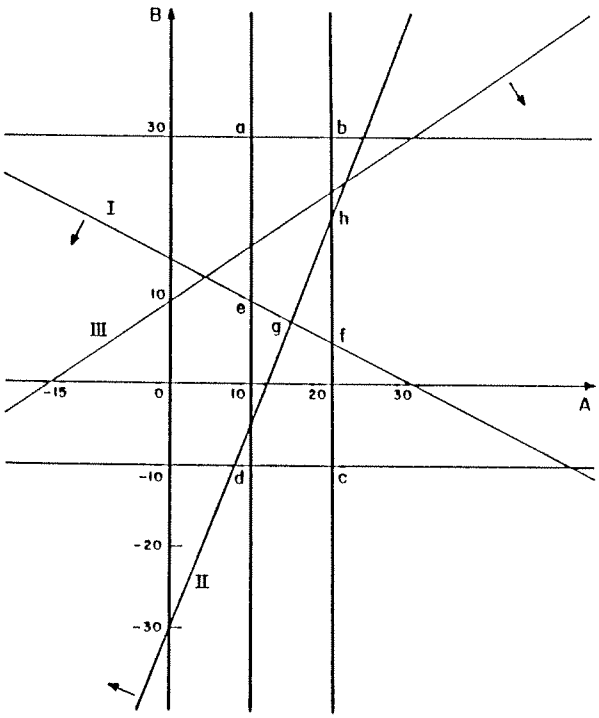
$$D(A) = 10 \leq A \leq 20$$
$$D(B) = -10 \leq B \leq 30\ ,$$

and three classification constraints on attribute A (we will not be concerned with classifying B here):

$$S_1 = (R, A, A + 2B \leq 30, \text{SECRET})$$
$$S_2 = (R, A, 5A - 2B \leq 60, \text{SECRET})$$
$$S_3 = (R, A, 3B - 2A \leq 30, \text{SECRET})\ .$$

The area for which A needs to be classified, namely $D(A) \cap D(B)$, is given by rectangle *abcd* in Figure 4.

Figure 4: Intersecting Regions for Completeness Checking



The condition in S_1 is satisfied by points c and d, but not by a and b. The quadrilateral $cdef$ is, therefore, classified by S_1, while the area $abfe$ remains to be classified. Constraint S_2 is satisfied by points a, b, and e, but not f. Thus, the region $abhge$ is classified by S_2, while ghf remains. Finally, S_3 is satisfied by all of the points g, h, and f, so that ghf is classified by S_3. Thus, the entire range $D(A)$ of values for A is completely classified, and the constraints are complete for A. (Because they all assign the class SECRET, they are also consistent.) □

In order to analyze the complexity of the algorithm, we define d_{iA} as the number of steps required to compute $D_i(A) = D(A) \cap (\cup_{B:B \in E_i, B \neq A} D(B))$; and h_{iA} as the number of steps required to compute $H_i(A) = D_i(A) \cap E_i$. Then the total number of steps required for Algorithm 3 is given by

$$T2 = \sum_{A, i:A \in A_i} (d_{iA} + h_{iA})\ . \qquad (5)$$

In order to be more concrete, assume that the database contains M attributes, each classified by at most $O(n)$ classification constraints, where n is the number of constraints. Further, let each E_i be two-dimensional linear, as in the preceding example. Then d_{iA} and h_{iA} are constant, and Formula (5) becomes:

$$T2 = O(Mn) . \tag{6}$$

5 Summary

We have given algorithms for checking the consistency and completeness of a set of classification constraints defined on a database schema. The algorithms use techniques from computational geometry to compute intersecting regions in a multidimensional space, where each region is defined by a classification constraint or integrity constraint. When the constraints are not too complex — e.g., are two-dimensional linear — then the algorithms are quite efficient.

Acknowledgments

This research was supported by the NSF under grant MCS-8313650.

To be presented at the 1987 Symp. on Security & Privacy, April 27-29, 1987, Oakland, CA

References

[1] D. E. Denning, S. G. Akl, M. Heckman, T. F. Lunt, M. Morgenstern, P. G. Neumann, and R. R. Schell. Views for multilevel database security. *IEEE Trans. on Software Eng.*, SE-13(2):129–140, Feb. 1987.

[2] D. E. Denning, T. F. Lunt, P. G. Neumann, R. R. Schell, M. Heckman, and W. Shockley. Security policy and interpretation for a class a1 multilevel secure relational database system. Nov. 1986. Computer Science Lab, SRI International.

[3] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. Shockley. A multilevel relational data model. In *Proc. of the 1987 Symp. on Security and Privacy*, IEEE Computer Society, 1987.

[4] D. E. Denning and M. Morgenstern. *Military Database Technology Study: AI Techniques for Security and Reliability.* Technical Report, Computer Science Lab, SRI International, August 1986.

[5] F. P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, New York, 1985.