Specification and Enforcement of Classification and Inference Constraints*

Steven Dawson
Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA
dawson@csl.sri.com

Sabrina De Capitani di Vimercati†
Dip. di Scienze dell'Informazione
Università di Milano
20135 Milano, Italy
decapita@dsi.unimi.it

Pierangela Samarati‡
Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA
samarati@csl.sri.com

Abstract

Although mandatory access control in database systems has been extensively studied in recent years, and several models and systems have been proposed, capabilities for enforcement of mandatory constraints remain limited. Lack of support for expressing and combating inference channels that improperly leak protected information remains a major limitation in today's multilevel systems. Moreover, the working assumption that data are classified at insertion time makes previous approaches inapplicable to the classification of existing, possibly historical, data repositories that need to be classified for release. Such a capability would be of great benefit to, and appears to be in demand by, governmental, public, and private institutions.

We address the problem of classifying existing data repositories by taking into consideration explicit data classification as well as association and inference constraints. Constraints are expressed in a unified, DBMS- and model-independent framework, making the approach largely applicable. We introduce the concept of minimal classification as a labeling of data elements that, while satisfying the constraints, ensures that no data element is classified at a level higher than necessary. We also describe a technique and present an algorithm for generating data classifications that are both minimal and preferred according to certain criteria. Our approach is based on preprocessing, or compiling, constraints to produce a set of simple classification assignments that can then be efficiently applied to classify any database instance.

*This work was supported in part by the National Science Foundation under grant ECS-94-22688 and by DARPA/Rome Laboratory under contract F30602-96-C-0337.

†This work was performed while the author was visiting SRI International, Computer Science Laboratory, supported in part by the National Science Foundation under grant ECS-94-22688.

‡On leave from Università di Milano. Author's permanent address: Università di Milano, Polo Didattico e di Ricerca di Crema, Via Bramante 65, 26013 Crema - Italy; e-mail: samarati@dsi.unimi.it.

1 Introduction

Information is today probably the most important and demanded resource. We live in an internetworked society that relies on the dissemination and sharing of information, in the private as well as in the public and governmental sectors. This situation is witnessed by a large body of research, and extensive development and use of shared infrastructures based on federated or mediated systems [27], in which organizations come together to selectively share their data. In addition, governmental, public, and private institutions are increasingly required to make their data electronically available. This often involves large amounts of historical data, once considered classified or accessible only internally, that must be made available outside.

This information sharing and dissemination process is clearly selective. Indeed, if on the one hand there is a need to disseminate some data, there is on the other hand an equally strong need to protect those data that, for various reasons, cannot be disclosed. Consider, for instance, the case of a private organization making available various data regarding its business (products, sales, etc.), but at the same time wanting to protect more sensitive information, such as the identity of its customers or plans for future products. As another example, government agencies, when releasing historical data, may require a sanitization process to "blank out" information considered sensitive, either directly or because of the sensitive information it would allow the recipient to infer. Effective information sharing and dissemination can take place only if the data holder has some assurance that, while releasing information, disclosure of private sensitive information is not a risk. Given the possibly enormous amount of data to be considered, and the possible inter-relationships between data, it is important that the security specification and enforcement mechanisms provide automatic support for the consideration of content-dependent specification, rather than requiring explicit labeling of each piece of data, and for complex security require-

ments, such as those due to inference channels.

Mandatory policies, providing a simple (in terms of specification and management) form of access control based on data and subject classification, appear suitable for the problem under consideration, where, in general, classes of data need to be released to classes of users. Unfortunately, capabilities of existing multilevel systems remain limited, and little, if any, support for the features mentioned above is provided. First, current DBMSs [2] work under the assumption that data are classified upon insertion (by assigning them the security level of the inserting subject) and therefore provide no support for the classification of existing, possibly unclassified, databases, where a different classification lattice and different classification criteria may need to be applied. Second, despite the large body of literature on the topic, they provide little or no support for expressing and combating inference channels [13]. In other words, although the access control models and techniques exist [2, 11, 14, 20, 29], deriving a correct and inference-free assignment of security classifications to data remains a major hurdle. This work represents a first step toward its resolution.

In this paper we consider the problem of classifying existing data repositories that are to undergo external release, where such release is governed by a mandatory policy. Our model provides support for explicit content-based data classification as well as for association and inference constraints. Association constraints express constraints on the association between values of different attributes. For instance, an association constraint can require the association of employees' `names` and `salaries` to be classified at a certain level. Inference constraints express constraints on the classification of information related by inference. There is an inference from a set of data elements to another data element if knowledge of the values in the set of data elements allows one to derive the value of the latter data element (exact inference), or to reduce it to a subset of possible values (inexact inference). For instance, there may exist an inference channel from `rank` and `department` to `salary`, meaning that knowing the values for `rank` and `department` would allow the recipient to infer the value of `salary`. To avoid improper leakage of information, only subjects cleared to see `salary` should be cleared to see both `rank` and `department`. Inference channels can also involve only specific values of the attributes (partial inference). For instance, the relationship above may hold only for those employees working in a specific department or in a given salary range. In our model, explicit classification as well as association and inference constraints can be expressed at the fine-grained element level, in a content-dependent fashion. Moreover, constraints are expressed in a unified, DBMS- and model-independent framework, making the approach widely applicable.

We characterize the problem of enforcing constraints in a way that minimizes information loss by ensuring that data are not overclassified. That is, data will not be withheld from release unless required for the satisfaction of the constraints. We introduce the concept of minimal classification as a labeling satisfying this property. Because of the potentially enormous amount of data to be classified (historical data repositories can contain records on the order of millions), direct enforcement of the constraints on the data is infeasible in practice. We therefore propose an approach based on preprocessing of the classification constraints to produce a set of simple classification assignments that can then be efficiently (in one pass) applied to produce a classified database. The basis of this approach is the observation that the number of attributes, and constraints, possibly with associated conditions, is orders of magnitude smaller than the number of elements recorded. In this way, the minimal classification can be computed more efficiently, without the need for accessing the data. The same assignments can then be efficiently enforced on different database instances that may need to be classified. We also describe a process for performing this classification.

It is important to note that our work does not propose a multilevel database model, nor is it intended as a substitute for current multilevel database systems and models [2, 11, 14, 20, 29]. Rather, it complements them with new and powerful data content and inference-related classification capabilities. Actual access control enforcement will require the support of a multilevel database. Classified databases produced by our approach can then be fed into any multilevel DBMS that will be responsible for enforcing access control.

2 Preliminaries

We assume standard notions from the relational database model. A *relation scheme* R is a finite set of *attributes*. A *database schema* $\mathcal{S}$ is a finite set of relation schemes $\{R_1, \ldots, R_n\}$. A *tuple* t is a mapping from a finite set $\mathcal{A}$ of attributes to a (possibly infinite) set $\mathcal{V}$ of *values*, where $t[A]$ denotes the mapping for attribute A in t. A *relation* r over relation scheme R is a finite set of tuples over R. A *database* over schema $\mathcal{S} = \{R_1, \ldots, R_n\}$ is a set of relations $\{r_1, \ldots, r_n\}$ where each r_i is a relation over R_i. $R_i.A$ denotes attribute A in R_i. Relation names may be omitted when clear from the context.

A *security lattice* $\mathcal{L}$ is a finite lattice $(L, \succeq)$, where L is a finite set of *security levels*, and $\succeq$ is a partial order on L called the *dominance relation*. The least and greatest elements of $\mathcal{L}$ are denoted $\bot$ and $\top$, respectively. Typical security levels are top secret (TS), secret (S), confidential (C), and unclassified (U), where $\mathsf{TS} \succeq \mathsf{S} \succeq \mathsf{C} \succeq \mathsf{U}$. In this paper we consider totally ordered security lattices. Examples will

refer to the levels TS, S, C, and U.

A *multilevel relation (with element-level labeling)* over relation scheme R and security lattice $\mathcal{L}$ is a pair (r, λ), where r is a relation over R, and λ is a mapping from elements in r to labels in $\mathcal{L}$, such that $\lambda(t[A]) = l$ if $t \in r$, A is an attribute in R, and $t[A]$ is classified at level $l \in L$. We denote by t^λ the pair (t, λ). We use the notation r^λ both as shorthand for the multilevel relation (r, λ) and to denote the multilevel relation resulting from application of a given mapping λ to an existing relation r. A *multilevel database (with element-level labeling)* over schema $\mathcal{S} = \{R_i, \ldots, R_n\}$ and lattice $\mathcal{L}$ is a set of multilevel relations $\{r_1^{\lambda_1}, \ldots, r_n^{\lambda_n}\}$, where each $r_i^{\lambda_i}$ is a multilevel relation over R_i and $\mathcal{L}$. We use the notation $\mathcal{B}^\lambda$ both as shorthand for a multilevel database $\{r_1^{\lambda_1}, \ldots, r_n^{\lambda_n}\}$, where $\lambda = \bigcup_{1 \leq i \leq n} \lambda_i$, and to denote the multilevel database resulting from application of a given mapping λ to each relation in an existing database $\mathcal{B}$. Figure 1 illustrates an example of relations r_1 and r_2 and corresponding multilevel relations $r_1^{\lambda_1}$ and $r_2^{\lambda_2}$ obtained by classifying all the elements in them. Note that we are interested here only in the relations' content with the corresponding classification and not in how multilevel relations would be actually represented in a specific DBMS. Depending on the specific DBMSs, some data/tuples may be duplicated (e.g., polyinstantiation constraints) or rearranged (e.g., if only tuple or key/nonkey attributes classification is supported). This transformation is outside the scope of this paper. Rules determining the security levels of data are assumed to be classified at a level at least as high as the level they assign. Hence, it is not possible for users to infer information on the values of some data by simply observing that the data are not visible to them (i.e., they are classified at a higher level) with the rules that lead to such a classification. However, if the fact that a piece of data is not visible at a certain level, because it is classified higher, can in itself leak information, a "fill in the blank" process can be executed whereby null values in low-level views are replaced by cover stories [7]. Note, however, that we assume the information recipients not to have update privileges (i.e., it is not possible for them to try to insert some values to determine whether a null value is actually a null value or is blanking out some protected data). Also, this assumption rules out inference channels resulting from classifying information at a level higher than the level of the subject who inserted it.

3 Classification Requirements Specification

We give preliminary definitions and concepts used in the remainder of the paper.

3.1 Classification constraints

Classification constraints define requirements that the security levels assigned to data elements must satisfy. We identify four different sources, and therefore possible classes, of constraints.

- *Basic classification constraints* explicitly assign a security level to certain attributes, possibly depending on some conditions, for instance, "attribute Salary is secret", "attribute Salary is secret when its value is greater than $1000", and "attribute Name is secret for employees whose Salary is greater than $1000".

- *Association constraints* classify the association between different attributes, possibly depending on some conditions, for instance, "the association between names and salaries is secret, when the Salary is greater than $1000".

- *Inference constraints* put conditions on the classification of attributes related by inference, for instance, with reference to the example in the introduction, the least upper bound of the security level of `rank` and `department` may be required to dominate the level of `salary`.

- *Classification integrity constraints* are constraints on the classification of related attributes that are required by the multilevel data model. Integrity constraints generally supported by multilevel databases require that for each tuple the key attributes be uniformly classified and their classification be dominated by that of the corresponding nonkey attributes (*primary key* constraints), and that the classification of an attribute representing a foreign key dominate the classification of the attribute for which it is the foreign key (*referential integrity* constraints). The fact of considering integrity constraints as input to the process ensures the general applicability of our approach.

We capture these four classes of constraints in a single general form of classification constraint consisting of two parts: a *labeling expression* and a *selection condition*. The labeling expression specifies a minimum security level, which may be expressed either as a level in the lattice (*absolute* constraint) or as the level of another attribute (*relative* constraint), that the least upper bound (lub) of some set of attribute elements must attain. The selection condition specifies a value or range of values of elements over the attributes to which the labeling expression applies.

Definition 3.1 (Labeling expression) *Given a schema $\mathcal{S}$ and a security lattice $\mathcal{L} = (L, \succeq)$, a* labeling expression *over $\mathcal{S}$ and $\mathcal{L}$ is an expression of the form*

r_1

M	N	O	P
a_1	b_1	5	e_1
a_2	b_1	8	e_2
a_3	b_2	27	e_3
a_4	b_3	13	e_4
a_5	b_4	2	e_5
a_6	b_2	10	e_6
a_7	b_5	11	e_7
a_8	b_6	27	e_8

r_2

F	G	H
e_1	3	10
e_2	5	1
e_3	1	7
e_4	17	6
e_5	0	14
e_6	5	13
e_7	2	87
e_8	37	35

$r_1^{\lambda_1}$

M	λ_1(M)	N	λ_1(N)	O	λ_1(O)	P	λ_1(P)
a_1	S	b_1	S	5	S	e_1	S
a_2	S	b_1	S	8	S	e_2	S
a_3	U	b_2	C	27	U	e_3	C
a_4	U	b_3	C	13	S	e_4	S
a_5	S	b_4	S	2	S	e_5	S
a_6	S	b_2	S	10	S	e_6	S
a_7	U	b_5	C	11	U	e_7	C
a_8	U	b_6	S	27	U	e_8	C

$r_2^{\lambda_2}$

F	λ_2(F)	G	λ_2(G)	H	λ_2(H)
e_1	C	3	C	10	TS
e_2	C	5	S	1	TS
e_3	C	1	C	7	TS
e_4	C	17	S	6	TS
e_5	C	0	C	14	C
e_6	C	5	S	13	C
e_7	C	2	C	87	C
e_8	C	37	S	35	C

Figure 1. An example of relations r_1, r_2 and corresponding multilevel relations $r_1^{\lambda_1}, r_2^{\lambda_2}$

$\mathsf{lub}\{\lambda(A_1), \ldots, \lambda(A_n)\} \succeq X$, *where* $n \geq 1$, $A_i \in \mathcal{A}$, $i = 1, \ldots, n$, *and* X *is either a security level* $l \in L$ *or is of the form* $\lambda(A)$, *with* $A \in \mathcal{A}$. *If* $n = 1$, *the expression may be abbreviated as* $\lambda(A_1) \succeq X$.

Definition 3.2 (Selection condition) *Given a schema* $\mathcal{S}$, *a* selection condition *over* $\mathcal{S}$ *is a conjunction of primitive conditions of the form* A *op* V, *where* $A \in \mathcal{A}$, *op is one of* $\{=, <, >, \leq, \geq, \neq\}$, *and* V *is either a constant* $V \in \mathcal{V}$ *or an attribute* $A' \in \mathcal{A}$.

Classification constraints can then be defined as follows.

Definition 3.3 (Classification constraint) *Given a schema* $\mathcal{S}$ *and a security lattice* $\mathcal{L}$, *a* classification constraint *over* $\mathcal{S}$ *and* $\mathcal{L}$ *is a pair* (e, s), *where* e *is a labeling expression over* $\mathcal{S}$ *and* $\mathcal{L}$, *and* s *is a selection condition over* $\mathcal{S}$.

Example 3.1 Figure 2 illustrates an example of classification constraints on relations r_1 and r_2 of Figure 1. Constraints c_1 through c_6 are basic classification constraints. Constraints c_7 through c_{11} express the primary key integrity constraints (M is key of r_1 and F is key of r_2). Constraint c_{12} expresses the referential integrity constraint (P is foreign key in r_1 pointing to F in r_2). Constraints c_{13} and c_{14} represent inference constraints, and c_{15} is an association constraint. △

The consideration of selection conditions only as conjunctions simplifies the treatment of constraints without loss of expressiveness. A classification constraint requiring a labeling expression to apply depending on a disjunction of conditions can be represented by multiple separate classification constraints. Also, note that all our constraints have the form $\succeq$ and security levels on the right-hand side only, that is, they specify a lower bound of the classification (which can be upgraded if needed for other protection requirements). For instance, a constraint requiring `salary` to be classified secret will be stated as $\lambda(\texttt{salary}) \succeq \mathsf{S}$, implying that `salary` must be classified *at least* secret. This interpretation is a property of the problem under consideration, where data classification may need to be upgraded to combat inference channels and to solve association constraints. The case where the classification of the attribute must be *exactly* secret can be captured as a preference of the classification process (Section 6).

$c_1 : \lambda(M) \succeq \mathsf{S}, O \leq 10$
$c_2 : \lambda(N) \succeq \mathsf{C}, O > 10$
$c_3 : \lambda(O) \succeq \mathsf{S}, O \leq 10$
$c_4 : \lambda(F) \succeq \mathsf{C}$
$c_5 : \lambda(G) \succeq \mathsf{S}, G \geq 5$
$c_6 : \lambda(G) \succeq \mathsf{C}, G < 5$
$c_7 : \lambda(N) \succeq \lambda(M)$
$c_8 : \lambda(O) \succeq \lambda(M)$
$c_9 : \lambda(P) \succeq \lambda(M)$
$c_{10} : \lambda(G) \succeq \lambda(F)$
$c_{11} : \lambda(H) \succeq \lambda(F)$
$c_{12} : \lambda(P) \succeq \lambda(F), P = F$
$c_{13} : \lambda(P) \succeq \lambda(O)$
$c_{14} : \mathsf{lub}\{\lambda(N), \lambda(O)\} \succeq \lambda(G), P = F$
$c_{15} : \mathsf{lub}\{\lambda(G), \lambda(H)\} \succeq \mathsf{TS}, H \leq 12$

Figure 2. An example of classification constraints

In the following, we use $rel(x)$ and $attr(x)$ to denote the relations and attributes appearing in x, where x can be a labeling expression e, its left-hand side $\mathsf{lhs}(e)$, its right-hand side $\mathsf{rhs}(e)$, or a selection condition s. Moreover, we refer to classification constraints in which the left-hand side of the labeling expression refers to a single attribute as *simple* classification constraints. All other classification constraints are *complex*.

Classification constraints can be interpreted as SQL-like integrity constraints over the multilevel relations resulting from the classification process. Each constraint (e, s) corresponds to the SQL-like integrity constraint "e IN $r_1^{\lambda_1}, \ldots, r_n^{\lambda_n}$ WHERE s," where $\{R_1, \ldots, R_n\}$ is $rel(e) \cup rel(s)$, and, for $1 \leq i \leq n$, $r_i^{\lambda_i}$ is a multilevel relation over R_i. This can be read as "in the multilevel database over schema $\mathcal{S}$ and lattice $\mathcal{L}$, the security levels assigned to elements must be such that the labeling expression e holds in all tuples t satisfying the selection condition s, written $t \models s$."

Classification constraints can be represented as a directed graph containing a node $\lambda(A)$ and a node l for each attribute $A \in \mathcal{A}$ and security level $l \in L$. Each constraint $(\mathsf{lub}\{\lambda(A_1), \ldots, \lambda(A_n)\} \succeq X, s)$ is represented by an edge labeled s from node $\lambda(A_1)$, if $n = 1$, or hypernode containing $\lambda(A_1), \ldots, \lambda(A_n)$, if $n > 1$, to node X. Figure 3 illustrates the graph representing the classification constraints in Figure 2, where, for simplicity, nodes representing attribute classification $(\lambda(A))$ are labeled only with the attribute name (A). In the figure, circle nodes represent attribute classification, square nodes represent security levels, and dashed ellipses represent hypernodes.

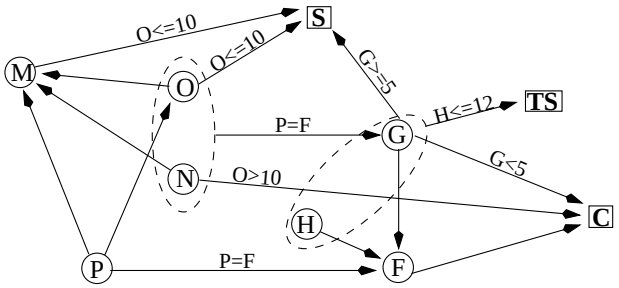

Figure 3. An example of classification constraint graph

3.2 Requirements and restrictions on classification constraints

For each database undergoing classification, a set C of constraints is specified whose enforcement allows the determination of a suitable classification for each possible data element in the database. To guarantee the possibility of defining a suitable classification for each and every element in the database, the set of classification constraints must be *complete* and *consistent*. A set of classification constraints is complete if it defines a classification for each possible element of the database. It is consistent if there exists an assignment of levels to the elements, that is, a definition of λ, that simultaneously satisfies all classification constraints. Intuitively, consistency guarantees the satisfiability of the classification constraints. Rather than requiring the Security Administrator to guarantee completeness of the specifications, we ensure it by adding a *default* classification constraint "$\lambda(A) \succeq \bot, \mathsf{true}$" for every attribute A in the database. In the following, we assume that these default classifications have already been added. Completeness is therefore trivially satisfied.[1] Consistency also is trivially satisfied because of the particular form of our constraints, whose labeling expression uses only the dominance relationship $\succeq$, and where security levels may appear only on the right-hand side. Thus, classification constraints requiring the same data element to dominate different levels can be satisfied simply by assigning to the data element the lub of those levels. Analogously, two constraints requiring two different data elements to mutually dominate each other, for instance, $\lambda(A) \succeq \lambda(B)$ and $\lambda(B) \succeq \lambda(A)$, can be satisfied by classifying both data elements at the same level. Given this, without loss of generality, we can make the following assumption.

Assumption 3.1 *The input set C of classification constraints is complete and consistent.*

Attributes appearing in a classification constraint may belong to different relations. For instance, constraint c_{14} in Figure 2 requires the lub of the level of N and O, belonging to relation r_1, to dominate the level of G, belonging to relation r_2. When the level of the elements of an attribute A in relation R depends on the value/level of elements of attributes in other relations $R_1, \ldots, R_n$, it is important that the conditions associated with the constraints define how to associate tuples in R with those in $R_1, \ldots, R_n$. To allow for our preprocessing and unambiguous interpretation and enforcement of the classification constraints, we require this tuple association to be consistent with respect to the condition evaluation. More precisely, we require each tuple (element of) to be classified to associate with the others on which it depends such that the result either satisfies or does not satisfy the conditions in the constraints. (Intuitively, we rule out the case where a tuple in a relation would associate with others resulting in both tuples that satisfy a given condition and tuples that do not satisfy it.) To express this requirement we first introduce the concept of key connection between relations. A key connection $kc_{i,j}$ between R_i and R_j is a conjunction of conditions $(R_i.A_{FK_1} = R_j.A_{K_1}) \wedge \ldots \wedge (R_i.A_{FK_n} = R_j.A_{K_n})$, where $(A_{FK_1}, \ldots, A_{FK_n})$ is a foreign key of R_i and $(A_{K_1}, \ldots, A_{K_n})$ is the corresponding key in relation scheme R_j. For instance, $P = F$ appearing in the constraints in Figure 2 represents a key connection between R_1 and R_2. Moreover, we introduce the definition of classification and evaluation context of an attribute, which will be used in the labeling process (see Section 7).

Definition 3.4 (Classification and evaluation context) *Let C be a set of constraints, and A be an attribute of a relation R. The* classification context *of A, written C_A, is the set $C_A \subseteq C$ defined as follows:*

1. $\forall (e_i, s_i) \in C : A \in attr(\mathsf{lhs}(e_i)) \Rightarrow (e_i, s_i) \in C_A$
2. $\forall (e_i, s_i) \in C : (\exists (e_j, s_j) \in C_A : attr(\mathsf{lhs}(e_i)) \cap (attr(\mathsf{lhs}(e_j)) \cup attr(\mathsf{rhs}(e_j))) \neq \emptyset) \Rightarrow (e_i, s_i) \in C_A$

The evaluation context *of A, written E_A, is the pair (R_A, KC_A), where $R_A = \bigcup_{(e,s) \in C_A} rel(e) \cup rel(s)$ and $KC_A = \bigcup_{(e,s) \in C_A} kc$ such that kc appears in s.*

Intuitively, the classification context of A is the set of constraints in C whose resolution affects the classification of A. In terms of the graph, C_A can be incrementally defined by including the set of constraints corresponding to edges leaving from A and hypernodes containing A, and transitively including then all constraints corresponding to edges leaving from a node/hypernode appearing in, contained in, or intersecting any node/hypernode representing

[1] Note that it is important that these default constraints be provided as input to our classification process. It is easy to see that assigning default classification $\bot$ at the end of the process to any element that did not get classified may result in a classification that does not satisfy the constraints.

a starting or arriving point of some edge in the context. The corresponding evaluation context defines the set of relations R_A whose content may affect $\lambda(A)$ and the key connection condition by which the relationships are linked.

We can now translate the above requirement to the requirement that each relation be uniquely key connected to the relations in its evaluation context, as stated in the following assumption.

Assumption 3.2 *Let C be a set of constraints, A be an attribute of relation R_i, and $E_A = (R_A, KC_A)$ its evaluation context. $\forall R_j \in R_A, R_j \neq R_i : \exists!$sequence$\langle kc_{i,1}, kc_{1,2} \ldots, kc_{n,j}\rangle$ in KC_A.*

Satisfaction of Assumption 3.2 ensures that each element t[A] corresponds to exactly one $t'[A]$ in the context in which it is labeled, that is, $\sigma_{KC_A}(r_1 \times \ldots \times r_n)$, where $(\{R_1, \ldots, R_n\}, KC_A)$ is the evaluation context of A.

4 Goals and Requirements

We define the correspondence between a set of constraints and a labeling mapping on the database and introduce the properties that the multilevel database and the labeling producing it are required to satisfy.

4.1 Correct and minimal classification

Given a database $\mathcal{B}$, a security lattice $\mathcal{L}$, and a set of constraints C, our goal is to produce a multilevel database $\mathcal{B}^\lambda$ whose labeling mapping λ satisfies the specified constraints. Given a tuple t over attribute set $\mathcal{A}$ and values $\mathcal{V}$, a lattice $\mathcal{L}$, a mapping λ from elements of t to levels in $\mathcal{L}$, and a labeling expression $e = \mathsf{lub}\{\lambda(A_1), \ldots, \lambda(A_n)\} \succeq X$ over $\mathcal{A}$ and $\mathcal{L}$, t^λ is said to *satisfy* e, denoted $t^\lambda \models e$, if and only if $\mathsf{lub}\{\lambda(t[A_1]), \ldots, \lambda(t[A_n])\} \succeq X$. A multilevel database $\mathcal{B}^\lambda$ over schema $\mathcal{S}$ and lattice $\mathcal{L}$ is said to *satisfy* a classification constraint $c = (e, s)$, denoted $\mathcal{B}^\lambda \models c$, if and only if every tuple t in the Cartesian product of all relations in c matching the selection condition s satisfies the labeling expression e. More formally, $\mathcal{B}^\lambda \models (e, s) \Leftrightarrow \forall t \in r' : t^{\lambda'} \models e$, where $(r', \lambda') = \sigma_s(r_1^{\lambda_1} \times \cdots \times r_n^{\lambda_n})$, $rel(e) \cup rel(s)$ is $\{R_1, \ldots, R_n\}$, and, for $1 \leq i \leq n$, $r_i^{\lambda_i}$ is a multilevel relation over scheme R_i and lattice $\mathcal{L}$.[2] Similarly, $\mathcal{B}^\lambda$ satisfies a set C of classification constraints, written $\mathcal{B}^\lambda \models C$, if and only if it satisfies every constraint in C. A database $\mathcal{B}^\lambda$ is said to be *correctly classified* with respect to a set C of classification constraints if it satisfies C, as formalized by the following definition.

[2]The selection and Cartesian product operators on multilevel relations are defined by assuming that the operations behave in the standard way (as in the case of unclassified relations), and that each element in the result maintains its original security level.

Definition 4.1 (Correct classification) *A multilevel database $\mathcal{B}^\lambda$ is* correctly classified *with respect to a set C of classification constraints if and only if $\mathcal{B}^\lambda \models C$.*

Given a database $\mathcal{B}$ and a set C of classification constraints, there may be many different assignments to the elements of the database that satisfy the constraints, that is, more than one such λ can be defined corresponding to different and correct $\mathcal{B}^\lambda$. However, not all of them are equally good. For instance, the labeling function that classifies all elements of all relations at the highest level in the lattice ($\top$) trivially satisfies any possible set of constraints. Such strong classification is clearly undesirable, unless required by the classification constraints, as it produces unnecessary information loss (not releasing information that could be safely released). Although the notion of information loss is difficult to make both sufficiently general and precise, it is clear that a first requirement in minimizing information loss is to prevent *overclassification* of data. That is, no database elements should be assigned security levels higher than necessary to satisfy the classification constraints. A database whose classification assignments meet this requirement is said to be *minimally classified*. Intuitively, a multilevel database $\mathcal{B}^\lambda$ is minimally classified with respect to a set C of constraints iff there does not exist another classification λ', $\lambda' \neq \lambda$ that assigns a classification lower than or equal to that assigned by λ to all the elements in $\mathcal{B}$. This is formalized by the following definition.

Definition 4.2 (Minimal classification) *Let $\mathcal{B}^\lambda$ be a multilevel database and C a set of constraints such that $\mathcal{B}^\lambda \models C$. $\mathcal{B}^\lambda$ is* minimally classified *with respect to C iff $\forall \lambda' \neq \lambda$: $\mathcal{B}^{\lambda'} \models C \Rightarrow \exists r_i \in \mathcal{B}, t \in r_i, A \in R_i, \lambda(t[A]) \not\succeq \lambda'(t[A])$.*

A labeling function λ is *minimal* for a database $\mathcal{B}$ and with respect to a set C of constraints, if $\mathcal{B}^\lambda$ is minimally classified with respect to C.

Example 4.1 Consider the classification constraints $(\lambda(A) \succeq \mathsf{TS}, \mathsf{true})$ and $(\mathsf{lub}\{\lambda(A), \lambda(B)\} \succeq \mathsf{S}, \mathsf{true})$. The mappings $\{\lambda(A) \mapsto \mathsf{TS}, \lambda(B) \mapsto \mathsf{U}\}$ and $\{\lambda(A) \mapsto \mathsf{TS}, \lambda(B) \mapsto \mathsf{S}\}$, are both solutions. However, the former is not minimal, since it classifies B at a level higher than necessary. △

4.2 Classification assignments

One possible approach to producing a minimally classified database from a given set of classification constraints is to evaluate the constraints directly over each database instance that needs to be classified, assigning security levels iteratively until a fixed point is reached for the database. Such an approach, however, is clearly inefficient, given the implied need to access the same data multiple times during the classification process, and to compute, store, and compare all the different solutions to determine the one to be

preferred. This is clearly far too expensive and impractical for general use.

Our approach to generating minimal classifications involves processing the classification constraints to obtain a set of much simpler classification assignments. These classification assignments can then be quickly applied to any instance of the given database schema to produce a minimally classified database. In effect, our process amounts to a preprocessing, or compilation, operation on the classification constraints. Classification constraints are processed only once, and the labeling mapping produced can then be efficiently reused for labeling multiple and large database instances. Given a set C of classification constraints over a database schema $\mathcal{S} = \{R_1, \ldots, R_n\}$, the product of our classification method is a set CA of *classification assignments* of the form $(\lambda(A) = l, s)$ that, when applied to a given database instance $\mathcal{B} = \{r_1, \ldots, r_n\}$, yield a correctly and minimally classified multilevel database $\mathcal{B}^\lambda$. To this end, the set *CA* of classification assignments must satisfy the following four properties.

Property 1 (Completeness) *For each database element, there is a classification assignment specifying its security level. Formally,* $\forall r_i \in \mathcal{B}, t \in r_i, A \in R_i, \exists(\lambda(A) = l, s) \in CA, t' \in r_1 \times \cdots \times r_n$ *such that* $t' \models s$ *and* $t[A] = t'[A]$.

Property 2 (Consistency) *No two classification assignments specify different security levels for the same database element. That is,* $\forall r_i \in \mathcal{B}, t \in r_i, A \in R_i : (\exists(\lambda(A) = l, s), (\lambda(A) = l', s') \in CA, t' \in r_1 \times \cdots \times r_n$ *such that* $t' \models s, t' \models s'$ *and* $t[A] = t'[A]) \Rightarrow l = l'$.

Property 3 (Correctness) *The database classified according to the classification assignments is correctly classified (Definition 4.1).*

Property 4 (Minimality) *The database classified according to the classification assignments is minimally classified (Definition 4.2).*

Intuitively, Properties 1 and 2 guarantee the definition of a $\mathcal{B}^\lambda$, ensuring for every element in $\mathcal{B}$ the existence of exactly one security level. Property 3 guarantees that the database classified by the assignments satisfies the classification constraints. Finally, Property 4 guarantees its minimality. It is important to note that completeness and consistency of the classification assignments is different from the completeness and consistency of the classification constraints. While the latter is trivially satisfied (see Section 3), the former is not — improper or naive processing of classification constraints could easily produce inconsistent classification assignments. The following sections provide a detailed description of our approach for generating classification assignments, or equivalently a labeling mapping, that satisfy the required properties.

5 Classification Strategies

Complex classification constraints offer multiple choices of attributes whose security levels must be constrained (raised). In the assumption of a totally ordered lattice, each complex constraint can be solved exactly by assigning the required level to any one of the attributes involved in the lub.[3] However, the choice of which attribute should be assigned the required level can affect whether the resulting solution is minimal. For instance, consider the two constraints $(\lambda(A) \succeq \mathsf{TS}, \mathsf{true})$ and $(\mathsf{lub}\{\lambda(A), \lambda(B)\} \succeq \mathsf{S}, \mathsf{true})$ of Example 4.1. Solving the lub constraint by upgrading, with respect to the lowest $\bot$ assigned as default, $\lambda(B)$ would produce the nonminimal solution $\lambda(A) = \mathsf{TS}$ and $\lambda(B) = \mathsf{S}$. The lub should be then solved by picking A.

We model the possible choices for solving a set C of constraints by decomposing each constraint in C into a set of simple constraints, where the decomposition of a simple constraint is the simple constraint itself. Given a classification constraint $c = (\mathsf{lub}\{\lambda(A_1), \ldots, \lambda(A_n)\} \succeq X, s)$, the *decomposition* of c, denoted D_c, is the set of simple classification constraints $D_c = \{(\lambda(A_k) \succeq X, s) \mid 1 \leq k \leq n\}$. C can then be satisfied by choosing one simple constraint from each set to arrive at one solution. Each possible combination of simple constraints from the decompositions of all constraints is called a *classification strategy*, or simply a *strategy*. The collection of strategies over C, denoted $\mathcal{T}_C$, is the set of all such strategies, that is, $\mathcal{T}_C = \{\{c'_1, \ldots, c'_n\} \mid c'_i \in D_{c_i}, 1 \leq i \leq n\}$.

Example 5.1 Consider the classification constraints in Figure 2. The decomposition of $(\mathsf{lub}\{\lambda(N), \lambda(O)\} \succeq \lambda(G),$ P=F) is $\{(\lambda(N) \succeq \lambda(G), P = F), (\lambda(O) \succeq \lambda(G), P = F)\}$. The decomposition of $(\mathsf{lub}\{\lambda(G), \lambda(H)\} \succeq \mathsf{TS}, H \leq 12)$ is $\{(\lambda(G) \succeq \mathsf{TS}, H \leq 12), (\lambda(H) \succeq \mathsf{TS}, H \leq 12)\}$. There are therefore four possible strategies: T_1, T_2, T_3, and T_4. In the remainder of the paper we assume that the complex constraints are solved by upgrading N and G in T_1, O and G in T_2, N and H in T_3, and O and H in T_4. △

Note that although all strategies have in common all the simple classification constraints, the effect of these constraints on the resulting classification may differ because of their integration with the specific decomposition of complex constraints. For instance, classification constraint $\lambda(P) \succeq \lambda(O)$ common to all the strategies has a different effect on the classification of P in each strategy, depending on how the two complex constraints given as input are solved.

The set of classification constraints that must be satisfied for a given element depends on what selection conditions are satisfied for that element. Since we are interested in

[3] Note that, because the schema is fixed, association constraints must be solved by explicitly upgrading some attribute (its values) appearing in the association.

determining the minimal classification for all database elements, we need to consider the possible combinations of conditions. We capture these combinations through the notion of *condition pattern*. Given a set S of selection conditions over a schema $\mathcal{S}$, a condition pattern p over S is a *satisfiable* conjunction of selection conditions containing, for each member $s \in S$, either s or its negation. Formally, $p = (\bigwedge_{s \in S} s')$, where $s' \in \{s, \neg s\}$ and p is satisfiable. A pattern p is satisfiable if there exists a database instance for which the pattern is applicable (i.e., evaluates to true). The satisfiability requirement, which, given the conjunctive form of our constraints can be easily checked, allows us to discard all the combinations of conditions that would never apply. In the following, given a set C of classification constraints, we consider the set P_{S_C} of condition patterns taken over the selection conditions, not including key connections, occurring in C, denoted S_C. Key connections are excluded in this process since, by definition, they are always satisfiable.[4] In addition, if a selection condition and its negation (e.g., $O \leq 10$ and $O > 10$) both occur in C, only one of them is included in S_C. For any possible set C of classification constraints, the condition patterns in P_{S_C} are disjoint (no two may be simultaneously satisfiable) and complete (every possible data context defined by the classification constraints is accounted for). Thus, each condition pattern can serve as an effective filter on the classification constraints that yields the smallest set of constraints that must be satisfied in a given context.

Example 5.2 Consider the classification constraints C in Figure 2. The condition set S_C is taken as $\{O \leq 10, G < 5, H \leq 12\}$. The set of all condition patterns P_{S_C} has the following eight members:

$p_1 : O \leq 10 \wedge G < 5 \wedge H \leq 12$ $\quad$ $p_5 : O \leq 10 \wedge G < 5 \wedge H > 12$
$p_2 : O \leq 10 \wedge G \geq 5 \wedge H \leq 12$ $\quad$ $p_6 : O \leq 10 \wedge G \geq 5 \wedge H > 12$
$p_3 : O > 10 \wedge G < 5 \wedge H \leq 12$ $\quad$ $p_7 : O > 10 \wedge G < 5 \wedge H > 12$
$p_4 : O > 10 \wedge G \geq 5 \wedge H \leq 12$ $\quad$ $p_8 : O > 10 \wedge G \geq 5 \wedge H > 12$

△

For each condition pattern, there is (at least) one strategy that leads to a minimal solution. However, the space of possible strategies is independent of the condition patterns. Thus, there is a natural separation of concerns in finding a minimal solution, in which we can first identify the (largest) constraint sets that must be satisfied for any strategy. Then, for each condition pattern, we can consider which strategy leads to a minimal solution for that pattern. The following section illustrates an approach to this.

6 Strategy Solution

Finding a minimal solution for a set of simple classification constraints, and hence for any particular strategy, is a relatively straightforward process. For each constraint on a given attribute, one can determine the "chains" of constraints ending in a security-level constant. The conjunction of selection conditions occurring along a chain of constraints is the condition under which the security level of an element over the attribute must dominate the level at the end of the chain. In cases where the conditions along two distinct chains from an attribute overlap, the level assigned to any element over the attribute must dominate the lub of the levels at the ends of the chains.

Taken together, the chains of constraints for an attribute, and the conditions under which they apply, capture all the relevant information for determining the minimal classification, according to a given strategy of that attribute's elements. Thus, they represent a kind of classification *template* from which the minimal classifications for the specific strategy under every condition pattern can be determined. The following definition formalizes this notion.

Definition 6.1 (Attribute classification template) *Given a set C of classification constraints, a strategy $T \in \mathcal{T}_C$, and an attribute A, the* attribute classification template *for A in T, denoted $template_T[A]$, is a set of pairs of the form (l, s) such that either $(\lambda(A) \succeq l, s) \in T$ or $\exists c_1 = (\lambda(A) \succeq \lambda(A_1), s_1), c_2 = (\lambda(A_1) \succeq \lambda(A_2), s_2), \ldots, c_n = (\lambda(A_{n-1}) \succeq l, s_n)$, with $s = s_1 \wedge \cdots \wedge s_n, c_i \in T, i = 1, \ldots, n$, and where $i \neq j \Rightarrow c_i \neq c_j, i, j \in \{1, \ldots, n\}$.*

Example 6.1 With reference to the constraints of Figure 2, Figure 4 illustrates the template for each attribute and each of the strategies of Example 5.1. For instance, the attribute classification template for attribute G in T_1 is $template_{T_1}[G] = \{(\mathsf{TS}, H \leq 12), (\mathsf{S}, G \geq 5), (\mathsf{C}, G < 5), (\mathsf{C}, \mathsf{true})\}$. △

Given the attribute classification templates for a given strategy, the minimum security level to be assigned to each attribute under a given condition pattern p, can be determined as the lub of all the levels whose associated conditions is consistent with p. The set of such levels for all attributes in the strategy is called a *conditional solution*.

Definition 6.2 (Conditional solution) *Given a set C of classification constraints, a strategy $T \in \mathcal{T}_C$, and a condition pattern $p \in P_{S_C}$, the* conditional solution *for T under p, denoted Sol_T^p, is $Sol_T^p = \bigcup_{A \in T}(A, Sol_T^p[A])$, where $Sol_T^p[A] = \mathsf{lub}\{l \mid (l, s) \in template_T[A]$ and $(s \wedge p)$ is satisfiable$\}$.*

Example 6.2 Consider the classification constraints of Figure 2. Table 1 lists the set of all conditional solutions for all possible strategies and condition patterns of Example 5.2. Each Sol_T^p corresponds to a column in the table. The crossing of row A with column T of pattern p indicates the level

[4] They will return in the relation labeling process (Section 7.2).

	$template_{T_1}$	$template_{T_2}$	$template_{T_3}$	$template_{T_4}$
M	(S, $O \leq 10$)	(S, $O \leq 10$)	(S, $O \leq 10$)	(S, $O \leq 10$)
N	(TS, $H \leq 12$),(S, $O \leq 10$),(S, $G \geq 5$), (C, $O > 10$), (C, $G < 5$), (C, true)	(S, $O \leq 10$),(C, $O > 10$)	(S, $O \leq 10$), (S, $G \geq 5$), (C, $O > 10$), (C, $G < 5$), (C, true)	(S, $O \leq 10$),(C, $O > 10$)
O	(S, $O \leq 10$)	(TS, $H \leq 12$),(S, $O \leq 10$), (C, $G < 5$), (C, true),(S, $G \geq 5$)	(S, $O \leq 10$)	(S, $O \leq 10$), (S, $G \geq 5$) (C, $G < 5$), (C, true)
P	(S, $O \leq 10$),(C, true)	(TS, $H \leq 12$),(S, $O \leq 10$), (S, $G \geq 5$),(C, true), (C, $G < 5$)	(S, $O \leq 10$),(C, true)	(S, $O \leq 10$), (S, $G \geq 5$), (C, $G < 5$), (C, true)
F	(C, true)	(C, true)	(C, true)	(C, true)
G	(TS, $H \leq 12$), (S, $G \geq 5$),(C, $G < 5$), (C, true)	(TS, $H \leq 12$),(S, $G \geq 5$), (C, $G < 5$), (C, true)	(S, $G \geq 5$), (C, $G < 5$), (C, true)	(S, $G \geq 5$),(C, $G < 5$), (C, true)
H	(C, true)	(C, true)	(TS, $H \leq 12$),(C, true)	(TS, $H \leq 12$),(C, true)

Figure 4. Attribute classification templates for T_1, T_2, T_3, and T_4

l such that $(A, l) \in Sol_T^p$. For instance, the conditional solution for strategy T_1 and condition pattern p_5 is $Sol_{T_1}^{p_5} = \{(M, \mathsf{S}), (N, \mathsf{S}), (O, \mathsf{S}), (P, \mathsf{S}), (F, \mathsf{C}), (G, \mathsf{C}), (H, \mathsf{C})\}$. △

Conditional solutions are specific to a strategy. Each is *locally* minimal within a given strategy (i.e., it upgrades each attribute at the lowest possible level to satisfy the constraints), but in general, not all will be *globally* minimal across all strategies. For a given set C of classification constraints and condition pattern p, a globally minimal solution for C with respect to p can be found by comparing the conditional solutions for all strategies T under p. This is captured by the following definition.

Definition 6.3 (Minimal conditional solution) *Let C be a set of classification constraints, $T \in \mathcal{T}_C$ be a strategy over C, and $p \in P_{S_C}$ be a condition pattern. A conditional solution Sol_T^p for T under p is said to be* minimal *if and only if $\forall T' \in \mathcal{T}_C \exists A \in C : Sol_T^p[A] \not\geq Sol_{T'}^p[A]$.*

The collection of minimal conditional solutions over all strategies $T \in \mathcal{T}_C$ under p, is denoted $MinSol_C^p$.

Example 6.3 All conditional solutions of Table 1, except those marked with a bullet, are minimal. $Sol_{T_4}^{p_3}$ is not minimal since the security levels assigned to all the attributes dominate the security levels assigned to all the attributes by conditional solution $Sol_{T_3}^{p_3}$. Analogously, $Sol_{T_2}^{p_7}$ and $Sol_{T_4}^{p_7}$ cannot be minimal because they dominate both $Sol_{T_1}^{p_7}$ and $Sol_{T_3}^{p_7}$. △

A minimal overall solution consists of a set of minimal conditional solutions, one for each condition pattern. In general, however, it is not possible to select any arbitrary minimal conditional solution for each pattern, because two distinct condition patterns may, in fact, coincide within the classification context of a particular attribute. Multiple strategies leading to distinct solutions for that attribute in such patterns would therefore result in inconsistent classifications. For instance, patterns p_1 and p_3, which differ only for condition $O \leq 10$, coincide for attribute G, whose classification is independent of the value of O. The strategies chosen as solutions for p_1 and p_3 must therefore agree on the classification assigned to G. For example, choosing either T_1 or T_2 for either p_1 or p_3 rules out the possibility of choosing T_3 for the other. To capture this constraint in the formal definition of minimal solution, we introduce the notion of projection of a condition pattern on the classification context of an attribute. Given a set C of classification constraints, a condition pattern $p \in P_{S_C}$ and an attribute A, the *projection* of p on the classification context of A, denoted p^A, is the conjunction of the conditions s in the pattern such that either s or $\neg s$ appears in the classification context of A. Then, a minimal solution is a set of minimal conditional solutions, one for each condition pattern, such that, for each attribute, if any two condition patterns coincide within the classification context for that attribute, the corresponding conditional solutions, that is, levels assigned by them, agree on that attribute.

Definition 6.4 (Minimal solution) *Given a set C of classification constraints, a* minimal solution *for C is a set of n pairs $\langle p_1, Sol_{T_{i_1}}^{p_1}\rangle, \ldots, \langle p_n, Sol_{T_{i_n}}^{p_n}\rangle$, where $Sol_{T_{i_j}}^{p_j} \in MinSol_C^{p_j}, 1 \leq j \leq n$, and $\forall x, y \in \{1, \ldots, n\}, \forall A \in C : {p_x}^A = {p_y}^A \Rightarrow Sol_{T_{i_x}}^{p_x}[A] = Sol_{T_{i_y}}^{p_y}[A]$.*

The collection of minimal solutions over C is denoted by $MinSol_C$.

Example 6.4 Consider the set of classification constraints shown in Figure 2 and the conditional solutions in Table 1. A minimal solution for these constraints, producing the multilevel relations of Figure 1, is $\{\langle p_1, Sol_{T_3}^{p_1}\rangle, \langle p_2, Sol_{T_3}^{p_2}\rangle, \langle p_3, Sol_{T_3}^{p_3}\rangle, \langle p_4, Sol_{T_4}^{p_4}\rangle, \langle p_5, Sol_{T_4}^{p_5}\rangle, \langle p_6, Sol_{T_2}^{p_6}\rangle, \langle p_7, Sol_{T_3}^{p_7}\rangle, \langle p_8, Sol_{T_1}^{p_8}\rangle\}$. △

In general, more than one minimal solution to a set of classification constraints may exist. Which solution is to be chosen may depend on specific criteria or preferences of the data holder/recipient. Possible preference criteria may be specified in terms of the cost of the classification, computed by associating a numeric value (cost) to each security level, summing the costs of the security levels returned by a solution, and preferring then the solution with the lowest cost [24]. Cost computation may also take into consideration weights of attributes (weighted cost) to express

		p_1				p_2				p_3				p_4				p_5				p_6				p_7				p_8			
		T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4
R_1	M	S	S	S	S	S	S	S	S	U	U	U	U	U	U	U	U	S	S	S	S	S	S	S	S	U	U	U	U	U	U	U	U
	N	TS	S	S	S	TS	S	S	S	TS	C	C	C	TS	C	U	C	S	S	S	S	S	S	S	S	C	C	U	C	U	C	S	C
	O	S	TS	S	S	S	TS	S	S	U	TS	U	C	U	TS	U	S	S	S	S	S	S	S	S	S	U	C	U	C	C	S	U	S
	P	S	TS	S	S	S	TS	S	S	C	TS	C	C	C	TS	C	S	S	S	S	S	S	S	S	S	C	C	C	C	C	S	C	S
R_2	F	C	C	C	C	C	C	C	C	TS	C	C	C	TS	C	C	C	C	C	C	C	C	S	C	S	C	C	C	C	S	C	S	C
	G	TS	TS	C	C	TS	TS	C	C	TS	C	C	C	TS	TS	C	S	C	C	C	C	S	C	S	C	C	C	C	C	S	S	S	C
	H	C	C	TS	C	C	C	TS	TS	C	C	TS	C	C	C	S	TS	C	C	C	C	C	S	C	S	C	C	C	C	C	C	C	S

Table 1. Conditional solutions for T_1, T_2, T_3, and T_4

the fact that not all attributes have the same importance for the release. Regardless of costs and weights, other possible policies may be those of upgrading the minimum possible number of attributes (maximum concentration), distributing the upgrading requirements over the maximum number of attributes (maximum flattening), maximizing the view at a certain level, accounting for explicit upgrading priorities, and requiring classification of a given attribute to be dominated by specified levels. This list is obviously not complete and other preference policies can be imagined, all applicable in different situations.

7 Determination and Enforcement of Classification Assignments

We describe an algorithm (*Classification Assignment*) that, given a set C of classification constraints over a schema $\mathcal{S}$ and a security lattice $\mathcal{L}$, derives a set of classification assignments satisfying the four properties discussed in Section 4.2. We also describe a procedure for labeling databases according to the generated assignments.

7.1 Classification Assignment algorithm

Figure 5 illustrates our Classification Assignment algorithm. The algorithm begins by determining the set of condition patterns (step 1), the simple constraint decomposition (step 2), and the set $\mathcal{T}_C$ of all possible strategies (step 3). Step 4 computes the attribute classification template for each attribute in the different strategies. The simple constraints common to all strategies are evaluated only once, producing a *basic-template* that is then updated for each strategy according to the constraints specific to the strategy. Computation of templates is performed by calling **compute-tmpl**. Given a set of constraints, and possibly an existent *basic-template*, **compute-tmpl** returns the template (Definition 6.1) for each attribute classified by the constraints. By keeping track of dependencies between attributes and levels as they are found (reachability information in graphical terms) and updating them whenever a new constraint is evaluated, the procedure performs the computation by evaluating each constraint exactly once. Dependencies are maintained in a structure *tmpl*$[A]$ for each attribute A, indicating the attributes/levels on which A depends (i.e., reachable from A). Upon evaluation of each constraint $(\lambda(A) \succeq X, s)$, the template of A, and those of each attribute Y depending on A are updated as follows. If X is a security level, *tmpl*$[A]$ is updated by adding to it the pair (X, s) and *tmpl*$[Y]$ is updated by adding to it a pair $(X, s \wedge s')$ for each (A, s') appearing in it. If X is an attribute level $\lambda(B)$, *tmpl*$[A]$ is updated by adding to it the pair $(Z, s' \wedge s)$ for each (Z, s') in *tmpl*$[B]$; *tmpl*$[Y]$ is updated by adding a pair $(Z, s'' \wedge s \wedge s')$ for each (A, s'') appearing in it and (Z, s') appearing in *tmpl*$[B]$. These new pairs express the dependencies of A due to the combination of the constraint with the dependencies of B and the dependencies of Y resulting by bridging those on A and those from B with the new constraint. After all constraints of a strategy T have been evaluated, each set $template_T[A]$ in step 4, which contains all the levels and attributes reachable from A, is updated by taking only the pairs (l, s), where l is a security level (reachability of attributes was needed only in the computation process and is not needed for the classification). Hence, step 5 computes the solution Sol_T^p for each condition pattern p and strategy T, and produces the set *MinStr*$_p$ of minimal solutions for p. Then, step 6 considers a classification pattern at a time and determines a preferred solution among the possible ones. Preferred solutions are determined by calling procedure `prefer`, omitted here for space constraints, that simply evaluates the different solutions according to the preference criteria specified. Every time a solution is chosen for a pattern p, the set *MinStr*$_{p_i}$ of possible solutions over patterns p_i coinciding with p is updated by eliminating those solutions that would result in inconsistent assignments (Section 6).

Example 7.1 Figure 6 is an example of classification assignments corresponding to the minimal solution of Example 6.4. For simplicity, multiple classification assignments defined for the same attribute and level are represented as a single assignment whose condition is the disjunction of all the condition patterns in the considered assignments. △

7.2 Labeling process

Once the classification assignments have been generated, applying them to an existing database can be done efficiently. We outline here a straightforward approach to applying the classification assignments that requires just one

Algorithm 7.1 *Classification Assignment algorithm*

INPUT: A set $C = \{c_1, \ldots, c_n\}$ of classification constraints over schema $\mathcal{S}$ and security lattice $\mathcal{L}$

OUTPUT: A set *CA* of classification assignments such that their application on a database $\mathcal{B}$ over schema $\mathcal{S}$ and lattice $\mathcal{L}$ produces a minimally classified multilevel database $\mathcal{B}^\lambda$ with $\mathcal{B}^\lambda \models C$

METHOD:

1. /* Determine the set P_{S_C} of condition patterns */
   Let S_C be the condition set associated with C.
   $P_{S_C} := \{p \equiv (\bigwedge_{s \in S_C} s') \mid s' \in \{s, \neg s\}$ and p is satisfiable$\}$
2. /* Decompose each constraint $c_i \in C$ into a set D_{c_i} of simple constraints */
   Let SC_C be the set of simple constraints in C and CC_C be the set of complex constraints in C. Let $c_i = (\mathsf{lub}\{\lambda(A_1), \ldots, \lambda(A_n)\} \succeq X, s)$.
   $D_{c_i} := \{(\lambda(A_k) \succeq X, s) \mid 1 \leq k \leq n\}$
3. /* Determine the set $\mathcal{T}_C$ of strategies over C */
   $\mathcal{T}_C := \{\{c'_1, \ldots, c'_n\} \mid c'_i \in D_{c_i}, 1 \leq i \leq n\}$
4. /* Compute the attribute classification template for each strategy */
   Let A_C be the set of attributes in C
   For each $A \in A_C$: *basic-template*$[A] := \{(A, \mathsf{true})\}$
   compute-tmpl(*SC*$_C$,*basic-template*)
   For each strategy $T \in \mathcal{T}_C$:
   Let T_C be the set of constraints in T resulting from the decomposition of complex constraints CC_C
   template$_T$:= *basic-template*
   compute-tmpl(T_C,*template*$_T$)
   For each $A \in A_C$:
   $template_T[A] := \{(l, s) \mid (l, s) \in template_T[A], l \in L\}$
5. /* Compute the minimal solutions for each pattern */
   For each $p \in P_{S_C}$:
   For each strategy $T \in \mathcal{T}_C$:
   For each $A \in A_C$:
   $Sol^p_T[A] := \mathsf{lub}\{l \mid (l, s) \in template_T[A]$ and $(s \wedge p)$ is satisfiable$\}$
   $Sol^p_T := Sol^p_T \cup \{(A, Sol^p_T[A])\}$
   $MinStr_p := \{T \mid \forall T' \in \mathcal{T}_C \exists A \in A_C : \ Sol^p_T[A] \not\succeq Sol^p_{T'}[A]\}$
6. /* Determine a set *CA* of classification assignments */
   Group all attributes in A_C into disjoint sets *AS*$_j$ such that each *AS*$_j$ contains all attributes A with the similar condition set S_{C_A}
   For each $p \in P_{S_C}$:
   pref-sol$_p$:= `prefer`(*MinStr*$_p$,*preference*)
   CA := *CA* $\cup \{(\lambda(A_i) = l, p) \mid (A_i, l) \in$ *pref-sol*$_p\}$
   $MinSol_C := MinSol_C \cup$ *pref-sol*$_p$
   For each *AS*$_j$:
   For each $p_i \neq p$ s. t. $p_i^{AS_j} = p^{AS_j} \wedge$ *pref-sol*$_{p_i} \notin MinSol_C$:
   $MinStr_{p_i} := MinStr_{p_i} \setminus \{T \mid T \in MinStr_{p_i}, \exists A_k \in AS_j$:
   $(A_k, l') \in Sol^{p_i}_T, (A_k, l) \in$ *pref-sol*$_p, l' \neq l\}$

Procedure compute-tmpl(*Constraints*,*tmpl*)
For each $(\lambda(A) \succeq X, s) \in$ *Constraints*:
Case X **of**
$X \in L$:*tmpl*$[A] :=$ *tmpl*$[A] \cup \{(X, s)\}$
For each Y, $Y \neq A$ such that $\exists (A, s') \in$ *tmpl*$[Y]$:
tmpl$[Y] :=$ *tmpl*$[Y] \cup \{(X, s' \wedge s) \mid (A, s') \in$*tmpl*[Y]$\}$
X=$\lambda(B)$:*tmpl*$[A] :=$ *tmpl*$[A] \cup \{(Z, s' \wedge s) \mid (Z, s') \in$ *tmpl*$[B]\}$
For each Y, $Y \neq A$, $Y \neq B$, s. t. $\exists (A, s'') \in$ *tmpl*$[Y]$:
tmpl$[Y] :=$ *tmpl*$[Y] \cup \{(Z, s'' \wedge s \wedge s') \mid$
$(Z, s') \in$ *tmpl*$[B], (A, s'') \in$ *tmpl*$[Y]\}$

Figure 5. Classification Assignment algorithm

$\lambda(M) = \mathsf{U}, p_3 \vee p_4 \vee p_7 \vee p_8$
$\lambda(M) = \mathsf{S}, p_1 \vee p_2 \vee p_5 \vee p_6$
$\lambda(N) = \mathsf{C}, p_3 \vee p_4 \vee p_7$
$\lambda(N) = \mathsf{S}, p_1 \vee p_2 \vee p_5 \vee p_6 \vee p_8$
$\lambda(O) = \mathsf{U}, p_3 \vee p_7 \vee p_8$
$\lambda(O) = \mathsf{S}, p_1 \vee p_2 \vee p_4 \vee p_5 \vee p_6$
$\lambda(P) = \mathsf{C}, p_3 \vee p_7 \vee p_8$
$\lambda(P) = \mathsf{S}, p_1 \vee p_2 \vee p_4 \vee p_5 \vee p_6$
$\lambda(F) = \mathsf{C}, \mathsf{true}$
$\lambda(G) = \mathsf{C}, p_1 \vee p_3 \vee p_5 \vee p_7$
$\lambda(G) = \mathsf{S}, p_2 \vee p_4 \vee p_6 \vee p_8$
$\lambda(H) = \mathsf{C}, p_5 \vee p_6 \vee p_7 \vee p_8$
$\lambda(H) = \mathsf{TS}, p_1 \vee p_2 \vee p_3 \vee p_4$

Figure 6. Example of classification assignments

classification pass through each relation to be classified. The first step in the labeling process is the determination of the evaluation context of each attribute, that is, the set of relations to be accessed (joined) to determine the attribute's level and the joining conditions. Figure 7 presents an algorithm for that process. The actual labeling process is facilitated by constructing a *decision tree* that captures all possible ways of classifying elements of one or more attributes that might occur in any database instance. It is neither strictly necessary nor desirable to construct a separate decision tree for each attribute of a relation. A single decision tree suffices for a set of attributes in a relation if the evaluation context (Definition 3.4) of each attribute in the set is contained in that of another attribute in the set, since the largest such evaluation context ensures the availability of all data from other relations to be evaluated. To this end, attributes are partitioned in sets such that all attributes in the same set X can share the same decision tree. In the following, we denote with S_X the selection conditions for the attributes in X, that is, $S_X = \bigcup_{A \in X} S_{C_A}$. For each set X so determined, a decision tree DT_X is defined as follows. Consider the conditions in $S_X = \{s_1, \ldots, s_n\}$ in any order. For each i from 1 to n, create 2^{i-1} nodes labeled s_i. Then, for each i from 1 to $n-1$, connect each node labeled s_i to two distinct nodes labeled s_{i+1}, with one edge labeled **T** (true) and the other labeled **F** (false). Create 2^n leaf nodes. Connect each node labeled s_n to two distinct leaf nodes, with one edge labeled **T** and the other labeled **F**. For every root-to-leaf path, label the leaf node with $\{p_1, \ldots, p_k\}$ for each condition pattern p_j such that every edge (s_i, s_{i+1}) along the path is labeled **T** (**F**) when selection condition s_i occurs positively (negatively) in $p_j, j = 1, \ldots, k$. Finally, for each attribute $A \in X$, a classification assignment of the form $\lambda(A) = l$ is placed at leaf $\{p_1, \ldots p_k\}$ if and only if $(\lambda(A) = l, p_j) \in CA, j = 1, \ldots, k$. Note that the fact that all patterns appearing in the leaf produce the same classification for attributes in X is guaranteed by the Classification Assignment algorithm (step 6). Figure 8 illustrates the decision tree for (all attributes of) relation schemas R_1 and R_2 of our running example.

Once the decision trees for all attributes of a relation scheme R have been constructed, any relation r over R

Algorithm 7.2 *Evaluation Context algorithm*

INPUT: A set $C = \{c_1, \ldots, c_n\}$ of classification constraints over schema $\mathcal{S}$ and security lattice $\mathcal{L}$

OUTPUT: The evaluation context $E_A = (R_A, KC_A)$ and the condition set S_{C_A} for each attribute A in C

METHOD:

Let A_C be the set of attributes in C

1. **For** each $A \in A_C$: R_A:=$\emptyset$; KC_A:=$\emptyset$; S_{C_A}:=$\emptyset$; *Affect*$[A]$:=A
2. **For** each constraint $(\mathsf{lub}\{\lambda(A_1), \ldots, \lambda(A_n)\} \succeq X, s)$:
   /* Determine the set *Rel* of relations to be added to R_{A_i}; key connections to be added to KC_{A_i}; selection conditions to be added to $S_{C_{A_i}}$; and the set *Affect*$[A_i]$ of attributes whose value affects A_i's level */
   2.1. **Case** X **of**
   - $X \in L$: *Rel* := $rel(s)$; *Key* := $\{kc \mid kc$ is a key connection in $s\}$; *Sel* := $\{s' \mid s'$ is a selection condition in s and s' is not a key connection$\}$; *Affect* := $\cup_{i=1}^{n}$ *Affect*$[A_i]$
   - X=$\lambda(B)$: *Rel* := $rel(s) \cup R_B$; *Key* := $\{kc \mid kc$ is a key connection in $s\} \cup KC_B$; *Sel* := $\{s' \mid s'$ is a selection condition in s and s' is not a key connection$\} \cup S_{C_B}$; *Affect* := $(\cup_{i=1}^{n}$ *Affect*$[A_i]) \cup$ *Affect*$[B]$

   2.2. **For** each $i = 1, \ldots, n$:
   $R_{A_i} := R_{A_i} \cup Rel$; $KC_{A_i} := KC_{A_i} \cup Key$; $S_{C_{A_i}} := S_{C_{A_i}} \cup Sel$; *Affect*$[A_i]$:= *Affect*

   2.3. **For** each $Y \notin \{A_1, \ldots, A_n\}$ s. t. $\exists A_i \in \{A_1, \ldots, A_n\}$, $A_i \in$ *Affect*$[Y]$:
   $R_Y := R_Y \cup Rel$; $KC_Y := KC_Y \cup Key$; $S_{C_Y} := S_{C_Y} \cup Sel$; *Affect*$[Y]$:= *Affect*$[Y] \cup$ *Affect*

Figure 7. Evaluation Context algorithm

can be efficiently classified by traversing the decision tree DT_X for each $t[X]$, with $t \in r$. The *Relation Labeling* algorithm, illustrated in Figure 9, works as follows. Step 1 initializes the labeling mapping λ_i. Step 2 computes, for each attribute set X_j, a relation $context_j$ containing all data necessary to select the proper classification assignment for any tuple over X_j. Intuitively, $context_j$ is the join of all relations in the evaluation context of X_j with join conditions of the key connections in the evaluation context. Note that $context_j$ is incrementally computed to ensure not only the correct association of join conditions with the relations, but also that whenever a condition $kc_{x,y}$ is applied, relation r_x is already included in $context_j$. Step 3 constructs the actual labeling mapping for the relation, r_i. For each tuple $t_i \in r_i$ and each attribute set X_j, the (unique) tuple t' from $context_j$ corresponding to t_i is selected and used to traverse the decision tree DT_{X_j} from root to leaf. For each $node$ in the tree, $node_{\mathbf{T}}$ ($node_{\mathbf{F}}$) denotes the next node reached by following the edge labeled **T** (**F**). Upon reaching the leaf,

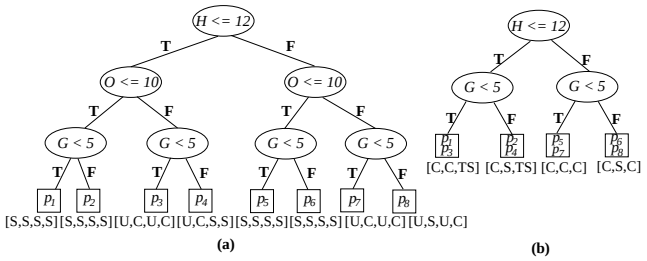

Figure 8. Classification decision tree for R_1 (a) and for R_2 (b)

the classification for each element of the tuple segment is generated according to the security levels specified at the leaf. Step 4 simply returns the resulting multilevel relation.

We conclude this section by noting that the decision tree construction outlined here can be optimized for space efficiency by using well-known techniques, such as BDDs [1] to produce a more compact representation.

8 Correctness, Complexity Evaluation, and Optimization

The correctness of our approach relies on the fact that (1) the classification assignments generated by Algorithm 7.1 satisfy the completeness, consistency, correctness, and minimality requirements, and (2) that the labeling process (Algorithm 7.3) correctly enforces them. This is stated by the following theorems, whose proofs are omitted here for space constraints.

Theorem 8.1 *The set CA of classification assignments produced by Algorithm 7.1 satisfies Property 1 (completeness), Property 2 (consistency), Property 3 (correctness), and Property 4 (minimality).*

Theorem 8.2 *Algorithm 7.3 correctly enforces the classification assignments produced by Algorithm 7.1.*

The computational cost of our approach derives from two main factors: the number of strategies $|\mathcal{T}_C|$ and the number of condition patterns $|P_{S_C}|$, since the classification assignment algorithm computes $|\mathcal{T}_C| \cdot |P_{S_C}|$ sets of classification assignments (each of size A_C, where A_C is the set of all attributes in C). Each such set of assignments is computed in time polynomial in the size of the input.[5] In the worst case, the number of different strategies is the product of the number of attributes in the left-hand sides of all labeling expressions. That is, if $C = \{(e_i, s_i) \mid 1 \leq i \leq n\}$, then

[5]The size of the input includes not only the number of classification constraints, but also the sizes of the labeling expressions and selection conditions in them.

Algorithm 7.3 *Relation Labeling algorithm*

INPUT: A database $\mathcal{B} = \{r_1, \ldots, r_n\}$ over a schema $\mathcal{S} = \{R_1, \ldots, R_n\}$, a designated relation $r_i \in \mathcal{B}$ over scheme $R_i \in \mathcal{S}$, a partition $\{X_1, \ldots, X_k\}$ of the attributes of R_i,and a decision tree DT_{X_j} for each X_j, $1 \leq j \leq k$

OUTPUT: A multilevel relation (r_i, λ_i), where λ_i is the labeling mapping defined by the set of classification assignments from which each DT_{X_j} was constructed

METHOD:

1. $\lambda_i := \emptyset$
2. **For** $j := 1$ to k
   /* Compute the data for the evaluation context of X_j. */
   Let (R_{X_j}, KC_{X_j}) be the evaluation context of X_j
   $context_j := r_i$
   $K := \{kc_{i,x} \mid kc_{i,x} \in KC_{X_j}\}$
   While $K \neq \emptyset$
      Select any $kc_{x,y} \in K$
      $context_j := context_j \bowtie_{kc_{x,y}} r_y$
      $KC_{X_j} := KC_{X_j} - kc_{x,y}$
      $K := (K - kc_{x,y}) \cup \{kc_{y,z} \mid kc_{y,z} \in KC_{X_j}\}$
3. **For** each $t_n \in r_i$
   For each X_j, $1 \leq j \leq k$
      Let $t \in context_j$ be the tuple such that $t[X_j] = t_n[X_j]$
      /* Walk down the decision tree. */
      $node :=$ root of DT_{X_j}
      While $node$ is not a leaf
         $s :=$ label of $node$
         if $t \models s$ **then** $node := node_{\mathbf{T}}$
         else $node := node_{\mathbf{F}}$
      /* Generate level mapping for each element in the tuple */
      $\lambda_i := \lambda_i \cup \{t_n[A] \mapsto l_A \mid A \in X_j\}$, where l_A is the security level specified for A at the leaf
4. **return** (r_i, λ_i)

Figure 9. Relation Labeling algorithm

$|\mathcal{T}_C| \leq \prod_{1 \leq i \leq n} |\mathsf{lhs}(e_i)|$. On the other hand, the number of condition patterns $|P_{S_C}|$ can be as high as 2^n. Although $|\mathcal{T}_C|$ can be quite large (possibly exceeding 2^n) if C consists mostly of complex constraints, we expect that $|P_{S_C}|$ will usually be the dominant factor in practice. The classification assignment process can then be optimized by reducing the number of selection conditions (and thus, the number of classification constraints) that must be considered together, since the number of distinct selection conditions determines the number of condition patterns. Reducing the number of condition patterns in this way will also tend to reduce the number of different strategies that must be considered. Along these lines we identify two approaches to reducing the cost of computing classification assignments: *constraint partitioning* and *incremental solution*.

Constraint partitioning consists of partitioning any input set of classification constraints into separately processable sets. To this end we observe that, if for two attributes A and B, neither $\lambda(A)$ depends (directly or indirectly) on $\lambda(B)$, nor $\lambda(B)$ depends on $\lambda(A)$, then the processes determining their classification are completely independent. In other words, the classification constraints pertaining to the two attributes can be solved independently. Thus, if an input set of constraints contains several such independent sets of attributes, it is possible to reduce one large constraint-solving problem to several smaller problems that can be solved more efficiently.

Incremental solution requires a more sophisticated evaluation of constraints, in which the attributes (and their associated constraints) are viewed as being ordered according to a level of dependency (of classification) on other attributes. For example, consider the constraints in Figure 2. The classification of attribute M does not depend on the classification of any other attribute, and thus, its constraints would be at the lowest dependency level. On the other hand, the classification of O depends directly on that of M and indirectly on that of N (via a lub constraint), and thus, its constraints would have a higher dependency level. Attribute classifications are then determined incrementally, from the lowest dependency level to the highest. This allows sharing of the computation performed in deriving classification assignments. More important, the incremental solution enables us to effectively "factor out" common portions in condition patterns, so that the enumeration of condition patterns becomes less of a concern, and furthermore the consistency checking across different strategies among coincident condition patterns is avoided.

9 Related Work

Our work has points in common with two main classes of work. The first is the work on view-based classification [4, 6, 18, 28], related to our support of content-dependent specification of classification. Unlike [4, 6, 28], consistent with the fact that we rely on existing DBMSs for access control enforcement, we consider content-dependent classifications as a means to determine the labeling to be associated with the data rather than to actually enforce access control. Denning et al., in the SeaView project [4, 14], first recognized the need for content-dependent classification constraints. Constraints are, however, checked only upon insert and update operations and only on the data being inserted (intuitively one tuple at a time), and not on the database itself, whose labeling may at some point not satisfy the constraints [5]. As already discussed, this insertion-based approach is not applicable to the problem under consideration. Like us, Qian [18] uses content-dependent specifications to provide data labeling. However, [18] considers only explicit data classification constraints and does not provide any support for inference constraints, assumes only tuple-level classification, and, as noticed by the author, may overclassify data.

The second class is the work on inference, related to our support of inference constraints. Most inference research addresses inference channels at the database design phase [3, 10, 19, 22, 26] or at query processing time [8, 17, 21, 25]. The proposals in the first category analyze the database schema to locate inference channels and eliminate them by upgrading selected schema components or redesigning the schema. The proposals in the second category evaluate database transactions to determine whether they lead to illegal inferences and, if so, disallow the query. Neither approach is applicable to the problem under consideration. Other inference work concerns the analysis of the database content, and possibly external information, to point out the existence of relationships among data that can introduce inference channels [9, 15, 16, 30]. These approaches are complementary to our work, and the inference relationships they determine can be provided as input to our process. The work closest to ours is represented by the work of Meadows [12], of Su and Ozsoyoglu [24] and of Stickel [23]. Meadows [12] proposes an approach to prevent leakage of high information due to release of data whose association is more sensitive than the pieces of data individually taken. While we solve this problem by explicitly upgrading individual data, the proposal in [12] keeps a history recording all the data released to an "environment" and denies the release of further data if their combination with data previously released would result in a security violation. To prevent easy bypassing of the constraints, the concept of environment encompasses both user and site identifiers. In addition, history logging and data association control crosses session boundaries. The constraints considered in [12] are a subset of the constraints considered by us, where the right-hand side is always an explicit security level and no conditions can be associated with the constraints (attribute-level classification is assumed). Su and Ozsoyoglu [24] consider the problem of upgrading data to block inference channels due to functional and multivalued dependencies. Their approach to the consideration of functional dependencies, given as input a set of attributes together with a proposed classification for them and a set of functional dependencies assumed to cause inference, returns an alternative inference channel-free classification for the attributes, obtained by upgrading the one provided as input. The approach by Su and Ozsoyoglu remains limited. The major limitations are that they assume attribute-level classification (i.e., classification can be specified only at the column level), consider only constraints within a single relation (and only due to functional dependencies), and base optimality of the result simply on assignments of weights to security values. Stickel [23] provides a formulation of the problem in terms of finding solutions using the Davis-Putnam theorem prover and discusses its modeling. This work, however, has the same limitations as [24].

10 Conclusions

Governmental, public, and private institutions are more and more frequently required to make data available for external release in a selective and secure fashion. Unfortunately, this ever-increasing need finds very little, if any, support in existing models and systems. The work presented in this paper aims to fill this gap by providing a framework for the specification and enforcement of classification constraints taking into consideration, at a fine-grained level, explicit data classification as well as association and inference constraints. The work reported represents only a starting point and leaves space for further developments. Future work, some of which we are currently investigating, includes the consideration of partially ordered lattices, the consideration of dynamic databases (i.e., subject to updates), the enrichment of the constraints, and the investigation of more efficient techniques for determining solutions, possibly guided by heuristics whenever preferences are not an issue.

References

[1] R.E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[2] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1995.

[3] H.S. Delugach and T.H. Hinke. Wizard: A Database Inference Analysis and Detection System. *IEEE Trans. on Knowledge and Data Engineering*, 8(1):56–66, February 1996.

[4] 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 Engineering*, 13(2):129–140, February 1987.

[5] D.E. Denning, T.F. Lunt, R. Schell, M. Heckman, and S. Shockley. Secure Distributed Data View (Sea View) – the Sea View Formal Security Policy Model. Technical report, SRI International, July 1987.

[6] C. Garvey. ASD_Views. In *Proc. of the IEEE Symp. on Research in Security and Privacy*, pages 85–95, Oakland, April 1988.

[7] T.D. Garvey and T.F. Lunt. Cover Stories for Database Security. In Carl E. Landwehr and Sushil Jajodia, editors, *Database Security V: Status and Prospects*, pages 363–380. North-Holland, 1992.

[8] J.T. Haigh, R.C. O'Brien, and D.J. Thomsen. The LDV Secure Relational DBMS Model. In S. Jajodia and C.E. Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 265–279, North-Holland, 1991. Elsevier Science Publishers.

[9] J. Hale and S. Shenoi. Catalytic Inference Analysis: Detecting Inference Threats due to Knowledge Discovery. In *Proc. of the 1997 IEEE Symp. on Security and Privacy*, pages 188–199, Oakland, May 1997.

[10] T. Hinke. Inference Aggregation Detection in Database Management Systems. In *Proc. of the IEEE Symp. on Research in Security and Privacy*, pages 96–107, Oakland, April 1988.

[11] S. Jajodia and R. Sandhu. Toward a Multilevel Secure Relational Data Model. In *Proc. of the 1991 ACM SIGMOD Conference*, pages 50–59, May 1991.

[12] C. Meadows. Extending the Brewer-Nash Model to a Multilevel Context. In *Proc. of the 1990 IEEE Symp. on Security and Privacy*, pages 95–102, Oakland, May 1990.

[13] S. Jajodia and C. Meadows. Inference Problems in Multilevel Secure Database Management Systems. In Marshall D. Abrams, Sushil Jajodia, and Harold J. Podell, editors, *Information Security - An Integrated Collection of Essays*, pages 570–584. IEEE Computer Society Press, 1995.

[14] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman, and W.R. Shockley. The SeaView Security Model. *IEEE Trans. on Soft. Eng.*, 16(6):593–607, June 1990.

[15] D.G. Marks. Inference in MLS Database Systems. *IEEE Trans. on Knowledge and Data Engineering*, 8(1):46–55, February 1996.

[16] M. Morgenstern. Security and Inference in Multilevel Database and Knowledge-Base Systems. In *Proc. of the 1987 ACM SIGMOD Conference*, pages 357–373, San Francisco, May 1987.

[17] A. Motro, D.G. Marks, and S. Jajodia. Enhancing the Controlled Disclosure of Sensitive Information. In *Proc. of the Fourth European Symp. on Research in Security and Privacy*, pages 290–303, September 1996.

[18] X. Qian. View-Based Access Control with High Assurance. In *Proc. of the 1996 IEEE Symp. on Security and Privacy*, pages 85–93, May 1996.

[19] X. Qian, M.E. Stickel, P.D. Karp, T.F. Lunt, and T.D. Garvey. Detection and Elimination of Inference Channels in Multilevel Relational Database. In *Proc. of the 1993 IEEE Symp. on Research in Security and Privacy*, pages 196–205, Oakland, May 1993.

[20] R. Sandhu and F. Chen. The Multilevel Relational (MLR) Data Model. *ACM Trans. on Information and System Security*, 1(1), November 1998.

[21] G.L. Sicherman, W. de Jonge, and R.P. van de Riet. Answering Queries Without Revealing Secrets. *ACM Trans. on Database System*, 8(1):41–59, March 1983.

[22] G.W. Smith. Modeling Security-Relevant Data Semantics. In *Proc. of the 1990 IEEE Symp. on Research in Security and Privacy*, pages 384–391, Oakland, 1990.

[23] M.E. Stickel. Elimination of Inference Channels by Optimal Upgrading. In *Proc. of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 168–174, Oakland, May 1994.

[24] T.A. Su and G. Ozsoyoglu. Controlling FD and MVD Inferences in Multilevel Relational Database Systems. *IEEE Trans. on Knowledge and Data Engineering*, 3(4):474–485, December 1991.

[25] B. Thuraisingham and W. Ford. Security Constraint Processing in a Multilevel Secure Distributed Database Management System *IEEE Trans. on Knowledge and Data Engineering*, 7(2):274–293, April 1995.

[26] B. Thuraisingham. The Use of Conceptual Structures for Handling the Inference Problem In C.E. Landwehr and S. Jajodia, editors, *Database Security, V: Status and Prospects*, pages 333-362, North-Holland, 1992. Elsevier Science Publishers.

[27] G. Wiederhold and M. Genesereth. The Conceptual Basis for Mediation Services. *IEEE Expert*, 12(5):38–47, September-October 1997.

[28] J. Wilson. Views as the Security Objects in a Multilevel Secure Relational Database Management System. In *Proc. of the IEEE Symp. on Research in Security and Privacy*, pages 70–84, Oakland, April 1988.

[29] M. Winslett, K. Smith, and X. Qian. Formal Query Languages for Secure Relational Databases. *ACM Trans. on Database Systems*, 19(4):626–662, December 1994.

[30] R.W. Yip and K.N. Levitt. Data Level Inference Detection in Database Systems. In *Proc. of the 11th IEEE Computer Security Foundations Workshop*, pages 179–189, Rockport, MA, June 1998.