Secure Databases: Constraints, Inference Channels, and Monitoring Disclosures

Alexander Brodsky, Csilla Farkas, and Sushil Jajodia, *Senior Member*, *IEEE*

Abstract—This paper investigates the problem of inference channels that occur when database constraints are combined with nonsensitive data to obtain sensitive information. We present an integrated security mechanism, called the *Disclosure Monitor*, which guarantees data confidentiality by extending the standard mandatory access control mechanism with a *Disclosure Inference Engine*. The Disclosure Inference Engine generates all the information that can be disclosed to a user based on the user's past and present queries and the database and metadata constraints. The Disclosure Inference Engine operates in two modes: *data-dependent* mode, when disclosure is established based on the actual data items, and *data-independent* mode, when only queries are utilized to generate the disclosed information. The disclosure inference algorithms for both modes are characterized by the properties of *soundness* (i.e., everything that is generated by the algorithm is disclosed) and *completeness* (i.e., everything that can be disclosed is produced by the algorithm). The technical core of this paper concentrates on the development of sound and complete algorithms for both data-dependent and data-independent disclosures.

Index Terms—Multilevel security, data confidentiality, inference problem, constraints, data-dependent disclosure, data-independent disclosure, inference algorithms, soundness, completeness, decidability.

1 Introduction

Information security policies in databases aim to protect the *confidentiality* (secrecy) and the *integrity* of data while ensuring data *availability*. In Multilevel Secure (MLS) Relational Database Management Systems (RDBMSs), direct violations of data confidentiality are prevented by mandatory access control (MAC) mechanisms, such as those based on the Bell-LaPadula (BLP) model. Mandatory policies are expressed via security classification labels, assigned to *subjects*, i.e., active computer system entities that can initiate requests for information, and to *objects*, i.e., passive computer system repositories that are used to store information. Classification labels, e.g., unclassified, confidential, secret, top-secret, form a mathematical lattice structure with a dominance relation among the labels. MAC policies control read and write operations on the data objects based on the classification labels of the requested data objects and the classification label (also called clearance) of the subject requesting the operation. For example, BLP policy ensures that a subject can read an object only if the subject's classification label dominates the object's classification label (simple-security property) and that a subject can write an object only if the object's classification label dominates the subject's classification label (star-property). However, MAC policies do not completely guarantee information secrecy. Illegal data accesses via *inference channels* may occur even if a properly functioning mandatory access control mechanism is present.[1] The detection and removal of inference channels are vital steps in providing secure database systems.

Database integrity constraints, such as functional, multivalued, and join dependencies, are especially important from the perspective of generating inference channels. Moreover, metadata that can be expressed by general Horn-clause constraints can also generate inference channels. The following example, which is similar to the one presented by Su and Ozsoyoglu [16], illustrates an inference channel via functional dependency. Let *(NAME, RANK, SALARY, EXPERIENCE)* be a relation schema, and assume that the relation *Employee* over this schema satisfies the functional dependency $RANK \rightarrow SALARY$. The security requirement is that only users with top-secret security clearances can access information about salaries of employees, i.e., users with secret or lower security clearances cannot access tuples with values for attributes *NAME* and *SALARY*. This classification scheme allows users with secret or lower security clearances to separately access these attributes, thus supporting data availability. However, while security violations via direct data access are prevented, the database still can be compromised by indirect data access. To see this, assume that a user u with secret security clearance requests the following two queries: "List the *RANK* and *SALARY* of all employees" and "List the *NAME* and *RANK* of all employees." None of the queries violates the security requirement because

- *C. Farkas is with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29212. E-mail: farkas@cs.sc.edu.*
- *A. Brodsky and S. Jajodia are with the Center for Secure Information Systems and the Department of Information and Software Engineering, George Mason University, Fairfax, VA 22030. E-mail: {brodsky, jajodia}@gmu.edu.*

Manuscript received 22 May 1998; accepted 29 June 1999. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 106784.

1. Covert channels are outside the scope of this paper.

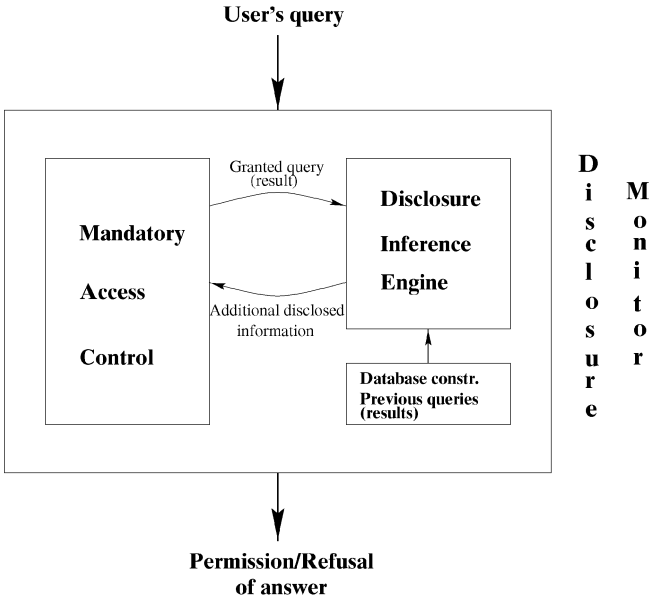


Fig. 1. Disclosure Monitor.

they do not contain the top-secret $< NAME, SALARY >$ pair. But clearly, because the relation *Employee* satisfies the functional dependency $RANK \rightarrow SALARY$ and the user knows the $< RANK, SALARY >$ pairs, the user can infer the salary of the employees.

1.1 Contributions

In this paper, we present an integrated security mechanism, called the *Disclosure Monitor*(DiMon) (see Fig. 1), which guarantees data confidentiality by extending a mandatory access control mechanism (MAC) with the *Disclosure Inference Engine* (DiIE). After a user's query is received, MAC enforces the standard security policy by allowing the user to read only the data that is dominated by the user's security clearance. If a security violation is detected, i.e., the requested data has higher security classification than the user's, the query is rejected. If direct security violation is not detected, the query is submitted to DiIE for further evaluation. DiIE generates all the information that can be disclosed by the user, based on the user's past and current requests and the database constraints. Then, the disclosed information is returned to MAC to be reevaluated. If no security violation is detected at this point, the query is answered and the user's *history-file* is updated.

The technical core of the paper introduces and studies the fundamental notions of *data-dependent* and *data-independent* disclosures. While motivated by our DiMon architecture, the developed notions of disclosure and their inferences are orthogonal to specific security architectures and, in fact, are applicable to a wide variety of (both design and query-time) security frameworks. We develop a Disclosure Inference Engine that operates in two modes. In *data-dependent* mode, the actual answers $P_1, \ldots, P_n$ to the user's past and present queries $Q_1, \ldots, Q_n$ and the constraints $\mathcal{D}$ are considered to generate the disclosed facts. In *data-independent* mode, only the past and present queries $Q_1, \ldots, Q_n$ (without the actual data items) and the constraints $\mathcal{D}$ are used to generate new queries which are disclosed.

We say that a fact PF (over a query Q) is data-dependently disclosed from query answers $P_1, \ldots, P_n$ (to queries $Q_1, \ldots, Q_n$, respectively) by a set of constraints $\mathcal{D}$ if PF is guaranteed to be in any relation r such that 1) r satisfies $\mathcal{D}$ and 2) $P_1, \ldots, P_n$ are the answers to the queries $Q_1, \ldots, Q_n$ on r. While the data-dependent mode provides the highest availability of data, it is computationally expensive and requires large storage overhead for maintaining a history-file. Disclosure can also be considered based solely on (past and current) queries, but independently of the (past or current) disclosed data. More specifically, a query Q is data-independently disclosed from queries $Q_1, \ldots, Q_n$ by a set of constraints $\mathcal{D}$ if there exists an example of data-dependent disclosure corresponding to queries $Q_1, \ldots, Q_n$, Q, and $\mathcal{D}$. (Formal definitions and properties of data-dependent and data-independent disclosure inferences are given in Sections 3, 4, and 5.) Data-independent disclosure is computationally less expensive and requires less storage overhead than the data-dependent disclosure, but it may indicate disclosures which are not present in a given database instance. However, without knowing the actual data, data-independent disclosure is the best one can achieve.

Our work is the first, to the best of our knowledge, that introduces a formal classification of disclosure inference algorithms by the properties of *completeness* and *soundness*. The *completeness* property ensures that the algorithm generates all of the disclosed information, thus no possible inference remains undetected. The algorithm is *sound* if the generated information (actual data in data-dependent mode and queries in data-independent mode) is indeed disclosed. Intuitively, soundness, along with completeness, provides the highest data availability without violating security. While it is desirable to develop algorithms that are both sound and complete, completeness is clearly more crucial than soundness from the perspective of data confidentiality. Without formal properties of disclosure inference, such as soundness and completeness, the strength of disclosure inference algorithms, in terms of providing availability of data and confidentiality, cannot be formally established.

We develop sound and complete disclosure inference algorithms for both data-dependent and data-independent modes. More specifically, we consider selection-projection queries of the form $\Pi_Y \sigma_C$, where C is a conjunction of equalities among attributes and constants. Furthermore, we consider a rich family of *Horn-clause constraints*, which are the extensions of (full) *generalized dependencies* [19] with constants.

For the data-dependent disclosure, we assume that disclosed items (i.e., security-labeled objects) may include specific (partial) tuples, attribute sets, equality-based selection conditions, or their combinations, and prove that the problem of disclosure inference is decidable. Moreover, we develop a disclosure inference algorithm (Algorithm 3) that we prove is both sound and complete.

For the data-independent disclosure, we assume that the disclosed items are selection-projection queries, in which selection conditions are equalities, and prove that

the problem of disclosure inference is decidable for the case when the selection conditions in queries and the constraints do not contain constants. Furthermore, for this case, we provide a sound and complete disclosure algorithm (Algorithm 4) that generates the set of data-independently disclosed queries. Moreover, we develop a general disclosure inference algorithm (Algorithm 5) that works without the above restrictions and is guaranteed to be complete.

1.2 Related Works and Their Limitations

Most of the inference channels in relational databases are raised by combining *metadata* (e.g., database constraints) with data in order to obtain information that has higher security classification than the original data (see [8] for a survey). Techniques to detect and remove inference channels can be organized into two categories: The first category includes techniques that detect inference channels *during database design*; any channels are removed by modifying the database design and/or by increasing the classification levels of some of the data items [2], [6], [7], [12], [17], [9], [13], [14], [16], [3], [4]. These techniques often result in overclassification of data and, therefore, reduce the availability of data. Techniques in the second category seek to eliminate inference channel violations during query time [5], [11], [15], [18]. If an inference channel is detected, the query is either refused or modified to avoid security violations.

Each of the categories above requires either data-dependent or data-independent inference algorithms. However, none of the above works has the formal notion of soundness and completeness for data-dependent and data-independent disclosure and, thus, cannot establish these formal properties of disclosure inference. Also, most authors [2], [6], [13], [14], [11], [15], [18], with the exception of [5], [7], [16], [3], [4], do not consider the problem of actual inference for specific families of constraints (and its decidability, soundness, completeness, etc.); rather, they develop a framework, assuming that disclosure inference algorithms are readily available. It is our view, however, that the main technical difficulty of solving the inference channel problem lies in developing (sound and complete) inference algorithms, especially for the data-independent case.

The work of Su and Ozsoyoglu [16] is the closest to ours in that they consider inference algorithms for specific constraint families, namely functional and multivalued dependencies. However, their inference methods are limited in several respects. First, as the source of illegal inferences, they consider only two types of dependencies, FDs and MVDs; inference algorithms are given for each of the two types separately, but not when they are present together, whereas our methods apply to Horn-clause constraints, which are more expressive than the combination of FDs and MVDs. Second, they use single-attribute security granularity for FD-compromise and tuple-level security granularity for MVD-compromise. In contrast, we propose a flexible security classification schema that can also express content and context-based security granularity, including selection-projection queries and arbitrary sets of (partial) tuples. While their inference algorithms seem to be sound and complete for single-attribute level security granularity, it would not be sound even for security granularity based on sets of attributes. Since their solution for eliminating a detected inference channel is based on increasing the classification level of individual attributes (i.e., single-attribute level granularity), it restricts the availability of data.

Finally, the scheme recommended by Su and Ozsoyoglu is database design-time-oriented rather than query-time-oriented, as in the case of DiMon. While design-time approach is easier to manage and implement, query-time approach allows more availability of data than design-time approach because more information (i.e., past and current data/queries) can be used for disclosure inference. Our disclosure inference algorithms are readily available to both design and query-time schemes (of our, only data-independent disclosure is relevant to the design-time approach.)

1.3 Organization of Paper

The rest of the paper is organized as follows: Section 2 defines the subjects and the objects and how security classification labels are assigned to them in our model and contains the algorithm used by DiMon. Section 3 defines the database constraints that form the basis of data disclosure, and introduces the notions of data-dependent and data-independent disclosures. The disclosure inference algorithms for both modes are given in Sections 4 and 5. Finally, Section 6 concludes and provides future research directions. Due to space limitation, proofs of all propositions can be found in [1].

2 Disclosure Monitoring Architecture

2.1 Data, Queries, and Security Classification

Our model is built upon lattice-based access control, therefore, it is necessary to define what the objects and the subjects are and how to assign security classification labels to them. For simplicity, we assume that users and *subjects*, executing on behalf of these users, are the same entities, thus we use the terms user and subject interchangeably. To formally define *objects*, we need the definition of projection facts, queries, and their combinations, as given below.

We consider *selection-projection* queries of the form $\Pi_Y \sigma_C$, where Π_Y denotes the projection of a relation on attributes Y and σ_C denotes the selection condition of the query, where C is a conjunction of equalities of the forms $A = B$ or $A = c$ such that A, B are attribute names and c is a constant. We assume the existence of a single, "universal" relation r with schema R. The symbol R is used to indicate both the relation name and the set of all its attributes.

In our model, it is possible to express complex security requirements dealing not only with content and context-based security restrictions, but different security granularity also, as demonstrated in the following example.

Definition 2.1 (Projection fact). *A* projection fact *(PF)* of type $A_{i_1}, \ldots, A_{i_k}$, *where* $A_{i_1}, \ldots, A_{i_k}$ *are attributes in* R, *is*

a mapping m from $\{A_{i_1}, \ldots, A_{i_k}\}$ to $\bigcup_{j=1}^{k} dom(A_{i_j})$ such that $m(A_{i_j}) \in dom(A_{i_j})$, for all $1 \leq j \leq k$. We will denote a projection fact by an expression of the form $R[A_{i_1} = a_1, \ldots, A_{i_k} = a_k]$, where R is the relation name and $a_1, \ldots, a_k$ are constants in $dom(A_{i_1}), \ldots, dom(A_{i_k})$, respectively. A relation instance r' over $R' \subseteq R$ can be viewed as a set of projection facts of type R'.[2]

Note that, since a projection fact is a mapping, the order of attributes $A_{i_1}, \ldots, A_{i_k}$ in $R[A_{i_1} = a_1, \ldots, A_{i_k} = a_k]$ is not important.

Definiton 2.2 (Query-answer pair). *An* atomic query-answer pair *(QA-pair) is an expression of the form $(PF, \Pi_Y \sigma_C)$, where PF is a projection fact over Y that satisfies C. A query-answer pair is either an atomic QA-pair or an expression of the form $(P, \Pi_Y \sigma_C)$, where P is a set of projection facts $\{PF_1, \ldots, PF_l\}$ such that every projection fact PF_i, $(i = 1, \ldots, l)$ is over Y and satisfies C.*

QA-pairs are used as security labeled objects. Specifically, an *object* in our security model can have one of the following forms:

1. An atomic QA-pair $(PF, \Pi_Y \sigma_C)$.
2. A query of the form $\Pi_Y \sigma_C$. This includes Π_Y representing $\Pi_Y \sigma_{TRUE}$ and σ_C, representing $\Pi_R \sigma_C$.
3. A set of atomic QA-pairs. As a security labeled object, a QA-pair $(P, \Pi_Y \sigma_C)$ will be interpreted as the set $\{(PF, \Pi_Y \sigma_C) \mid PF \in P\}$ and a set of QA-pairs

$$\{(P_1, \Pi_{Y_1} \sigma_{C_1}), \ldots, (P_n, \Pi_{Y_n} \sigma_{C_n})\}$$

will be interpreted as

$$\{(PF, \Pi_{Y_i} \sigma_{C_i}) \mid 1 \leq i \leq n,\ PF \in P_i\}$$

We will also use the shortcut $\Pi_Y \sigma_C(r)$ for $(P, \Pi_Y \sigma_C)$, where $P = \Pi_Y \sigma_C(r)$, as well as $\Pi_Y(r)$ for $\Pi_Y \sigma_{TRUE}(r)$ and $\sigma_C(r)$ for $\Pi_R \sigma_C(r)$. Furthermore, in data-dependent mode, $\Pi_Y \sigma_C$ will be interpreted as $\Pi_Y \sigma_C(r)$, where r is the current database relation, i.e., all QA-pairs of $\Pi_Y \sigma_C(r)$.

Definition 2.3 (Security classification). *Let SL be a set of security labels, e.g., unclassified, secret, top-secret, etc. A* security classification *is a triple $< \mathcal{O}, \mathcal{U}, \lambda >$, where $\mathcal{O} = \{o_1, \ldots, o_n\}$ is a set of security objects, $\mathcal{U}$ is a set of subjects (users), and $\lambda : \mathcal{O} \cup \mathcal{U} \rightarrow SL$ is a security classification mapping.*

Since some of the security objects in O may be sets of atomic QA-pairs, we denote by $A(O')$, for a subset O' of $\mathcal{O}$, the set of all atomic objects that appear in O', i.e.,

$$A(O') = \{o \mid o \in O' \text{ or there is a } o' \in O' \text{ such that } o \in o'\}.$$

To ensure data confidentiality, the access control mechanism has to guarantee that the users cannot access objects unless their security clearance is greater than or equal to the security classification of the objects, i.e., $\lambda(user) \geq \lambda(object)$.

2. Often, the name *partial tuples* is used in literature.

Because $\mathcal{O}$ does not necessarily contain all of the objects that may be requested by the user, we need to describe a mechanism that detects whether the objects that are currently requested or previously received by the user disclose objects in $\mathcal{O}$ for which the user is not authorized.

2.2 Disclosure Monitor

Disclosure Monitor (DiMon), presented next, is an enhancement of the standard mandatory access control mechanism with a Disclosure Inference Engine to protect against inference channels that result from database constraints (see Fig. 1). Algorithm 1, given in Fig. 2, describes the steps followed by DiMon. Note that the user's id, given to the algorithm as an input value, may refer to a particular user or to a group of users sharing the same security clearance. We illustrate the functionality of DiMon by the following example.

Example 2.1. Again, consider the relation schema *(NAME, RANK, SALARY, EXPERIENCE)* and the relation *Employee* (see Table 1) that satisfies the functional dependency $RANK \rightarrow SALARY$. The security requirement is that only users with *top-secret* security clearances can access the salaries of the employees, i.e., $\lambda(\Pi_{NAME,SALARY}) = \textit{top-secret}$. In the data-dependent mode, the relation *Employee* is used by DiIE.

Assume that a user u with secret clearance has already received the answer to the query

$$\Pi_{SALARY} \sigma_{RANK=Clerk}(Employee),$$

which is $< 34,000 >$. When representing the information revealed to the user, in both data-dependent and data-independent mode, besides the actual data values that were returned to the user, we also represent information contained in the selection conditions of the queries. For example, while the previous query only returned the value for the *SALARY*, the user also knows that this is the salary of the employee who is a clerk. Therefore, the user's history file in the data-dependent mode contains the QA-pair

$$(R[RANK = Clerk, SALARY = 34,000], \Pi_{RANK,SALARY})$$

and, in the data-independent mode, the query

$$\Pi_{RANK,SALARY} \sigma_{RANK=Clerk}.$$

Next, suppose that u requests the query

$$\Pi_{NAME,RANK} \sigma_{EXPERIENCE=10}(Employee).$$

According to our algorithm, DiMon first verifies if there is a direct security violation. Because only tuples over $NAME$ and $RANK$ are classified as secret, there is no security violation detected. Next, DiMon checks for possible indirect security violations. First, a set O' is constructed.

1. In the *data-dependent mode*

$$O' = \{(\Pi_{NAME,SALARY}(Employee), \Pi_{NAME,SALARY})\},$$

thus the set of all atomic objects (QA-pairs) is

Algorithm 1:	Disclosure Monitor
INPUT	1. User's query (object) Q_i 2. User's id u 3. Security classification $< \mathcal{O}, \mathcal{U}, \lambda >$ 4. User's history-file: objects which were previously retrieved by the user (QA-pairs in data-dependent mode and queries in data-independent mode) 5. $\mathcal{D}$, a set of Horn-clause constraints
OUTPUT	Answer to Q_i and update of the user's history-file or refusal of Q_i
METHOD	MAC evaluates direct security violation: **if** there is a classified object $o \in \mathcal{O}$ such that $\lambda(u) \not\geq \lambda(o)$ **then** reject Q_i (note, direct security violation is detected, DiMon functions as the basic MAC mechanism) **else** (no direct security violation was detected) **begin** 1. Find $O' \subseteq \mathcal{O}$, the set of classified objects that cannot be returned to the user; i.e., $O' = \{o \in \mathcal{O} \mid \lambda(u) \not\geq \lambda(o)\}$. 2. Using *Disclosure Inference Engine* (DiIE), determine whether any of the (atomic) objects in $A(O')$ (i.e., disallowed objects) is disclosed from the user's history, the current query, and the constraints $\mathcal{D}$. 3. **if** disclosure is detected in (2) (i.e., security is violated) **then** reject Q_i **else** (security is not violated) answer Q_i and update the user's history-file. **end**

Fig. 2. Disclosure monitor.

$$A(O') = (\{R[NAME = Brunnel, P., SALARY = 34,000], \Pi_{NAME,SALARY}), \ldots, R[NAME = Smith, R., SALARY = 28,000]\}, \Pi_{NAME,SALARY}).$$

Next, the answers to the current and previous queries and the FD are used to determine if any of the atomic objects in $A(O')$ are disclosed. Table 2 summarizes the answers to the first (first tuple) and to the second queries (second tuple). Note, that the δ_is represent unique null values, i.e., values that the user does not know. Using DiIE, it is determined that no atomic object in $A(O')$ is disclosed. Intuitively, this happens in the example because the two answered tuples have different values for attribute *RANK*, and, therefore they cannot be combined by using the FD.

2. In the *data-independent mode*, O' only contains the query $\Pi_{NAME,SALARY}$, thus $A(O') = O'$. Next, the user's previously answered queries, the current query, and the FD are submitted to DiIE. Here, it is determined that the object $\Pi_{NAME,SALARY}$ is disclosed from the queries $\Pi_{SALARY}\sigma_{RANK=Clerk}$, $\Pi_{NAME,RANK}\sigma_{EXPERIENCE=10}$, and the FD. Intuitively, without knowing the actual data, there exists an example (of data) which constitutes a data-dependent disclosure. For example, if the employee *W. Hammer* was a *clerk*, then the answers to the two queries would be as given in Table 3. From these answers, by using the FD, users could infer that Hammer's salary is $\$34,000$.

3 Database Constraints and Data Disclosure

In our model, *database constraints* form the basis of data disclosures. In this paper, we consider Horn-clause constraints which are defined as follows.

Definition 3.1 (Horn-clause constraint). *A* Horn-clause constraint *is an expression of the form*

$$\forall x_1, \ldots, x_m \; (p_1 \wedge \ldots \wedge p_n \rightarrow q),$$

TABLE 1
The Employee Relation to Illustrate Possibler Inference Channel via the Functional Dependency $RANK \rightarrow SALARY$

NAME	RANK	SALARY ($)	EXPERIENCE (years)
Brunnel, P.	Clerk	34,000	5
Evan, S.	Clerk	34,000	3
Hammer, W.	Director	65,000	10
Joels, R.	Clerk	34,000	3
Smith, A.	Accountant	41,000	6
Smith, R.	Secretary	28,000	8

TABLE 2
Query Answers

NAME	RANK	SALARY	EXPERIENCE (years)
δ_1	Clerk	34,000	δ_3
Hammer, W.	Director	δ_4	10

TABLE 3
Example for Data-Independent Disclosure

NAME	RANK	SALARY	EXPERIENCE (years)
δ_1	Clerk	34,000	δ_3
Hammer, W.	Clerk	δ_4	10

where $n \geq 1$ and $x_1, \ldots, x_m$ are all of the free variables in $p_1 \wedge \ldots \wedge p_n \rightarrow q$. Each p_i is of the form

$$R[A_{i_1} = b_1, \ldots, A_{i_k} = b_k],$$

where b_i is either a variable or a constant and each q has one of the following two forms:

1. *$R[A_1 = a_1, \ldots, A_n = a_n]$, where $A_1, \ldots, A_n$ are all of the attributes of R (i.e., the constraint is full) and each a_i is either a constant or a variable that must appear in $p_1 \wedge \ldots \wedge p_n$. In this case, the constraint is called* tuple-generating.
2. *Equality of the form $a_i = a_j$, where each a_i is either a constant or a variable that must appear in $p_1 \wedge \ldots \wedge p_n$. In this case, the constraint is called* equality-generating.

We will refer to $p_1 \wedge \ldots \wedge p_n$ as the body *and to q as the* head *of the constraint. We require that all the variables of q appear in the body. We will use the shorthand $p_1 \wedge \ldots \wedge p_n \rightarrow q$ for Horn-clause constraints. Horn-clause constraints extend generalized dependencies [19] with constants and can express functional, multivalued, and join dependencies, as well as a wide variety of user defined knowledge.*

Definition 3.2 (Truth value of the projection fact). *We say that a projection fact PF of type $A_{i_1}, \ldots A_{i_k}$ is* TRUE *with respect to a set of projection facts $\mathcal{P}$ if $\mathcal{P}$ contains a projection fact PF' with the same predicate symbol and the same values for attributes $A_{i_1}, \ldots, A_{i_k}$. (PF' might contain other attributes as well.)*

Definition 3.3 (Constraint satisfaction). *A database constraint d is* satisfied *by a set $\mathcal{P}$ of projection facts if its formula is TRUE for $\mathcal{P}$. (The truth value is defined in the standard way.)*

Since the relation r over R can be viewed as a set of projection facts of type R, we can also speak of d being satisfied by r. In the following, we restrict ourselves to constraints which can be represented in Horn-clause forms with equalities. We can now formally define the notion of data-dependent disclosure.

Definition 3.4 (Data-dependent disclosure). *Let $\mathcal{D}$ be a set of database constraints, $P_1, \ldots, P_n$ be sets of projection facts over attribute sets $X_1, \ldots, X_n$, and PF be a projection fact over Y. We say that the set of QA-pairs*

$$\mathcal{P} = \{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_n, \Pi_{X_n}\sigma_{C_n})\}$$

discloses $(PF, \Pi_Y\sigma_C)$ *in* data-dependent mode, *denoted as $\mathcal{P} \models_{\mathcal{D}} (PF, \Pi_Y\sigma_C)$ if, for every r over R that satisfies $\mathcal{D}$,*

$$P_i \subseteq \Pi_{X_i}\sigma_{C_i}(r) \quad for \ all \ \ i = 1, \ldots, n$$

implies $PF \in \Pi_Y\sigma_C(r)$.

Given a set S of atomic QA-pairs, we say that $\mathcal{P} \models_{\mathcal{D}} S$ if for every (atomic) QA-pair,

$$(PF, \Pi_Y\sigma_C) \in S, \mathcal{P} \models_{\mathcal{D}} (PF, \Pi_Y\sigma_C).$$

Finally, we denote by

$$(PF, \Pi_Y\sigma_C) \models (PF', \Pi_{Y'}\sigma_{C'})$$

the disclosure $\{(PF, \Pi_Y\sigma_C)\} \models_{\mathcal{D}} (PF', \Pi_{Y'}\sigma_{C'})$, where $\mathcal{D}$ is empty. In this case, we say that $(PF, \Pi_Y\sigma_C)$ data-dependently dominates *or simply* dominates *$(PF', \Pi_{Y'}\sigma_{C'})$.*

Note that, throughout the paper, we use $\models$ to denote both data-dependent disclosure when $\mathcal{D}$ is empty as well as to denote logical entailment. The meaning of $\models$ will be clear from its context. By the Definition 3.4, if $\mathcal{D}$ is inconsistent (i.e., no relation satisfies it), then everything is disclosed. The construction of data-dependent disclosure is computationally expensive and requires large storage overhead. Therefore, we introduce the notion of data-independent disclosure, where only the queries are considered to detect whether an example of data-dependent disclosure can be found that corresponds to the queries.

Definition 3.5 (Data-independent existential disclosure). *Let $\mathcal{D}$ be a set of database constraints and $\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_n}\sigma_{C_n}$ queries over R. We say that the set of queries $\mathcal{P} = \{\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_n}\sigma_{C_n}\}$* data-independently (or existentially) discloses *the query $\Pi_Y\sigma_C$ under $\mathcal{D}$, denoted as $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C$, if there exist*

1. *r over R that satisfies $\mathcal{D}$,*
2. *sets $P_1 \subseteq \Pi_{X_1}\sigma_{C_1}(r), \ldots, P_n \subseteq \Pi_{X_n}\sigma_{C_n}(r)$, and*
3. *$PF \in \Pi_Y\sigma_C(r)$, such that*

$$\{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_n, \Pi_{X_n}\sigma_{C_n})\} \models_{\mathcal{D}} (PF, \Pi_Y\sigma_C).$$

We say, that $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \mathcal{P}'$ if, for every $\Pi_Y\sigma_C$ in $\mathcal{P}'$, $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C$. Further, $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \Pi_Y$ will denote

$$\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_{TRUE},$$

and $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \sigma_C$ will denote $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \Pi_R\sigma_C$, where R is the set of all attributes of R. Also, we say that $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} (PF, \Pi_Y\sigma_C)$ if $\mathcal{P} \rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_{C'}$, where C' is constructed as follows: If $PF = R[A_{i_1} = a_1, \ldots, A_{i_k} = a_k]$, where $A_{i_1}, \ldots, A_{i_k} = Y$, then C' is C conjuncted with all equalities of the form

$$A_{i_j} = a_j, \ (j = 1, \ldots, k).$$

Finally, we denote by $\Pi_{Y'}\sigma_{C'} \rightsquigarrow_{\exists} \Pi_Y\sigma_C$ the case where $\{\Pi_{Y'}\sigma_{C'}\} \rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C$ and $\mathcal{D}$ is empty. In this case, we say that $\Pi_{Y'}\sigma_{C'}$ existentially dominates *$\Pi_Y\sigma_C$.*

To solve the problem of existential disclosure, we also need the notation of universal disclosure, defined as follows.

Definition 3.6 (Universal disclosure). *Let $\mathcal{D}$ be a set of database constraints and $\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_n}\sigma_{C_n}$ queries over R. We say, that the set of queries $\mathcal{P} = \{\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_n}\sigma_{C_n}\}$* universally discloses *the query $\Pi_Y\sigma_C$ under $\mathcal{D}$, denotes as $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} \Pi_Y\sigma_C$ if, for every PF over $\Pi_Y\sigma_C$, there exist 1) r over R that satisfies $\mathcal{D}$, and 2) sets $P_1 \subseteq \Pi_{X_1}\sigma_{C_1}(r), \ldots, P_n \subseteq \Pi_{X_n}\sigma_{C_n}(r)$ such that*

$$\{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_n, \Pi_{X_n}\sigma_{C_n})\} \models_{\mathcal{D}} (PF, \Pi_Y\sigma_C).$$

Similarly to the existential disclosure, we say that $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} \mathcal{P}'$ if for every $\Pi_Y\sigma_C$ in $\mathcal{P}'$, $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} \Pi_Y\sigma_C$. Further, $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} \Pi_Y$ will denote $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} \Pi_Y\sigma_{TRUE}$, $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}\,\sigma_C}$ will denote $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} \Pi_R\sigma_C$, where R is the set of all attributes of R, and $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} (PF, \Pi_Y\sigma_C)$ if $\mathcal{P} \twoheadrightarrow_{\forall\mathcal{D}} \Pi_Y\sigma_{C'}$ and C' is constructed as follows: if $PF = R[A_{i_1} = a_1, \ldots, A_{i_k} = a_k]$, where $A_{i_1}, \ldots, A_{i_k} = Y$, then C' is C conjuncted with all equalities of the form $A_{i_j} = a_j$, $(j = 1, \ldots, k)$. Finally, we denote by $\Pi_{Y'}\sigma_{C'} \twoheadrightarrow_{\forall} \Pi_Y\sigma_C$ the case where

$$\{\Pi_{Y'}\sigma_{C'}\} \twoheadrightarrow_{\forall\mathcal{D}} \Pi_Y\sigma_C$$

and $\mathcal{D}$ is empty. In this case, we say that $\Pi_{Y'}\sigma_{C'}$ universally dominates *$\Pi_Y\sigma_C$.*

Note that, by Definitions 3.5 and 3.6, if $\mathcal{D}$ is inconsistent, then nothing is disclosed. We study the interconnection between the existential and the universal disclosure in Section 5.

4 Data-Dependent Disclosure

In this section, we develop an algorithm to construct a cover of all disclosed query-answer pairs under database constraints $\mathcal{D}$ from the set $\mathcal{P}$ of query-answer pairs. First, we state the decidability result for the data-dependent disclosure.

Theorem 4.1. *The following problem is decidable: Given a set $\mathcal{D}$ of database constraints and a set $\mathcal{P}$ of QA-pairs,*

1. *whether $\mathcal{P} \models_{\mathcal{D}} (PF, \Pi_Y\sigma_C)$ for a given atomic QA-pair $(PF, \Pi_Y\sigma_C)$,*
2. *whether $\mathcal{P} \models_{\mathcal{D}} S$ for a given set S of atomic QA-pairs.*

Theorem 4.1 will be a corollary to Theorem 4.2 that states correctness (i.e., soundness and completeness) of Algorithm 3 that computes a disclosure cover which is defined in this section. To define disclosure cover, first we need the following propositions.

Proposition 4.1. *Let $(PF, \Pi_Y\sigma_C)$ and $(PF', \Pi_{Y'}\sigma_{C'})$ be two QA-pairs, and let C_e, C'_e be the selection conditions generated from C and C', respectively, by extending them with the equalities explicitly given in PF and PF'. Then,*

1. $(PF, \Pi_Y\sigma_C) \models (PF', \Pi_{Y'}\sigma_{C'}) \Longleftrightarrow C_e \models C'_e$
2. *Data-dependent dominance is transitive, that is,*

$$(PF, \Pi_Y\sigma_C) \models (PF', \Pi_{Y'}\sigma_{C'}) \wedge (PF', \Pi_{Y'}\sigma_{C'}) \models (PF'', \Pi_{Y''}\sigma_{C''}) \Rightarrow (PF, \Pi_Y\sigma_C) \models (PF'', \Pi_{Y''}\sigma_{C''}).$$

Note that data-dependent dominance can be extended to QA-pairs that contain sets of projection facts. We say, that $(P, \Pi_Y\sigma_C)$ dominates $(P', \Pi_{Y'}\sigma_{C'})$ if, for every $PF' \in P'$, there exists a $PF \in P$ such that $(PF, \Pi_Y\sigma_C) \models (PF', \Pi_{Y'}\sigma_{C'})$.

Clearly, there may be different QA-pairs Q_1 and Q_2 that are equivalent in the sense that each dominates the other (i.e., $Q_1 \models Q_2$ and $Q_2 \models Q_1$). To provide a uniform representation of equivalent QA-pairs, we define the notion of normal forms of QA-pairs as follows:

Definition 4.1 (Normal form of a QA-pair). *Given a QA-pair $(PF, \Pi_Y\sigma_C)$, its* normal form, *$(PF', \Pi_{Y'}\sigma_{C'})$ is constructed as follows:*

Let C_e be the selection condition constructed from C by extending it with all the equalities explicitly given in PF. Initially, $PF' = PF, Y' = Y$, and $C' = C$.

***Step 1**. For every attribute $A_i \notin Y'$ such that $C_e \models A_i = c$, where c is a constant, do:*

1. *Extend PF' over A_i by adding $A_i = c$ and*
2. *Add A_i to Y'.*

***Step 2**. For every equality $A_i = B \in C'$, where B is either a constant or an attribute such that $A_i \in Y'$ remove $A_i = B$ from C' (i.e., C' contains only equalities not given explicitly in PF').*

Proposition 4.2. *Data-dependent disclosure is transitive, that is, for any sets $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ of QA-pairs $\mathcal{P}_1 \models_{\mathcal{D}} \mathcal{P}_2 \wedge \mathcal{P}_2 \models_{\mathcal{D}} \mathcal{P}_3 \Rightarrow \mathcal{P}_1 \models_{\mathcal{D}} \mathcal{P}_3$*

Definition 4.2 (Data-dependent disclosure cover of $\mathcal{P}$ under $\mathcal{D}$ ($DDC_{\mathcal{D}}(\mathcal{P})$)). *Given a set of dependencies $\mathcal{D}$ and a set of QA-pairs $\mathcal{P} = \{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_n, \Pi_{X_n}\sigma_{C_n})\}$* data-dependent disclosure cover *under $\mathcal{D}$, denoted as $DDC_{\mathcal{D}}(\mathcal{P})$, is defined as a set $\mathcal{S}$ of QA-pairs $(P, \Pi_Y\sigma_C)$ that is*

1. sound, *i.e., for every $(P, \Pi_Y\sigma_C) \in \mathcal{S}$ and $(P', \Pi_{Y'}\sigma_{C'})$,*

$$(P, \Pi_Y\sigma_C) \models (P', \Pi_{Y'}\sigma_{C'}) \Rightarrow \mathcal{P} \models_{\mathcal{D}} (P', \Pi_{Y'}\sigma_{C'})$$

2. complete, *i.e., for every $(PF', \Pi_{Y'}\sigma_{C'})$, $\mathcal{P} \models_{\mathcal{D}} (PF', \Pi_{Y'}\sigma_{C'})$ implies that there exists $(P, \Pi_Y\sigma_C)$ in $\mathcal{S}$ such that $(P, \Pi_Y\sigma_C) \models (PF', \Pi_{Y'}\sigma_{C'})$*
3. compact, *for every $(P, \Pi_Y\sigma_C) \in \mathcal{S}$,*
   a. *It is in normal form,*
   b. *C is nonredundant, that is, no strict subset of C is equivalent to C,*
   c. *There does not exist $(P', \Pi_{Y'}\sigma_{C'})$ in $\mathcal{S}$ that is different from $(P, \Pi_Y\sigma_C)$ and that dominates $(P, \Pi_Y\sigma_C)$.*

Before we can describe the algorithm that is based on the *chase* process [19], we need to first define the notions of atom mapping and application of dependencies.

Definition 4.3 (Atom mapping of dependencies). *Given a Horn-clause constraint $B_1, \ldots, B_n \rightarrow H$ and a relation r, we define an* atom mapping *as a function $h : \{B_1, \ldots, B_n\} \rightarrow r$ such that*

Algorithm 2:	Chase process
INPUT	1. Set of Horn-clause constraints $\mathcal{D}$ 2. Relation r, which may contain null-values
OUTPUT	Updated relation r.
METHOD	**begin** Apply dependencies in $\mathcal{D}$ on r until no more changes to r occur. **end**

Fig. 3. Chase process.

1. *h preserves constants, i.e., if $h(R[\ldots, A_i = c, \ldots]) = (c_1, \ldots, c_i, \ldots, c_n) \in r$ and c is a constant, then $c = c_i$.*
2. *h preserves equalities, i.e., if $B_i = R[\ldots, A_k = a, \ldots]$, $B_j = R[\ldots, A_l = a, \ldots]$ and*

$$h(B_i) = (c_1, \ldots, c_k, \ldots, c_n),$$
$$h(B_j) = (c'_1, \ldots, c'_l, \ldots, c'_n),$$

then $c_k = c'_l$.

Clearly, h can be extended to work from the set S_d of the symbols (i.e., variables and constants) of the constraint to the symbols of r. We will use the symbol h for this extended mapping as well.

Definition 4.4 (Application of dependencies). *A dependency d is applied on a relation r by using an atom mapping h as follows:*

1. *If d is an* equality-generating dependency *of the form $B_1, \ldots, B_n \rightarrow a = b$, then* equate *$h(a)$ and $h(b)$ as follows:*
   a. *If both $h(a)$ and $h(b)$ are null-values, then replace all occurrences of one of them in r with the other.*
   b. *If one of them, say $h(a)$, is not a null-value, then replace all occurrence of $h(b)$ in r with $h(a)$.*
   c. *If both are not null-values (i.e., constants), do nothing. If $h(a) \neq h(b)$, we say that* inconsistency *occurred.*
2. *If d is* a tuple-generating dependency *of the form $B_1, \ldots, B_n \rightarrow R[A_1 = a_1, \ldots, A_n = a_n]$ and the tuple $(h(a_1), \ldots, h(a_n))$ is not in r, then add it to r.*

Now that the application of the dependencies is defined, we can describe the chase process, which is a variation of the *Chase* defined in [19]. Algorithm 2, for the chase process, is given in Fig. 3.

In order to preserve the information encapsulated in the selection conditions of the queries, we generate dependencies from them. These dependencies are then applied to the answers of the corresponding queries. Let C be a conjunction of conditions of the forms $A_i = A_j$ or $A_i = c$, where A_i and A_j are attribute names and c is a constant. We *generate dependencies* from each condition in C as follows: 1) For each condition $A_i = A_j$, we create a dependency

$$R[A_i = a_i, A_j = a_j] \rightarrow a_i = a_j$$

and 2) for each condition $A_i = c$, we create a dependency $R[A_i = a_i] \rightarrow a_i = c$. Fig. 4 shows Algorithm 3 that generates the data-dependent disclosure cover of a set $\mathcal{P}$ of query-answer pairs under dependencies $\mathcal{D}$.

Theorem 4.2 (data-dependent disclosure). *Algorithm 3 effectively computes the data-dependent disclosure cover $DDC_{\mathcal{D}}(\mathcal{P})$, i.e., the algorithm terminates and the computed $\mathcal{S}$ is*

1. sound,
2. complete, *and*
3. compact.

Proof. First, Algorithm 3 must terminate since the applications of dependencies do not introduce new constants or symbols and, therefore, there is only a finite number of tuples that can be generated. We need to show that the set $\mathcal{S}$ computed by the algorithm is

1. sound,
2. complete, and
3. compact.

Proof of soundness. Let $\mathcal{S}$ be the set produced by Algorithm 3. Because data-dependent disclosure is transitive (Proposition 4.2), to prove *soundness* of $\mathcal{S}$, it is sufficient to prove that every QA-pair $(P, \Pi_Y \sigma_C)$ in $\mathcal{S}$ is disclosed, i.e., $\mathcal{P} \models_{\mathcal{D}} (P, \Pi_Y \sigma_C)$. This is based on the following claim:

Claim 1. *For every relation r that satisfies $\mathcal{D}$, there exist symbol mappings (preserving equalities and constants) from the tuples of the relations r_i, r'_i, and r'', which were generated by the algorithm in Step 1a, 1c, and 2, respectively, to the tuples of r.*

Before proving Claim 1, we complete the proof of soundness. If *inconsistency* occurred in Step 2, then, by Claim 1, there does not exist r that satisfies $\mathcal{D}$ such that $P_i \subseteq \Pi_{X_i} \sigma_{C_i}(r)$ for all $i = 1, \ldots, m$. Therefore,

$$\mathcal{S} = \{(ALL, \Pi_R \sigma_{TRUE})\}$$

is vacuously disclosed. If *no inconsistency* occurred, then, by using Claim 1, every $(PF, \Pi_Y \sigma_C)$ generated in Step 3 of the algorithm is indeed disclosed. Finally, Step 4 of Algorithm 3 clearly leaves the set $\mathcal{S}$ sound. The only remaining part is the *proof of Claim 1*, for which we will use the following claim:

Claim 2. *Let r_1 be a relation that possibly contains null values, and r_2 be a regular relation such that r_1 and r_2 are over the same schema and r_2 satisfies a set of dependencies DD. If there exists a symbol mapping (preserving equalities and constants) from r_1 to r_2, then there also exists a symbol mapping from $Chase_{DD}(r_1)$ to r_2, where $Chase_{DD}(r_1)$ is the relation obtained by applying the dependencies DD on r_1.*

Algorithm 3:	Data-Dependent Disclosure Cover
INPUT	1. Set $\mathcal{P} = \{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_m, \Pi_{X_m}\sigma_{C_m})\}$ of QA-pairs 2. Set of Horn-clause constraints $\mathcal{D}$
OUTPUT	Data-dependent disclosure cover $DDC_{\mathcal{D}}(\mathcal{P})$ (The set $\mathcal{S}$ at the conclusion of the algorithm.)
METHOD	1. **for** every $(P_i, \Pi_{X_i}\sigma_{C_i})$ in $\mathcal{P}$ **do** (a) *Augment* each $PF \in P_i$ with new unique null values for attributes in $R - X_i$ to create r_i (b) *Generate dependencies* D_i from C_i (c) *Chase* r_i with D_i to construct r'_i 2. Construct $r' = \bigcup_{i=1}^{m} r'_i$ and *chase* r' with dependencies $\mathcal{D}$ to get r'' 3. **if** during Step 1 or 2 *inconsistency* occurred, **then** return $\mathcal{S} = \{(ALL, \Pi_R\sigma_{TRUE})\}$ as output, where ALL is the domain of R (i.e., everything is disclosed) **else** let $\mathcal{S}$ to be the set of all $(PF, \Pi_Y\sigma_C)$ such that there exists $t \in r''$ and (a) Y contains all attributes in which t is not null (b) $PF = \Pi_Y(t)$ (c) C encodes only the equalities among the attributes which do not appear explicitly in PF. 4. (a) Continue until no more changes occur: **if** two different QA-pairs $(PF, \Pi_Y\sigma_C)$ and $(PF', \Pi_{Y'}\sigma_{C'})$ are in $\mathcal{S}$, and $(PF, \Pi_Y\sigma_C) \models (PF', \Pi_{Y'}\sigma_{C'})$ then remove $(PF', \Pi_{Y'}\sigma_{C'})$ from $\mathcal{S}$. (b) Continue until no more changes occur: **if** two different QA-pairs $(P, \Pi_Y\sigma_C)$ and $(P', \Pi_Y\sigma_{C'})$ are in $\mathcal{S}$ such that $C \equiv C'$ do: i. Add $(P \cup P', \Pi_Y\sigma_C)$ to $\mathcal{S}$, and ii. Remove $(P, \Pi_Y\sigma_C)$ and $(P', \Pi_Y\sigma_{C'})$ from $\mathcal{S}$. (c) **for** each $(P, \Pi_Y\sigma_C)$ in $\mathcal{S}$ replace C by an equivalent minimal subset of C. 5. Return $\mathcal{S}$ as the output.

Fig. 4. Data-dependent disclosure cover.

Before proving Claim 2, we complete the proof of Claim 1. Clearly, there is a mapping from r_i to $\Pi_{X_i}\sigma_{C_i}(r)$ and $\Pi_{X_i}\sigma_{C_i}(r)$ satisfies D_i. From this and Claim 2, it results that there is a symbol mapping from $r'_i = Chase_{D_i}(r_i)$ to $\Pi_{X_i}\sigma_{C_i}(r)$ and therefore to r. But then, there are symbol mappings $p_1, \ldots, p_m$ from every r'_i $i = 1, \ldots, m$ to r. Since two different relations r'_i and r'_j cannot share a common null value because of the way r_is are constructed in Step 1a and the fact that $Chase_{D_i}$ only operates on single relations, the union of the symbol mappings $p_1 \cup \ldots \cup p_m$ constitutes a symbol mapping from $r' = \bigcup_{i=1}^{m} r'_i$ to r. But then, by Claim 2, there exist a symbol mapping from $r'' = Chase_{\mathcal{D}}(r')$ to r.

The only remaining part is *Claim 2*, which we prove by induction on the number of application of dependencies on r_1. We know that, originally, there is a symbol mapping from r_1 to r_2 and that r_2 satisfies the dependencies DD. Assume, that dependencies $d_1, \ldots, d_k$ of DD were applied on r_1 and that, by induction hypothesis, there exists a symbol mapping p^k from $r_1^k = Chase_{d_1,\ldots,d_k}(r_1)$ to r_2. We want to show that, after the application of the next dependency d_{k+1} on r_1^k, there still exists a symbol mapping p^{k+1} from r_1^{k+1} to r_2.

Let tuples $t_{1_1}, \ldots, t_{1_l}$ in r_1^k satisfy the body of d_{k+1}, i.e., there exists an atom mapping h from the body of d_{k+1} to the tuples $t_{1_1}, \ldots, t_{1_l}$, and let tuples $t_{2_1}, \ldots, t_{2_l}$ be their corresponding tuples in r_2, i.e.,

$$p^k(t_{1_i}) = t_{2_i} \text{ for all } i = 1, \ldots, l.$$

If d_{k+1} is a(n):

1. **Equality-generating dependency** of the form $B_1, \ldots, B_n \rightarrow a = b$, then $h(a)$ and $h(b)$ are the two attribute values of r_1^k which are equated, say all occurrences of $h(a)$ are replaced by $h(b)$. We define p^{k+1} as follows: If tuple t in r_1^{k+1} was created from tuple t' in r_1^k, then $p^{k+1}(t) = p^k(t')$. (Note this includes the case when $t = t'$.)

   We claim, that p^{k+1} is indeed a symbol mapping, i.e., it preserves equalities and constants. For this, let $p^k(h(a)) = c_1$ and $p^k(h(b)) = c_2$, where c_1 and c_2 are constant values of r_2. Since r_2 satisfies d_{k+1} and p^k preserves equalities and constants, c_1 must be equal to c_2 and, therefore, p^{k+1} is a symbol mapping from r_1^{k+1} to r_2.

2. **Tuple-generating dependency** of the form

$$B_1, \ldots, B_n \rightarrow R[A_1 = a_1, \ldots, A_n = a_n]$$

and a new tuple $(h(a_1), \ldots, h(a_n))$ is added to r_1^k. We define p^{k+1} as follows:

a. For the tuples in r_1^k, p^{k+1} coincides with p^k.
b. For the new tuple,

$$p^{k+1}((h(a_1), \ldots, h(a_n))) = (p^k(h(a)), \ldots, p^k(h(a_n))) = t.$$

We claim, that t is indeed in r_2. This follows from the fact, that r_2 satisfies d_{k+1} and that p^k preserves equalities and constants. This completes the proof of the soundness of the algorithm.

Proof of completeness. If *inconsistency* occurred, then completeness trivially follows since $(ALL, \Pi_R \sigma_{TRUE})$ dominates every QA-pair.

If *no inconsistency* occurred, then let us suppose that a QA-pair $(PF, \Pi_Y \sigma_C)$ is disclosed from

$$\mathcal{P} = \{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_m, \Pi_{X_m}\sigma_{C_m})\}$$

by $\mathcal{D}$. We construct a relation r from r'' at the end of Step 2 by replacing each unique null value δ_i with a new unique constant while preserving equalities. It is clear that r satisfies $\mathcal{D}$ and

$$P_i \subseteq \Pi_{X_i}\sigma_{C_i}(r) \text{ for all } i = 1, \ldots, m.$$

But then, by the definition of data-dependent disclosure, $PF \in \Pi_Y \sigma_C(r)$, meaning that in Step 3, the algorithm should have produced a QA-pair q which dominates $(PF, \Pi_Y \sigma_C)$. Since by construction of $\mathcal{S}$ in Step 4 of the algorithm a created QA-pair $q' \in \mathcal{S}$ must dominate q, q' must therefore also dominate $(PF, \Pi_Y \sigma_C)$. This completes the proof of completeness.

Proof of compactness. It is given by contradiction. Assume that $\mathcal{S}$ is not compact. Clearly, by construction of $(PF, \Pi_Y \sigma_C)$ in Steps 3a, 3b, and 3c, it is in normal form. Also, by Step 4c, C is nonredundant. But then, there is a QA-pair $(P, \Pi_Y \sigma_C) \in \mathcal{S}$ for which there exists a $(P', \Pi_{Y'} \sigma_{C'})$ in $\mathcal{S}$ such that $(P', \Pi_{Y'} \sigma_{C'}) \neq (P, \Pi_Y \sigma_C)$ and $(P', \Pi_{Y'} \sigma_{C'}) \models (P, \Pi_Y \sigma_C)$.

By construction in Step 4b, P, P' are nonempty and therefore, there are $PF \in P$ and $PF' \in P'$ such that $PF = \Pi_Y(PF')$. But then, we must have had QA-pairs $(PF', \Pi_{Y'} \sigma_{C'_e})$ and $(PF, \Pi_Y \sigma_{C_e})$, where $C'_e \equiv C'$ and $C_e \equiv C$ after Step 4a of the algorithm such that $(PF', \Pi_{Y'} \sigma_{C'_e}) \models (PF, \Pi_Y \sigma_{C_e})$. This is a contradiction since at least one of them should have been eliminated in Step 4a. This completes the proof that the set $\mathcal{S}$ at the end of the algorithm is compact. □

The complexity of Algorithm 3 is dominated by the *chase* process in Step 2. In it, an iteration (application of a dependency) is bounded by $O(I^T)$, where I is the number of symbols in r' and T is the total number of attributes of r'. Each iteration, if done naively, may check $O(I^{nT})$ of potential atom mappings, where n is the number of formulas in the body of the dependency, spending $O(nT)$ time for a check. Thus, the work in each iteration is bounded by $O(nT \cdot I^{nT})$. Therefore, the overall time of *chase* is bounded by $O(nT \cdot I^{nT} \cdot I^T) = O(nT \cdot I^{(n+1)T})$. Since the number of symbols in r' is bounded by qT, where q is the overall number of atomic QA-pairs in $\mathcal{P}$, the complexity of Algorithm 3 is bounded by $O(nT \cdot (qT)^{(n+1)T})$, i.e., polynomial in the size of the input, if the number of attributes is bounded by a constant, and exponential in the number of attributes.

Example 4.1. This is a detailed illustration of generating the data-dependent disclosure cover for the two queries presented in Example 2.1. The input for Algorithm 3 is the QA-pairs

$$\mathcal{P} = \{(R[RANK = Clerk, SALARY = 34,000], \Pi_{RANK,SALARY}) (R[NAME = Hammer, W., RANK = Director], \Pi_{NAME,RANK}\sigma_{EXPERIENCE=10})\}$$

and the Horn-clause constraint

$$R[RANK = r, SALARY = s_1] \wedge R[RANK = r, SALARY = s_2] \rightarrow s_1 = s_2$$

that represents the functional dependency

$$RANK \rightarrow SALARY.$$

Table 2 shows the relation r' (Step 2 of Algorithm 3) that is chased by the FD. Since there does not exist any atom mapping from the FD to the tuples of r', the FD cannot be applied (see Definition 4.4) and, therefore, r' is also the output relation (r'') of chase process.

In Step 3, r'' is used to construct $\mathcal{S}$ that is data-dependent disclosure cover of $\mathcal{P}$ under FD.

$$\mathcal{S} = \{(R[RANK = Clerk, SALARY = 34,000], \Pi_{RANK,SALARY}), (R[NAME = Hammer, W., RANK = Director, EXPERIENCE = 10], \Pi_{NAME,RANK,EXPERIENCE})\}.$$

Since $\mathcal{S}$ does not disclose any top-secret object, the query is answered to the user.

Example 4.2. Now, consider the previous example with the difference that the employee Hammer, W is a clerk, that is, the third tuple in the *Employee* relation is $< \text{Hammer, W., Clerk}, 34,000, 10 >$. Then, the input for Algorithm 3 is the QA-pairs

$$\mathcal{P} = \{(R[RANK = Clerk, SALARY = 34,000], \Pi_{RANK,SALARY}), (R[NAME = Hammer, W., RANK = Clerk], \Pi_{NAME,RANK}\sigma_{EXPERIENCE=10})\}.$$

In this case, there exists an atom mapping h from the FD to the tuples generated from $\mathcal{P}$ (see Table 3), that is, $h(r) = Clerk$, $h(s_1) = 34,000$, and $h(s_2) = \delta_4$. The

result of the application of the functional dependency with mapping h is that δ_4 is replaced with $34,000$. Since the FD cannot be applied again, the final output of Algorithm 3 is

$$\mathcal{S} = \{(R[NAME = Hammer, W., RANK = Clerk, SALARY = 34,000, EXPERIENCE = 10], \Pi_{NAME,RANK,SALARY,EXPERIENCE})\}.$$

Since $\mathcal{S}$ discloses a top-secret object, the second query is rejected.

5 Data-Independent Disclosure

We start the section with the decidability result of data-independent disclosure.

Theroem 5.1. *If queries and constraints* do not involve constants, *then the following problem is decidable: Given a set of queries $\mathcal{P} = \{\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_n}\sigma_{C_n}\}$, a set of Horn-clause constraints $\mathcal{D}$, and a query $\Pi_Y\sigma_C$ determine whether $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C$.*

The proof of Theorem 5.1 is based on the correctness of the algorithm that computes existential disclosure cover that we present in this section. As in the data-dependent case, we need the notion of normal forms.

Definition 5.1 (Normal form of $\Pi_Y\sigma_C$). *Given a query $\Pi_Y\sigma_C$, its* normal form *is $\Pi_{Y'}\sigma_C$, where Y' is the minimal set of attributes that is closed under the following properties:*

1. $Y \subseteq Y'$,
2. *If $C \models A_i = A_j$ and $A_j \in Y$, then $A_i \in Y'$, and*
3. *If $C \models A_i = c$, where c is a constant, then $A_i \in Y'$.*

If $Y' = Y$, we say that $\Pi_Y\sigma_C$ is in normal form.

The following proposition establishes syntactic conditions (that can be effectively tested) for universal and existential dominance which are used in this section.

Proposition 5.1. *Let $\Pi_Y\sigma_C$ and $\Pi_{Y'}\sigma_{C'}$ be two queries in normal forms. Then,*

1. $\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y'}\sigma_{C'} \iff$
   a. $Y' \subseteq Y$,
   b. $C' \wedge C$ *is consistent, and*
   c. $C \models C'_{\neg Y}$, *where $C'_{\neg Y}$ is the conjunction of equalities $A_i = A_j$ such that $C' \models A_i = A_j$ and $A_i \notin Y$ or $A_j \notin Y$.*
2. $\Pi_Y\sigma_C \not\rightsquigarrow_{\forall} \Pi_{Y'}\sigma_{C'} \iff$
   a. $\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y'}\sigma_{C'}$ *and*
   b. $C' \models (C \wedge C')_{Y'}$, *where $(C \wedge C')_{Y'}$ is the conjunction of equalities*
      i. $A_i = c$ *such that $C \wedge C' \models A_i = c$, where c is a constant and $A_i \in Y'$, or*
      ii. $A_i = A_j$ *such that $C \wedge C' \models A_i = A_j$, where A_i and A_j are in Y'.*

Next, we establish the relationship between the notions of universal and existential dominance.

Proposition 5.2. *For every $\Pi_Y\sigma_C$, $\mathcal{P}$, and $\mathcal{D}$:*

1. *If C is consistent, then*
$$\mathcal{P} \not\rightsquigarrow_{\forall\mathcal{D}} \Pi_Y\sigma_C \Rightarrow \mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C.$$
2. *If $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C$, then there exists a consistent set C' such that $(C' \models C)$, $\mathcal{P} \not\rightsquigarrow_{\forall\mathcal{D}} \Pi_Y\sigma_{C'}$ and*
$$\Pi_Y\sigma_{C'} \not\rightsquigarrow_{\exists} \Pi_Y\sigma_C.$$

The following two propositions deal with the transitivity of universal and existential disclosures.

Proposition 5.3. *Universal disclosure is transitive. Let $\mathcal{P}_1, \mathcal{P}_2$, and $\mathcal{P}_3$ be sets of queries:*

$$\mathcal{P}_1 \not\rightsquigarrow_{\forall\mathcal{D}} \mathcal{P}_2 \ \wedge\ \mathcal{P}_2 \not\rightsquigarrow_{\forall\mathcal{D}} \mathcal{P}_3 \Rightarrow \mathcal{P}_1 \not\rightsquigarrow_{\forall\mathcal{D}} \mathcal{P}_3.$$

In particular, universal dominance is transitive, i.e.,

$$\Pi_{Y_1}\sigma_{C_1} \not\rightsquigarrow_{\forall} \Pi_{Y_2}\sigma_{C_2} \ \wedge\ \Pi_{Y_2}\sigma_{C_2} \not\rightsquigarrow_{\forall} \Pi_{Y_3}\sigma_{C_3} \Rightarrow \Pi_{Y_1}\sigma_{C_1} \not\rightsquigarrow_{\forall} \Pi_{Y_3}\sigma_{C_3}.$$

Proposition 5.4. *Existential disclosure ($\not\rightsquigarrow_{\exists}$) is not transitive.*

Similarly to the data-dependent case, we need the notion of compact cover.

Definition 5.2 (Data-independent disclosure cover of $\mathcal{P}$ under $\mathcal{D}$). *Given $\mathcal{P} = \{\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_n}\sigma_{C_n}\}$ a set of queries, the* data-independent disclosure cover *of $\mathcal{P}$ under $\mathcal{D}$, denoted as $IDC_{\exists\mathcal{D}}(\mathcal{P})$, is a set $\mathcal{S}$ of queries $\Pi_Y\sigma_C$ that is*

1. sound, *i.e., for every $\Pi_Y\sigma_C$ in $\mathcal{S}$ and query $\Pi_{Y'}\sigma_{C'}$, $\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y'}\sigma_{C'} \Rightarrow \mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$.*
2. complete, *i.e., for every query $\Pi_{Y'}\sigma_{C'}$, $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$ implies that there exists $\Pi_Y\sigma_C$ in $\mathcal{S}$ such that $\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y'}\sigma_{C'}$*
3. compact, *i.e., for every query $\Pi_Y\sigma_C \in \mathcal{S}$*
   a. *C is nonredundant, that is, no strict subset of C is equivalent to C and*
   b. *there does not exist $\Pi_{Y'}\sigma_{C'}$ in $\mathcal{S}$ that is different from $\Pi_Y\sigma_C$ and universally dominates $\Pi_Y\sigma_C$.*

Next, we present an algorithm that generates data-independent disclosure cover when neither the selection conditions of the queries nor the database dependencies contain constants. The algorithm is shown in Fig. 5.

Theorem 5.2 (Constant-free Data-Independent Disclosure Cover). *Algorithm 4 effectively computes $IDC_{\exists\mathcal{D}}(\mathcal{P})$, i.e., the algorithm terminates and the computed $\mathcal{S}$ is*

1. sound,
2. complete, *and*
3. compact.

Proof. Clearly, since the application of dependencies do not generate new symbols, there is only a finite number of distinct tuples that can be generated; therefore, the algorithm must *terminate*. We need to show that the set $\mathcal{S}$ computed by the algorithm is

Algorithm 4:	Constant-free Data-Independent Disclosure Cover
INPUT	1. $\mathcal{P} = \{\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_m}\sigma_{C_m}\}$ (C_is are conjunctions of equalities between attributes) 2. $\mathcal{D}$ (set of Horn-clause constraints without constants)
OUTPUT	$\mathcal{S}$ (existential disclosure cover of $\mathcal{P}$ under $\mathcal{D}$)
METHOD	1. (*Initialization*) **for** every $\Pi_{X_i}\sigma_{C_i} \in \mathcal{P}$ *generate* a tuple t_i as follows: (a) **for** each attribute $A_j \in X_i$ assign the variable x. (b) **if** C_i contains equality among two attributes A_i and A_j such that at least one of them is in X_i then assign variable x to both A_i and A_j. (c) Assign unique null values to attributes that were not considered previously, while preserving equalities in C_i; i.e., if $A_i = A_j \in C_i$ and $A_i, A_j \notin X_i$ then assign the same null value to both A_i and A_j. 2. *Construct* r to be $\{t_1, \ldots, t_m\}$ 3. *Chase* r with $\mathcal{D}$ to generate r' 4. Compute $\mathcal{S}$ as follows: **for** each tuple $t \in r'$ generate a query $\Pi_Y\sigma_C$, where (a) Y contains all the attributes A_i where $t[A_i] = x$ (b) C has all the equalities among null values. 5. (a) Continue until no more changes occur: **for** every two different queries $\Pi_Y\sigma_C$ and $\Pi_{Y'}\sigma_{C'}$ in $\mathcal{S}$, if $\Pi_Y\sigma_C \rightsquigarrow_\forall \Pi_{Y'}\sigma_{C'}$ then remove $\Pi_{Y'}\sigma_{C'}$ from $\mathcal{S}$. (b) **for** every query $\Pi_Y\sigma_C$ in $\mathcal{S}$, replace C by an equivalent minimal subset of C. 6. Return $\mathcal{S}$ as output.

Fig. 5. Constant-free data-independent disclosure cover.

1. sound,
2. complete, and
3. compact.

Proof of soundness. It is sufficient to show that $\mathcal{S}$ constructed in Step 4 (i.e., before Step 5) of Algorithm 4 is *sound*. This follows from the following two claims:

Claim 5. *Every $\Pi_Y\sigma_C \in \mathcal{S}$ is existentially disclosed from $\mathcal{P}$ under $\mathcal{D}$ and*

Claim 6. *If $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C$ and $\Pi_Y\sigma_C \not\rightsquigarrow_\exists \Pi_{Y'}\sigma_{C'}$, then $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$.*

To prove Claim 5, we construct an example relation r_{ex} that satisfies $\mathcal{D}$, sets $P_i \subseteq \Pi_{X_i}\sigma_{C_i}(r_{ex})$, $(i = 1, \ldots, m)$, $PF \in \Pi_Y\sigma_C(r_{ex})$, and show that

$$(P_m, \Pi_{X_m}\sigma_{C_m})\} \models_\mathcal{D} (PF, \Pi_Y\sigma_C).$$

Let r_{ex} be a relation over R that contains only one tuple t_{ex} such that, for each attribute, $A_i \in R$, $t_{ex}[A_i] = c$, where c is a constant. Clearly, r_{ex} satisfies $\mathcal{D}$. Let $P_i = \Pi_{X_i}\sigma_{C_i}(r_{ex})$ for $i = 1, \ldots, m$. Note that, since C_i contains only equalities among attributes, t_{ex} satisfies all of the queries and the answer to the query $\Pi_{X_i}\sigma_{C_i}$ is the projection of t_{ex} on attributes X_i. Also, $PF = \Pi_Y\sigma_C(r_{ex})$.

We use Algorithm 3 (Data-dependent disclosure cover algorithm) to show

$$\{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_m, \Pi_{X_m}\sigma_{C_m})\} \models_\mathcal{D} (PF, \Pi_Y\sigma_C).$$

Let $\bar{r}$ be the relation generated from $P_1, \ldots, P_m$ at the beginning of Step 2 on Algorithm 3. Note that, at this stage, each P_i, $i = 1, \ldots, m$ has already been *chased* by the dependencies which were generated by the selection conditions of the queries. This application of dependencies is equivalent to the initialization Steps 1a and 1b of Algorithm 4.

We want to show that if a sequence of dependency applications by Algorithm 4 created a tuple t that was used to generate $\Pi_Y\sigma_C$ in $\mathcal{S}$, then the same sequence of dependency applications will generate a tuple $\bar{t} \in \bar{r}$ by Algorithm 3 such that PF over $\Pi_Y\sigma_C$ can be generated from $\bar{t}$. From this and Theorem 4.2, it follows that every $\Pi_Y\sigma_C \in \mathcal{S}$ is existentially disclosed from $\mathcal{P}$ under $\mathcal{D}$.

To show that the application of dependencies on $\bar{r}$ will indeed generate the tuple $\bar{t}$, notice that $\bar{r}$ and r, the input relation in Step 2 of Algorithm 4, are *isomorphic* with the only difference that, in places where r has a variable, $\bar{r}$ has the constant c. But then, the *chase* process executes exactly the same way on both r and $\bar{r}$.

To prove Claim 6, note that, because $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_Y\sigma_C$, there exists r over R that satisfies $\mathcal{D}$, $P_i \subseteq \Pi_{Y_i}\sigma_{C_i}(r)$ for all $i = 1, \ldots, n$, and $PF \in \Pi_Y\sigma_C(r)$ such that

$$\{(P_1, \Pi_{Y_1}\sigma_{C_1}), \ldots, (P_n, \Pi_{Y_n}\sigma_{C_n})\} \models_\mathcal{D} (PF, \Pi_Y\sigma_C).$$

Also, from $\Pi_Y\sigma_C \not\rightsquigarrow_\exists \Pi_{Y'}\sigma_{C'}$ and Proposition 5.1, we know that $Y' \subseteq Y$, $C' \wedge C$ is consistent and $C \models C'_{\neg Y}$. To show that $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$, we construct an example projection fact PF' over $\Pi_{Y'}\sigma_{C'}$ such that we can prove that there exists a relation r' that satisfies $\mathcal{D}$, sets of projection facts P_i, $i = 1, \ldots, n$, such that $P_i \subseteq \Pi_{Y_i}\sigma_{C_i}(r')$, $PF' \in \Pi_{Y'}\sigma_{C'}(r')$ such that

$\{(P_1, \Pi_{Y_1}\sigma_{C_1}), \ldots, (P_n, \Pi_{Y_n}\sigma_{C_n})\} \models_{\mathcal{D}} (PF', \Pi_{Y'}\sigma_{C'})$.

For this, construct a tuple t that satisfies $C \wedge C'$ out of PF. This tuple exists since $C \wedge C'$ is consistent. Then, let PF' be $\Pi_{Y'}(t)$. By construction, and the fact that $Y' \subseteq Y$, we have $PF' = \Pi_{Y'}(PF)$. From

$$\{(P_1, \Pi_{Y_1}\sigma_{C_1}), \ldots, (P_n, \Pi_{Y_n}\sigma_{C_n})\} \models_{\mathcal{D}} (PF, \Pi_Y\sigma_C)P_i \subseteq \Pi_{Y_i}\sigma_{C_i}(r).$$

for all $i = 1, \ldots, n$ implies $PF \in \Pi_Y\sigma_C(r)$. Then, there must exists a tuple t' in r' such that $PF = \Pi_Y(t')$ and, because $PF' = \Pi_{Y'}(PF)$, we have $PF' = \Pi_{Y'}(t')$. Also, because t' satisfies C and $C \models C'_{\neg Y}$, t' also satisfies C'. But then, we have $PF' = \Pi_{Y'}\sigma_{C'}(t')$ and, therefore, $PF' \in \Pi_{Y'}\sigma_{C'}(r')$. But, this is exactly the requirement for existential disclosure, thus $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$. This completes the proof of Claim 6. This completes the proof of soundness.

Proof of completeness. First we show that, for $\mathcal{S}$ generated in Step 4 (i.e., before Step 5) in Algorithm 4 and for every query $\Pi_{Y'}\sigma_{C'}$, $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$ implies that there exists $\Pi_Y\sigma_C \in \mathcal{S}$ such that $\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y'}\sigma_{C'}$, that is, Step 4 of Algorithm 4 generates a tuple over Y and equality among null values such that $C \models A_i = A_j \in C'$, where A_i or $A_j \notin Y$.

Since $\mathcal{P} \not\rightsquigarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$, there must exist

1. a relation r that satisfies $\mathcal{D}$,
2. $P_i \subseteq \Pi_{X_i}\sigma_{C_i}(r)$, $i = 1, \ldots, m$, and
3. $PF \in \Pi_{Y'}\sigma_{C'}(r)$ such that

$$\{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_1, \Pi_{X_m}\sigma_{C_m})\} \models_{\mathcal{D}} (PF, \Pi_{Y'|prime}\sigma_{C'}).$$

But then, by Theorem 4.2, Algorithm 3 will generate a projection fact that contains PF if $P_1, \ldots, P_m$ are given as input.

Let $\bar{r}$ be the relation generated from $P_1, \ldots, P_m$ at the beginning of Step 2 of Algorithm 3, i.e., each P_i is already *chased* by dependencies generated from the C_is. Let $D_1, \ldots, D_l$ the sequence of dependency applications that generated the tuple $\bar{t}$ which was used to extract $(PF, \Pi_{Y'}\sigma_{C'})$. We want to show that 1) $D_1, \ldots, D_l$ can be applied on r, the relation generated in Step 2 of Algorithm 4, and 2) the application of $D_1, \ldots, D_l$ by Algorithm 4 will generate a tuple that corresponds to query $\Pi_Y\sigma_C$, where $\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y'}\sigma_{C'}$. For this, we use the notion of generally corresponding tuples. We say, that the tuple $t \in r$ generally corresponds to tuples $\bar{t_1}, \ldots, \bar{t_k}$ if there exist mappings $\nu_1, \ldots, \nu_k$, which preserve equalities such that $t = \nu_1(\bar{t_1}), \ldots, t = \nu_k(\bar{t_k})$, where ν_i maps all constants into x and is the identity mapping for the null values.

Clearly, at the beginning there exist such mappings from all of the tuples of $\bar{r}$ to the tuples of r. Assume that D_i is applied on tuples $\bar{t_{i1}}, \ldots, \bar{t_{ij}} \in \bar{r}$ that created the tuple $\bar{t}$. Since ν_is preserve equalities, D_i can be applied on the generally corresponding tuples of $\bar{t_{i1}}, \ldots, \bar{t_{ij}}$, and the generated tuple t is defined as follows:

1. If D_i is an equality generating dependency which replaces all occurrences of a null value in $\bar{r}$ and, because ν_is are the identity mappings for the null values, the application of D_i on r will replace all occurrences of the same null value in the generally corresponding tuples.
2. If D_i is a tuple generating dependency that generated the new tuple $\bar{t} = (h(a_1), \ldots, h(a_n))$ in $\bar{r}$, when applied on the generally corresponding tuples it will generate a tuple

$$(\nu(h(a_1)), \ldots, \nu(h(a_n))) = \nu(h(a_1), \ldots, h(a_n))$$

in r.

But then, after the application of dependencies on both $\bar{r}$ and r, there is a generally corresponding tuple in r for every tuple $\bar{t} \in \bar{r}$. To complete the proof of completeness, we need to show that eliminating queries from $\mathcal{S}$ in Step 5 of Algorithm 4 still leaves the set $\mathcal{S}$ complete. Specifically, we need to show that if $\Pi_{Y'}\sigma_{C'}$ is eliminated from $\mathcal{S}$ in Step 5 because $\Pi_Y\sigma_C \not\rightsquigarrow_{\forall} \Pi_{Y'}\sigma_{C'}$ for a query $\Pi_Y\sigma_C$ in $\mathcal{S}$, then, for every $\Pi_{Y''}\sigma_{C''}$ such that

$$\Pi_{Y'}\sigma_{C'} \not\rightsquigarrow_{\exists} \Pi_{Y''}\sigma_{C''} \Rightarrow \Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y''}\sigma_{C''}.$$

First, since C must be consistent, by Proposition 5.2, universal dominance implies existential dominance and thus $\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y'}\sigma_{C'}$. Then, since $\Pi_{Y'}\sigma_{C'} \not\rightsquigarrow_{\exists} \Pi_{Y''}\sigma_{C''}$, by transitivity of existential dominance (Claim 6),

$$\Pi_Y\sigma_C \not\rightsquigarrow_{\exists} \Pi_{Y''}\sigma_{C''},$$

which completes the proof of completeness.

Proof of compactness. The compactness property follows directly from the way $\mathcal{S}$ is constructed in Step 5 of Algorithm 4. □

Similarly to Algorithm 3, the complexity of Algorithm 4 is $O(nT \cdot I^{(n+1)T})$, where I is the number of symbols in r (see Step 2 of Algorithm 4). Since I is bounded by qT, where q is the number of queries in $\mathcal{P}$, the complexity is bounded by $O(nT \cdot (qT)^{(n+1)T})$. Of course, a typical input size of Algorithm 4 is much smaller than the input size of Algorithm 3 since Algorithm 4 only considers queries, not the actual data. Next, we give an example of applying Algorithm 4.

Example 5.1. This is a detailed illustration of generating a data-independent disclosure cover for the queries,

$$\Pi_{SALARY,RANK}$$

and $\Pi_{NAME,RANK}$. The input to Algorithm 4 is:

$$\mathcal{P} = \{\Pi_{SALARY,RANK}, \Pi_{NAME,RANK}\}$$

and

$$\mathcal{D} = \{R[RANK = r, SALARY = s_1] \wedge R[RANK = r, SALARY = s_2] \rightarrow s_1 = s_2\}.$$

Table 4a shows tuples generated in Step 1 of Algorithm 5 from $\mathcal{P}$. Next, the functional dependency in $\mathcal{D}$ is applied with mapping h, where $h(r) = x$, $h(s_1) = x$, and $h(s_2) = \delta_3$. The result of the application is that δ_3 is replaced with x (Table 4b). Since the

TABLE 4
Relations Generated by Algorithm 4

NAME	RANK	SALARY	EXPERIENCE (years)
δ_1	x	x	δ_2
x	x	δ_3	δ_4

(a)

NAME	RANK	SALARY	EXPERIENCE (years)
δ_1	x	x	δ_2
x	x	x	δ_4

(b)

functional dependency cannot be applied again, $\mathcal{S}$ that is generated in Step 4 of Algorithm 4 is:

$$\mathcal{S} = \{\Pi_{RANK,SALARY}, \Pi_{NAME,RANK,SALARY}\}.$$

Since $\Pi_{NAME,RANK,SALARY} \twoheadrightarrow_{\forall} \Pi_{RANK,SALARY}$, the final output of Algorithm 1 is $\mathcal{S} = \{\Pi_{NAME,RANK,SALARY}\}$. Since $\mathcal{S}$ discloses a *top-secret* object, the second query is rejected.

For the case with constants, we need additional notions, such as weak-transitivity and generalized atom mapping, which we define next.

Definition 5.3 (Weak-transitivity). *Let E be a binary relation over the set V of variable names, null-values, and constants. E is* weakly transitive *if for every a, b, and c in V where a and c are not two different constants, $(a, b) \in E$, $(b, c) \in E$ imply that $(a, c) \in E$. From now on, we denote $(a, b) \in E$ by $a \sim b$.*

Definition 5.4 (Generalized atom mapping). *Let d be a dependency of the form $B_1, \ldots, B_n \rightarrow H$, r be a relation over the set V of variable names, null-values, and constant, and E be a weakly transitive binary relation over V. A* generalized atom mapping *h (with respect to d, r, and E) is a function $h : \{B_1, \ldots, B_n\} \rightarrow r$ such that*

1. *h preserves constants, i.e., if $h(R[\ldots, A_i = c, \ldots]) = (c_1, \ldots, c_i, \ldots, c_n)$ and c is a constant, then $c \sim c_i$.*
2. *h preserves equalities, i.e., if $B_i = R[\ldots, A_k = a, \ldots]$, $B_j = R[\ldots, A_l = a, \ldots]$, and*

$$h(B_i) = (c_1, \ldots, c_k, \ldots, c_n),$$
$$h(B_j) = (c'_1, \ldots, c'_l, \ldots, c'_n),$$

then $c_k \sim c'_l$.

Note that h extends to a mapping from the symbols of d to sets of symbols of r. The reason is that, by definition of atom mapping h, two equal symbols of d may be mapped to two (possibly different) symbols of r that are related by E, i.e., if $s_1, s_2 \in h(a)$, then $s_1 \sim s_2$. The application of dependencies is defined as follows:

Definition 5.5 (Application of dependencies). *An* E-Application *of a dependency d on (r, E) with respect to atom mapping h is defined as follows:*

1. *If d is an* equality-generating dependency *of the form $B_1, \ldots B_n \rightarrow a = b$, then, for every pair s_1, s_2 such that $s_1 \in h(a)$, $s_2 \in h(b)$, s_1, s_2 are not two different constants, and $s_1 \sim s_2$ is not in E*
   a. *add $s_1 \sim s_2$ to E, and*
   b. *close the result under weak transitivity*
2. *If d is a* tuple-generating dependency *of the form $B_1, \ldots B_n \rightarrow R[A_1 = a_1, \ldots, A_n = a_n]$, then select representatives $s_1, \ldots, s_n$ from the sets $h(a_1), \ldots, h(a_n)$ as follows:*
   a. *If there exists a constant $c \in h(a_i)$, then $s_i = c$,*
   b. *Else, if there exists a variable $x \in h(a_i)$, then $s_i = x$,*
   c. *Otherwise, $s_i = \delta_i$, where δ_i is any null-value in $h(a_i)$.*

   If tuple $(s_1, \ldots, s_n)$ is not in r, then add it to r.

Algorithm 5, presented in Fig. 6, computes all the queries that are existentially disclosed from a set $\mathcal{P}$ of queries by database constraints $\mathcal{D}$.

Theorem 5.3 (Existential disclosure). *Algorithm 5 terminates and its output is a* complete *and* compact *set of existentially disclosed queries.*

Proof. Algorithm 5 must *terminate* because the application of dependencies do not generate new symbols and, therefore, only a finite number of distinct tuples and relations of E' can be created. We need to show that the set $\mathcal{S}$ computed by the algorithm is 1) complete and 2) compact.

Proof of completeness. Let $\mathcal{S}$ be the output of Algorithm 5. To show completeness of $\mathcal{S}$, we have to prove that, for every query $\Pi_{Y'}\sigma_{C'}$, $\mathcal{P} \twoheadrightarrow_{\exists\mathcal{D}} \Pi_{Y'}\sigma_{C'}$ implies that there exists $\Pi_Y\sigma_C \in \mathcal{S}$ such that $\Pi_Y\sigma_C \twoheadrightarrow_{\exists} \Pi_{Y'}\sigma_{C'}$. Assume that $\mathcal{P} \twoheadrightarrow_{\exists D} \Pi_{Y'}\sigma_{C'}$, that is, by Definition 3.5, there exist

1. a relation r that satisfies $\mathcal{D}$,
2. sets of projection facts $P_i \subseteq \Pi_{X_i}\sigma_{C_i}(r), i = 1, \ldots, m$, and
3. $PF' \in \Pi_{Y'}\sigma_{C'}(r)$ such that

$$\{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_m, \Pi_{X_m}\sigma_{C_m})\} \models_{\mathcal{D}} (PF', \Pi_{Y'}\sigma_{C'}).$$

But then, by Theorem 4.2, if

$$\{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_m, \Pi_{X_m}\sigma_{C_m})\}$$

and $\mathcal{D}$ are given as the input for Algorithm 3, it must generate a QA-pair $(PF'', \Pi_{Y''}\sigma_{C''})$ that dominates $(PF', \Pi_{Y'}\sigma_{C'})$. The proof of completeness is based on the following claim:

Claim 7. *The output $\mathcal{S}$ of Algorithm 5 contains a query $\Pi_Y\sigma_C$ such that $\Pi_Y\sigma_C \twoheadrightarrow_{\exists} \Pi_{Y''}\sigma_{C''_e}$, where C''_e is the extension of C'' with all the equalities explicitly given in PF''.*

We first complete the proof of completeness and then prove Claim 7. By construction of $\Pi_Y\sigma_C$ in Step 5 of Algorithm 5, $\Pi_Y\sigma_C$ is in normal form. Let $\Pi_{Y^*}\sigma_{C'}$ be the normal form of the query $\Pi_{Y'}\sigma_{C'}$. To prove $\Pi_Y\sigma_C \twoheadrightarrow_{\exists} \Pi_{Y'}\sigma_{C'}$, it is sufficient to show that,

$$\Pi_Y\sigma_C \twoheadrightarrow_{\exists} \Pi_{Y^*}\sigma_{C'},$$

Algorithm 5: General Data-Independent Disclosure Algorithm

INPUT
1. $\mathcal{P} = \{\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_m}\sigma_{C_m}\}$ (C_is are conjunctions of equalities of the forms $A_i = A_j$ or $A_i = c$, where A_i, A_j are attributes and c is a constant..)
2. $\mathcal{D}$ (set of Horn-clause constraints)

OUTPUT $\mathcal{S}$ (Complete and compact set of existentially disclosed queries)

METHOD

1. (*Initialization*) **for** every $\Pi_{X_i}\sigma_{C_i} \in \mathcal{P}$, $(i = 1, \ldots, m)$ *generate* a tuple t_i by assigning symbols to each attribute A_j, $j = 1, \ldots, n$ as follows:
   (a) **if** $C_i \models A_j = c$, where c is a constant, then assign c to A_j.
   (b) **else**, **if**
   - $A_j \in X_i$ or
   - there exists $A_k \in X_i$ such that $C_i \models A_j = A_k$

   then assign variable x to A_j,
   (c) **otherwise**, assign null-values to A_j as follows:
   - **if** $C_i \models A_j = A_k$, $(k = 1, \ldots, j$ and A_k was assigned δ_k then assign δ_k to A_j.
   - **else**, assign a unique null-value δ_j to A_j.
2. *Construct* r to be $\{t_1, \ldots, t_m\}$
3. *Construct* a weakly transitive binary relation E over the set V of symbols of r and the constants of $\mathcal{D}$ as follows:
   (a) E contains all *reflexive pairs* of the form $x \sim x$, $\delta_i \sim \delta_i$, and $c \sim c$.
   (b) **for** every constant c that appears in r or in $\mathcal{D}$, E contains equalities of the form $x \sim c$.
4. *E-Chase* (r, E) with $\mathcal{D}$; that is, E-Apply $\mathcal{D}$ on (r, E) until no more changes occur. The result is (r', E').
5. Compute $\mathcal{S}$ as follows:
   for each tuple $t \in r'$ generate a query $\Pi_Y\sigma_C$, where
   (a) Y contains all the attributes A_i such that
   - $t[A_i] = x$,
   - $t[A_i] = c$, where c is a constant, or
   - $t[A_i] = \delta_i$ and E' contains either $\delta_i \sim c$ or $\delta_i \sim x$

   (b) C contains the following equalities
   - $A_i = c$ if $t[A_i] = c$ and
   - $A_i = A_j$ if $A_i, A_j \notin Y$, and $t[A_i] = \delta_i$, $t[A_j] = \delta_j$, and E' contains $\delta_i \sim \delta_j$.
6. (a) Continue until no more changes occur:
   for every two different queries $\Pi_Y\sigma_C$ and $\Pi_{Y'}\sigma_{C'}$ in $\mathcal{S}$, if $\Pi_Y\sigma_C \rightsquigarrow_\forall \Pi_{Y'}\sigma_{C'}$ then remove $\Pi_{Y'}\sigma_{C'}$ from $\mathcal{S}$.
   (b) **for** every query $\Pi_Y\sigma_C$ in $\mathcal{S}$, replace C by an equivalent minimal subset of C.
7. Return $\mathcal{S}$ as output.

Fig. 6. General data-independent disclosure algorithm.

that is, by Proposition 5.1,

1. $Y^* \subseteq Y$,
2. $C \wedge C'$ is consistent, and
3. $C \models C'_{\neg Y}$.

In the proof, we use the following facts:

Fact 1. By Proposition 4.1, $(PF'', \Pi_{Y''}\sigma_{C''}) \models (PF', \Pi_{Y'}\sigma_{C'})$ implies $C''_e \models C'_e$.

Fact 2. By Proposition 5.1, $\Pi_Y\sigma_C \rightsquigarrow_\exists \Pi_{Y''}\sigma_{C''_e}$ implies

1. $Y'' \subseteq Y$,
2. $C''_e \wedge C$ is consistent, and
3. $C \models C''_{\neg Y}$

By construction of $(PF'', \Pi_{Y''}\sigma_{C''})$ in Algorithm 3, Y'' contains all the attributes that have constants assigned to and C'' contains only equalities among attributes that are not in Y''. By Fact 1, $C''_e \models C'_e$ and, therefore, C''_e must entail all of the equalities of the form $A_i = c$ which are explicitly given in PF'. Therefore, $Y^* \subseteq Y''$ must hold. This and $Y'' \subseteq Y$ (Fact 2.1) gives $Y^* \subseteq Y$ follows.

From $C''_e \models C'_e$ (Fact 1) and $C''_e \wedge C$ is consistent (Fact 2.2), it follows that $C \wedge C'$ is consistent. The only remaining part is to show that $C \models C'_{\neg Y}$. Since $(PF'', \Pi_{Y''}\sigma_{C''})$ is in normal form (Theorem 4.2), C''_e contains two disjoint groups of equalities: 1) equalities among attributes that are not in Y'' (equalities of C'') and 2) equalities among attributes of Y'' (equalities of PF''). Therefore, if $C''_e \models A_i = A_j$, where A_i and A_j are not in Y'', then only equalities of group a can be used to derive the entailment. We know that $C''_e \models C'_e$, more specifically, $C''_e \models C'_{\neg Y}$. Since $Y = Y''$ and, by the above argument, we have $C''_{\neg Y} \models C'_{\neg Y}$. Then, by using $C \models C''_{\neg Y}$ (Fact 2.3) and the transitivity of logical entailment, we have $C \models C'_{\neg Y}$. This completes the proof of completeness.

To prove Claim 7, we use the notion of *generally corresponding tuples*. Initially, we define general corresponding tuples as follows: Let r_1 denote the relation that was generated from $\{(P_1, \Pi_{X_1}\sigma_{C_1}), \ldots, (P_m, \Pi_{X_m}\sigma_{C_m})\}$ by Algorithm 3 in Step 2 as r' and r_2 denote the relation generated from $\{\Pi_{X_1}\sigma_{C_1}, \ldots, \Pi_{X_m}\sigma_{C_m}\}$ by Algorithm 5 in Step 2 as r. Also, let $t_{i_1}, \ldots, t_{i_l}$ in r_1 denote the tuples that were generated in Step 1 of Algorithm 3 from the QA-pair $(P_i, \Pi_{X_i}\sigma_{C_i})$ $(i = 1, \ldots, m)$ and $\bar{t}_i$ in r_2 denote the tuple that was generated from the query $\Pi_{X_i}\sigma_{C_i}$ $(i = 1, \ldots, m)$ in Step 1 of Algorithm 5. Initially, we say, that $\bar{t}_i$ *generally corresponds* to the tuples $t_{i_1}, \ldots, t_{i_l}$. Let ν be the mapping that maps each tuple of r_1 to its generally corresponding tuple in r_2.

Consider the applications $a_1 = (d_1, h_1), \ldots, a_k = (d_k, h_k)$ of dependencies in Algorithm 3 on r_1, where d_i $(i = 1, \ldots, k)$ is the dependency being applied and h_i $(i = 1, \ldots, k)$ is the mapping being used. Let r_1^i, $(i = 1, \ldots, k)$ denote the relation that was produced from $r_1 = r_1^0$ after applications of $a_1, \ldots, a_i$. In the following claim, we recursively define the corresponding applications $\bar{a}_1 = (d_1, \bar{h}_1), \ldots, \bar{a}_k = (d_k, \bar{h}_k)$ on $r_2 = r_2^0$ (to be used in Algorithm 5) that will produce $r_2^1, \ldots, r_2^k$, and relations $E = E_0, E_1, \ldots, E_m$. Further, we define the mappings $\nu = \nu_0, \nu_1, \ldots, \nu_k$, each ν_i $(i = 0, \ldots, k)$ from r_1^i to r_2^i.

Claim 8. *$\nu = \nu_0$ from $r_1 = r_1^0$ to $r_2 = r_2^0$ preserves constants and equalities, i.e., for tuples t_l and t_k in r_1,*

1. *If $t_l[A_i] = c$, where c is a constant, then*
   a. *$\nu(t_l)[A_i] \neq c_1$, where c_1 is a constant different from c(i.e., $c_1 \neq c$), and*
   b. *$\nu(t_l)[A_i] \sim x$.*
2. *If $t_l[A_i] = t_k[A_j]$, then $\nu(t_l)[A_i] \sim \nu(t_k)[A_j]$, where $\sim$ denotes the relation E defined in Step 3 of Algorithm 5.*

Further, for all i, $i = 1, \ldots, k$

1. Definition. *Given the application $a_i = (d_i, h_i)$, where d_i is of the form $\{B_1, \ldots, B_n\} \rightarrow H$, $\bar{h}_i$ is defined as $\bar{h}_i(B_j) \stackrel{def}{=} \nu_{i-1}(h_i(B_j))$, for all $j = 1, \ldots, n$.*
2. Claim. *$\bar{h}_i$ is indeed a generalized atom mapping with respect to d_i, r_2^{i-1}, and E_{i-1}, i.e., $\bar{h}_i$ preserves constants and equalities.*
3. Definition. *ν_i is defined as follows:*
   a. *If d_i is an equality-generating dependency and let $t' \in r_1^i$ be the tuple that was originated from $t \in r_1^{i-1}$, then if $\nu_{i-1}(t) = \bar{t}$, then $\nu_i(t') = \bar{t}$.*
   b. *If d_i is a tuple-generating dependency such that a_i generated the tuple $t' \in r_1^i$ and $\bar{a}_i$ generated $\bar{t}' \in r_2^i$, then $\nu_i(t) = \nu_{i-1}(t)$ for all tuples t in r_1^{i-1} and $\nu_i(t') = \bar{t}'$.*
4. Claim. *ν_i from r_1^i to r_2^i preserves constants and equalities.*

Before proving Claim 8, we complete the proof of Claim 7. Let $a_1, \ldots, a_m$ be the dependency applications on r_1 by Algorithm 3 that generated the tuple t that was used in Step 3 to construct the QA-pair $(PF'', \Pi_{Y''}\sigma_{C''})$. Let $\bar{a}_1, \ldots, \bar{a}_m$ be the corresponding dependency applications, where each $\bar{h}_i$ $(i = 1, \ldots, m)$ is constructed as defined in Claim 8.2. We order the dependency applications performed by Algorithm 5 on r_2 such that $\bar{a}_1, \ldots, \bar{a}_m, \ldots, \bar{a}_n$, i.e., $\bar{a}_1, \ldots, \bar{a}_m$ are completed before any other dependency is applied by Algorithm 5. We can reorder the dependency applications of Algorithm 5, by the following Claim 9 which will be proven after we complete the proof of Claim 7.

Claim 9. *The output of Algorithm 5 is independent of the order of the dependency applications.*

Next, we apply $\bar{a}_1, \ldots, \bar{a}_m$ on r_2 in Step 4 of Algorithm 5. By Claim 8, items 2 and 4, we know that, for every tuple t in r_1^m, there is a tuple $\bar{t}$ in r_2^m such that $\nu_m(t) = \bar{t}$ and ν_m preserves equalities and constants. Note that it is possible that Algorithm 5 will apply additional dependencies after $\bar{a}_1, \ldots, \bar{a}_m$ are completed. However, since a dependency application can only add tuples to r_2^m or extend E_m, the final relation r' at the end of Step 4 of Algorithm 5 must contain $\bar{t}$ and E' must contain E_m.

Let $\Pi_Y\sigma_C$ be the query that was generated in Step 5 of Algorithm 5 from $\bar{t}$ and E'. To complete the proof of Claim 7, we need to show that $\Pi_Y\sigma_C$ existentially discloses $\Pi_{Y''}\sigma_{C''_e}$, i.e., by Proposition 5.1 and 5.2, that $Y'' \subseteq Y$, $C \wedge C''_e$ is consistent, and $C \models C'''_{\neg Y}$. $Y'' \subseteq Y$ follows from the fact that Y'' are exactly all attributes in which t has a constant, the fact that ν_m (recall $\bar{t} = \nu_m(t)$) preserves constants and the way of construction Y in Step 5 of Algorithm 5.

To show that $C \wedge C''_e$ is consistent, we use the following observations:

Observation 1. *$C = C_Y \wedge C_{\neg Y}$, where 1) C_Y is the conjunction of equalities of C of the form $A_i = c$, where A_i is an attribute name and c is a constant, and $A_i \in Y$, i.e., $\bar{t}[A_i] = c$, and 2) $C_{\neg Y}$ is the conjunction of equalities of C of the form $A_i = A_j$, where A_i or A_j is not in Y, and $\bar{t}[A_i] \sim \bar{t}[A_j]$. We claim that $Attr(C_Y) \cap Attr(C_{\neg Y}) = \emptyset$.*

Proof. Assume, by contradiction, that

$$A_i \in Attr(C_Y) \cap Attr(C_{\neg Y}).$$

But then, $A_i = c$ is in C, where c is a constant and $A_i = A_j$ is in C, where $A_i \in Y$ and $A_j \notin Y$. Then, $\bar{t}[A_i] = c$, and by construction of E, $c \sim x$. Also, $\bar{t}[A_j] = s$, where s is not a constant and $s \not\sim x$ because $A_j \notin Y$. However, from

$\bar{t}[A_i] \sim \bar{t}[A_j]$, we have $s \sim x$, which constitutes a contradiction, thus $Attr(C_Y) \cap Attr(C_{\neg Y}) = \emptyset$ holds.

Observation 2. *$C''_e = C''_{Y''} \wedge C''_{\neg Y''}$, where 1) $C''_{Y''}$ is the conjunction of equalities of C''_e of the form $A_i = c$, where A_i is an attribute name and c is a constant, and $A_i \in Y''$, i.e., $t[A_i] = c$, and 2) $C''_{\neg Y''}$ is the conjunction of equalities of C''_e of the form $A_i = A_j$, where A_i or A_j is not in Y'' and $t[A_i] = t[A_j]$. We claim that $Attr(C''_{Y''}) \cap Attr(C''_{\neg Y''}) = \emptyset$.*

Proof. Similar to the proof of 1.

Observation 3. *$C''_{\neg Y''} = C''_{Y-Y''} \wedge C''_{\neg Y}$, where 1) $C''_{Y-Y''}$ is the conjunction of equalities of $C''_{\neg Y''}$, where A_i or A_j are in Y, and 2) $C''_{\neg Y}$ is the conjunction of equalities of C''_e, where A_i or A_j are not in Y. We claim that $Attr(C''_{Y-Y''}) \cap Attr(C''_{\neg Y}) = \emptyset$.*

Proof. Assume, by contradiction, that $A_i = A_j$ in $C''_{\neg Y''}$, where $A_i \in Y - Y''$ and $A_j \in \neg Y$. Then, $t[A_i] = t[A_j] = \delta$, where δ is some null-value. Because ν_m preserves equalities, we know that

$$\nu_m(t)[A_i] = \bar{t}[A_i] \sim \nu_m(t)[A_j] = \bar{t}[A_j].$$

Further, because $A_i \in Y$, $\bar{t}[A_i] \sim x$. However, because $A_j \notin Y$, $\bar{t}[A_j]$ not a constant and $\bar{t}[A_j] \not\sim x$. But this is a contradiction.

To complete the proof that $C \wedge C''_e$ is consistent, consider a graph G, in which the nodes are all the attribute names, null-values, and constants of C and C''_e. There is an edge between two nodes A and B if and only if C or C''_e contains the equality $A = B$. By contradiction, assume that $C \wedge C''_e$ is inconsistent. Then, there must exist a path in G from a constant to a different constant. Moreover, there must exist such a path, say P, of *minimal length*, i.e., every path that is shorter than P cannot be inconsistent. Let a and b ($a \neq b$) be the two end points of P and e_1 be the first edge in P from a. The edge must be of the form (a, A), i.e., corresponding to the equation $a = A$, where A is an attribute name. Therefore, by 1 and 2, e_1 is either in $C''_{Y''}$ or in C_Y.

Case 1. If e_1 is in $C''_{Y''}$, then $t[A] = a$. Consider the second edge e_2. It must be of the form (A, c), where c is a constant, because $Y'' \subseteq Y$ and, from Observations 1 and 2, we know that A cannot appear in equalities of $C''_{\neg Y''}$ or $C_{\neg Y}$. So, e_2 must appear in $C''_{Y''}$ or in C_Y that have equalities of the form (A, c). If e_2 is in $C''_{Y''}$, then $c = a$ because $C''_{Y''}$ is consistent. If e_2 is in C_Y, then $\bar{t}[A] = \nu_m(t)[A] = c$. Since ν_m preserves constants, c must be the same constant as a. But then, the node a appears twice in P, which contradicts the minimality of P.

Case 2. If e_1 is in C_Y, then $\bar{t}[A] = a$. Consider the second edge e_2. If $A \in Y''$, then an argument similar to Case 1 can be repeated to derive contradiction to the minimality of P. Otherwise, if $A \in Y - Y''$, then, by Observations 1, 2, and 3, e_2 must be either in C_Y or in $C''_{Y-Y''}$. If e_2 is in C_Y, then it must be of the form $A = c$ and, because C_Y is consistent, $a = c$ must hold, contradicting the minimality of P. If e_2 is in $C''_{Y-Y''}$, then it must be of the form (A, A_1), where A_1 is an attribute name. Let $(a, A), (A, A_1), \ldots, (A_{k-1}, A_k), (A_k, c)$ denote a subpath of P from a to the first constant node c. By 1, 2, and 3, all edges $(A, A_1), \ldots, (A_{k-1}, A_k)$ must come from $C''_{Y-Y''}$, and $t[A] = t[A_1] = \ldots, = t[A_k] = \delta$, where δ is some null-value. The edge (A_k, c) must be in C_Y and, therefore, $\bar{t}[A_k] = c$. Because ν_m preserves equalities, we know that

$$\nu_m(t)[A] = a \sim \nu_m(t)[A_1] \sim \ldots \sim \nu_m(t)[A_k] = c.$$

Then, $a \sim c$ is in E. Since E cannot contain $a \sim c$ for two different constant, c must be the same constant as a. But then, the node a appears twice in P, which contradicts the minimality of P.

This completes the proof that there does not exist a path in G that connects two different constants, and, therefore $C \wedge C''_e$ must be consistent.

Finally, if t has the same null-value for attributes A_i and A_j, then, because ν_m preserves equalities, attributes A_i and A_j of $\bar{t}$ must have symbols that are related by E. If A_i and A_j are not in Y, then both $\bar{t}[A_i]$ and $\bar{t}[A_j]$ are two null-values which are related by E and, therefore, C contains the equality $A_i = A_j$. Therefore, $C \models C''_{\neg Y}$. But then, by Proposition 5.1, we proved that, for any $(PF'', \Pi_{Y'}\sigma_{C''})$, Algorithm 5 will generate a query $\Pi_Y\sigma_C$ such that $\Pi_Y\sigma_C \overset{\sim}{\rightarrow}_{\exists} \Pi_{Y''}\sigma_{C''_e}$.

To complete Claim 7, we have to show that, after the query $\Pi_Y\sigma_C$ is removed from $\mathcal{S}$ in Step 6a of Algorithm 5 because there exists a query $\Pi_{\bar{Y}}\sigma_{\bar{C}} \in \mathcal{S}$ such that $\Pi_{\bar{Y}}\sigma_{\bar{C}} \overset{\sim}{\rightarrow}_{\forall} \Pi_Y\sigma_C$, we still have $\Pi_{\bar{Y}}\sigma_{\bar{C}} \overset{\sim}{\rightarrow}_{\exists} \Pi_{Y''}\sigma_{C''_e}$.

1. From $\Pi_Y\sigma_C \overset{\sim}{\rightarrow}_{\exists} \Pi_{Y''}\sigma_{C''_e}$, we know that
   a. $Y'' \subseteq Y$,
   b. $C''_e \wedge C$ is consistent, and
   c. $C \models C''_{\neg Y}$.
2. From $\Pi_{\bar{Y}}\sigma_{\bar{C}} \overset{\sim}{\rightarrow}_{\forall} \Pi_Y\sigma_C$, we know that
   a. $Y \subseteq \bar{Y}$,
   b. $C \wedge \bar{C}$ is consistent,
   c. $\bar{C} \models C_{\neg \bar{Y}}$, and
   d. $C \models \bar{C}_Y$.

From 1a and 2a, it follows that $Y'' \subseteq \bar{Y}$. To show that $\bar{C} \wedge C''_e$ is consistent, assume, by contradiction, that $\bar{C} \wedge C''_e$ is inconsistent, i.e., $C''_e \models A_i = a$ and $\bar{C} \models A_i = b$, where a and b are two different constants. By construction, both $\Pi_{\bar{Y}}\sigma_{\bar{C}}$ and $\Pi_{Y''}\sigma_{C''_e}$ are in normal form. Then, if $C''_e \models A_i = a$, $A_i \in Y''$ and, since $Y'' \subseteq Y$, $A_i \in Y$. From 2d, we know that $C \models \bar{C}_Y$ and, therefore, $C \models A_i = b$. Moreover, from 1b, we know that $C''_e \wedge C \models A_i = a \wedge A_i = b$ is consistent. But, this is a contradiction, therefore, $\bar{C} \wedge C''_e$ is consistent. Finally, from 1c and 2c and the facts that $Y'' \subseteq Y \subseteq \bar{Y}$, it follows that $\bar{C} \models C''_{\neg \bar{Y}}$. Thus, by Proposition 5.1, $\Pi_{\bar{Y}}\sigma_{\bar{C}} \overset{\sim}{\rightarrow}_{\exists} \Pi_{Y''}\sigma_{C''_e}$. This completes the proof of Claim 7.

We **prove Claim 8** by induction on the number of dependency applications in Algorithm 3 and the corresponding applications in Algorithm 5. First, we show that ν_0 preserves constants and equalities. Consider tuples $t_{i_1}, \ldots, t_{i_l}$ in r^0_i and $\bar{t}_i$ in r^0_2, as they were defined initially for generally corresponding tuples. By construction of t_{i_j} ($j = 1, \ldots, l$) in Step 1 of Algorithm 3, t_{i_j} contains a constant for an attribute A_k if and only if either

1. $C_i \models A_k = c$,
2. $A_k \in X_i$, or
3. there exists $A_j \in X_i$ such that $C_i \models A_k = A_j$.

In addition, by construction of $\bar{t}_i$ in Step 1 of Algorithm 5, we know that 1) if $C_i \models A_k = c$, then $\bar{t}_i[A_k] = c$ else 2) if $A_k \in X_i$ or there exists $A_j \in X_i$ such that $C_i \models A_k = A_j$, then $\bar{t}_i[A_k] = x$.

Then, for all attributes A_k that t_{i_j} $(j = 1, \ldots, l)$ contains a constant, $\bar{t}_i$ contains either a constant or the symbol x. If $\bar{t}_i[A_k] = c$ $(C_i \models A_k = c)$, then all tuples t_{i_j} $(j = 1, \ldots, l)$ must have $t_{i_j}[A_k] = c$ and $\nu_0(t_{i_j})[A_k] = c$; Property 1a must hold. Because E, that was constructed in Step 3 of Algorithm 5, contains all reflexive pairs $x \sim x$ and $x \sim c$ for all constants c that appear in r_2 or in the dependencies, Property 1b must hold and, therefore, Property 1 holds. Moreover, equalities among the constants of t_{i_j} $(j = 1, \ldots, l)$ are preserved. To show that ν_0 also preserves equalities among the null-values of t_{i_j}, notice that, initially, we assign unique null-values for all attributes of t_{i_j} $(j = 1, \ldots, l)$ that do not already have a constant assigned to. In Step 1c, Algorithm 3 will equate null-values of t_{i_j} only if C_i entails their equalities. But then, by construction of $\bar{t}_i$ in Step 1c of Algorithm 5, $\bar{t}_i$ must contain the same null-value for these attributes and, therefore, ν_0 preserves equalities. This completes the proof that ν_0 preserves constants and equalities.

Assume that the claim holds for $i-1$ application, i.e., that ν_{i-1} preserves constants and equalities. Note that h_i is an atom mapping, thus it also preserves constants and equalities. Then, if

$$h_i(R[\ldots, A_j = c, \ldots]) = (c_1, \ldots, c_j, \ldots, c_n) \in r_1^{i-1},$$

then $c_j = c$. Moreover, if

$$\nu_{i-1}(c_1, \ldots, c_j, \ldots, c_n) = (s_1, \ldots, s_j, \ldots, s_n) \in r_2^{i-1},$$

then s_j is not a different constant from c_j, and $x \sim s_j$. By construction of E, we know that $x \sim c$ and, therefore, $s_j \sim c$, thus

$$\bar{h}_i(R[\ldots, A_j = c, \ldots]) = (s_1, \ldots, s_j, \ldots, s_n),$$

where $s_j \sim c \sim x$, thus, $\bar{h}_i$ preserves constants.

Further, if

$$h_i(R[\ldots, A_k = a, \ldots]) = (c_1, \ldots, c_k, \ldots, c_n) \in r_1^{i-1}$$

and

$$h_i(R[\ldots, A_l = a, \ldots]) = (c'_1, \ldots, c'_l, \ldots, c'_n) \in r_1^{i-1},$$

then $c_k = c'_l$. Moreover, if

$$\nu_{i-1}(c_1, \ldots, c_k, \ldots, c_n) = (s_1, \ldots, s_k, \ldots, s_n) \in r_2^{i-1}$$

and

$$\nu_{i-1}(c'_1, \ldots, c'_l, \ldots, c'_n) = (s'_1, \ldots, s'_l, \ldots, s'_n) \in r_2^{i-1},$$

then $s_k \sim s'_l$. Therefore,

$$\bar{h}_i(R[\ldots, A_k = a, \ldots]) = (s_1, \ldots, s_k, \ldots, s_n),$$
$$\bar{h}_i(R[\ldots, A_l = a, \ldots]) = (s'_1, \ldots, s'_l, \ldots, s'_n),$$

and $s_k \sim s'_l$, thus $\bar{h}_i$ preserves equalities.

The only remaining part is to show that ν_i from r_1^i to r_2^i preserves constants and equalities. If d_i is an **equality-generating dependency** of the form $B_1, \ldots, B_n \rightarrow a = b$, then a_i equates two symbols $h_i(a)$ and $h_i(b)$ of r_1^{i-1} by replacing all occurrences of one of them with the other, say all occurrences of $h_i(b)$ are replaced with $h_i(a)$. We know that all occurrence of $h_i(b)$ are mapped by ν_{i-1} into symbols of r_2^{i-1}, which are related to each other by E_{i-1}. The application $\bar{a}_i$ extends E_{i-1} by adding relations $s_1 \sim s_2$ for every pair s_1, s_2 such that $s_1 \in \bar{h}_i(a) = \nu_{i-1}(h_i(a))$ and $s_2 \in \bar{h}_i(b) = \nu_{i-1}(h_i(b))$ and closes it under weak transitivity.

Since a_i equated $h_i(a)$ and $h_i(b)$ of r_1^{i-1} and ν_{i-1} preserves constants and equalities, we know that $\bar{h}_i(a)$ and $\bar{h}_i(b)$ cannot be different constants. Moreover, by extending E_{i-1} for all possible pairs $s_1 \sim s_2$, it is ensured that all occurrences of $h(a)$ in r_1^i are mapped into symbols of r_2^i that are related by E_i. For example, if $t_1[A_l] = h_i(a)$, $t_2[A_k] = h_i(b)$, $\nu_{i-1}(t_1)[A_l] = s_1$, and $\nu_{i-1}(t_2)[A_k] = s_2$, then, after the applications a_i and $\bar{a}_i$, we have $t_2[A_k] = h_i(a)$, $\nu_{i-1}(t_2)[A_k] = s_2$ where $s_1 \sim s_2$ is in E_i. Therefore, after the applications (d_i, h_i) on r_1^{i-1} and $(d_i, \bar{h}_i)$ on r_2^{i-1}, where d_i is an equality-generating dependency, ν_i from r_1^i to r_2^i preserves constants and equalities.

If d_i is a **tuple-generating dependency** and a_i generated the tuple $t = (h_i(a_1), \ldots, h_i(a_n))$ in r_1^i, then $\bar{a}_i$ will generate a tuple $\bar{t}$ in r_2^i, where

$$\bar{t} = (\nu_{i-1}(h_i(a_1)), \ldots, \nu_{i-1}(h_i(a_n))) = \nu_{i-1}(h_i(a_1), \ldots, h_i(a_n)).$$

Then, $\nu_{i-1}(t) = \bar{t}$. Clearly, ν_i from r_1^i to r_2^i coincides with ν_{i-1} on the tuples of r_1^{i-1} and r_2^{i-1}. Further, ν_i maps the new tuple $t \in r_1^i$ to the new tuple $\bar{t} \in r_2^i$, where $\nu_{i-1}(t) = \bar{t}$. But then, since ν_{i-1} preserved constants and equalities, ν_i does too. This completes the proof of Claim 8.

The only remaining part is the proof of Claim 9. Let (r'_2, E') and (r''_2, E'') denote the results of two different sequences of dependency applications, $\bar{a}'_1, \ldots, \bar{a}'_l$ and $\bar{a}''_1, \ldots, \bar{a}''_k$, at the end of Step 4 of Algorithm 5. Assume, by contradiction, that (r''_2, E'') and (r''_2, E'') are not the same (up to the renaming of the null-values), i.e., one of them contains either a tuple or a pair $s_1 \sim s_2$ that is not present in the other. Assume that (r'_2, E') contains a tuple or a pair that is not in (r''_2, E''). Since r''_2 must contain all the tuples of r_2 and E'' must contain all of the relations of E, we can apply $\bar{a}'_1, \ldots, \bar{a}'_l$ on (r''_2, E''), i.e., using r''_2 instead of r_2 and E'' instead of E. We know that the applications $\bar{a}'_1, \ldots, \bar{a}'_l$ must produce the tuple or the relation that was not in (r''_2, E''). But, this is a contradiction because, originally, when (r''_2, E'') was generated, the *E-Chase* process could be terminated only if no more dependency could be applied that resulted in a change. Therefore, (r'_2, E') and (r''_2, E'') must be equal, and this completes the proof of Claim 9.

Proof of compactness. The compactness property follows directly from the way $\mathcal{S}$ is constructed in Step 6 of Algorithm 5. □

TABLE 5
Tuples Generated in Step 1 of Algorithm 5 from the Queries $\Pi_{SALARY\sigma RANK=Clerk}$ and $\Pi_{NAME,RANK\sigma EXPERIENCE=10}$

NAME	RANK	SALARY	EXPERIENCE (years)
δ_1	Clerk	x	δ_2
x	x	δ_3	10

The complexity analysis of Algorithm 5 is similar to the one of Algorithm 4, but requires an additional argument. The difference is that not every iteration of the *E-Chase* in Step 4 of Algorithm 5 creates a new tuple. Each application of a tuple-generating dependency is bounded by I^T, where I is the number of symbols in r and T is the number of attributes of r. Every equality-generating dependency application must add a pair to the binary relation E. Since the size of E is bounded by I^2, the number of such iterations is bounded by I^2. In total, the number of iterations is bounded by $I^T + I^2$, which is $O(I^T)$, the same as the complexity of the *chase* in Algorithm 4. Therefore, the complexity of Algorithm 5 is the same as the complexity of Algorithm 4, that is, $O(nT \cdot (qT)^{(n+1)T})$, where q is the number of queries in $\mathcal{P}$. Next, we give an example to show how data-independent disclosure cover can be computed.

Example 5.2. This is the detailed illustration of generating the data-independent disclosure cover for the queries, $\Pi_{SALARY}\sigma_{RANK=Clerk}$ and $\Pi_{NAME,RANK}\sigma_{EXPERIENCE=10}$, which were presented in Example 2.1. The input of Algorithm 5 is:

$$\mathcal{P} = \{\Pi_{SALARY}\sigma_{RANK=Clerk}, \Pi_{NAME,RANK}\sigma_{EXPERIENCE=10}\}$$

and

$$\mathcal{D} = \{R[RANK = r, SALARY = s_1] \wedge R[RANK = r, SALARY = s_2] \rightarrow s_1 = s_2\}.$$

Table 5 shows tuples generated in Step 1 of Algorithm 5 from $\mathcal{P}$. The weakly transitive relation, E, contains all reflexive pairs of the form $s \sim s$ over the symbols s of Table 5 and the pairs $x \sim Clerk$ and $x \sim 10$. Next, the functional dependency in $\mathcal{D}$ is applied with generalized atom mapping h, where $h(r) = Clerk \sim x$, $h(s_1) = x$, and $h(s_2) = \delta_3$. The result of the application is the extension of E with the pair $x \sim \delta_3$. After this, E is closed under weak transitivity to derive new relations $Clerk \sim x \sim \delta_3$ and $10 \sim x \sim \delta_3$. Since the functional dependency cannot be applied again, the final output of Algorithm 5 is

$$\mathcal{S} = \{\Pi_{NAME,RANK,SALARY,EXPERIENCE}\sigma_{EXPERIENCE=10}\}.$$

Since $\mathcal{S}$ discloses a *top-secret* object, the second query is rejected.

6 Conclusions and Future Work

In this paper, we attempted a systematic classification of the inference problem in terms of

1. data-dependent versus data-independent disclosure,
2. properties of soundness and completeness of disclosure inference algorithms to ensure confidentiality of data while supporting data availability,
3. expressiveness of constraints domains used to represent database and metadata constraints, and
4. design-time versus query-time inference detection and elimination frameworks.

Most importantly, we developed actual inference algorithms for both data-dependent and data-independent disclosures for a highly expressive family of Horn-clause constraints. These constraints can express not only functional, multivalued, and join dependencies (or their combinations), but also metadata that might be known by users.

We conclude by listing some suggestions for *further work*. This paper focuses on multiple attacks of a single user. However, in a real-life environment, it is possible that malicious users share their information to obtain data for which they do not have the proper authorization. With little modification, DiMon can protect against collaborating users as well. From the perspective of users' interaction, we distinguish among the following types of attacks:

1. single user attacks, where each user is considered individually,
2. attacks of collaborating users, where all of the users are assumed to collaborate with each other, and
3. attacks of collaborating groups of users, where only certain groups of users are assumed to share their information.

Another future direction is to incorporate the problem of data aggregation into the model [10]. The aggregation problem occurs if a certain number (threshold) of data items can be released safely while if the size of the aggregate exceeds this number, the security is violated. We believe that database dependencies, especially tuple generating dependencies, should be incorporated to provide secure aggregation control. Note that the inference problem can be viewed as a special type of aggregation problem with zero threshold value.

Finally, our model could be extended to handle other kinds of constraints, such as more general arithmetic constraints or embedded generalized dependencies. How to do this with preservation of soundness and completeness of inference algorithms remains an open question.

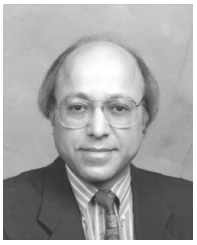
References

[1] A. Brodsky, C. Farkas, and S. Jajodia, "Data Disclosure and Inference Channels," technical report, George Mason Univ., 2000.
[2] L.J. Buczkowski, "Database Inference Controller," *Database Security III: Status and Prospects,* D.L. Spooner and C. Landwehr, eds., pp. 311–322, 1990.
[3] S. Dawson, S. De, C. di Vimercati, and P. Samarati, "Minimal Data Upgrating to Prevent Inference and Association Attacks," *Proc. 18th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems,* pp. 114–125, 1999.
[4] S. Dawson, S. De, C. di Vimercati, and P. Samarati, "Specification and Enforcement of Classification and Inference Constraints," *Proc. IEEE Symp. Security and Privacy,* 1999.

[5] D.E. Denning, "Commutative Filters for Reducing Inference Threats in Multilevel Database Systems," *Proc. IEEE Symp. Security and Privacy,* pp. 134–146, 1985.
[6] J.A. Goguen and J. Meseguer, "Unwinding and Inference Control," *Proc. IEEE Symp. Security and Privacy,* pp. 75–86, 1984.
[7] T.H. Hinke, "Inference Aggregation Detection in Database Management Systems," *Proc. IEEE Symp. Security and Privacy,* pp. 96–106, 1988.
[8] S. Jajodia and C. Meadows, "Inference Problems in Multilevel Secure Database Management Systems," *Information Security: An Integrated Collection of Essays,* M.D. Abrams, S. Jajodia, and H. Podell, eds., pp. 570–584, Los Alamitos, Cailf.: IEEE CS Press, 1995.
[9] D.G. Marks, "Inference in MLS Database Systems," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, no. 1, pp. 46–55, Feb. 1996.
[10] D.G. Marks, A. Motro, and S. Jajodia, "Enhancing the Controlled Disclosure of Sensitive Information," *Proc. European Symp. Research in Computer Security,* pp. 290–303, 1996.
[11] S. Mazumdar, D. Stemple, and T. Sheard, "Resolving the Tension between Integrity and Security Using a Theorem Prover," *Proc. ACM Int'l Conf. Management of Data,* pp. 233–242, 1988.
[12] C. Meadows, "Extending the Brewer-Nash Model to a Multilevel Context," *Proc. IEEE Symp. Security and Privacy,* pp. 95–102, 1990.
[13] M. Morgenstern, "Controlling Logical Inference in Multilevel Database Systems," *Proc. IEEE Symp. Security and Privacy,* pp. 245–255, 1988.
[14] G.W. Smith, "Modeling Security-Relevant Data Semantics," *Proc. IEEE Symp. Research in Security and Privacy,* pp. 384–391, 1990.
[15] P.D. Stachour and B. Thuraisingham, "Design of LDV: A Multilevel Secure Relational Database Management System," *IEEE Trans. Knowledge and Data Eng.,* vol. 2, no. 2, pp. 190–209, June 1990.
[16] T. Su and G. Ozsoyoglu, "Inference in MLS Database Systems," *IEEE Trans. Knowledge and Data Eng.,* vol. 3, no. 4, pp. 474–485, Dec. 1991.
[17] T.H. Hinke, H.S. Delugach, and A. Chandrasekhar, "A Fast Algorithm for Detecting Second Paths in Database Inference Analysis," *J. Computer Security,* vol. 3, nos. 2 and 3, pp. 147–168, 1995.
[18] B.M. Thuraisingham, "Security Checking in Relational Database Management Systems Augmented with Inference Engines," *Computers and Security,* vol. 6, pp. 479–492, 1987.
[19] J.D. Ullman, *Principles of Database and Knowledge-Base Systems,* vols. 1 and 2. Rockville Md.: Computer Science Press, 1988.

Alexander Brodsky received the BSc degree in mathematics and computer science and the MS and PhD degrees in computer science in 1991 and 1983, respectively, from the Hebrew University of Jerusalem. He is currently an associate professor of information and software engineering at George Mason University (GMU), where he began in 1993 from the IBM T.J. Watson Research Center. At GMU, he leads a research group in constraint databases and programming, funded by the US National Science Foundation (NSF), Office of Naval Research, and NASA. His main research interests and publications combine the area of databases (including integration of constraint, spatial, and temporal data within constraint databases, database optimization, algebras, algorithms, indexing and filtering, and models and languages) and the area of constraint programming (including algorithms for arithmetic and finite domain constraints, mathematical programming, and logical f systems with decidable constraint theories). He is a recipient of the NSF research initiation award and of the NSF CAREER award. He served on the program committees of numerous computer science conferences, as an invited member of the ACM Strategic Directions in Computing Research Group in Constraint Programming and in the group on Electronic Commerce and Digital Libraries, and coedited an LNCS volume on Constraint Databases and Applications. He served as conference chair of the Fifth International Conference on Principles and Practice of Constraint Programming (CP '99).

Csilla Farkas received the BS degree in geology from Eotvos Lorand University, the BS degree in computer science from SZAMALK, Hungary, the MS degree in computer science and the PhD degree in information technology from George Mason University (GMU), Fairfax, Virginia, respectively. She is an assistant professor at the Department of Computer Science and Engineering at the University of South Carolina, Columbia. Her research interests include database management systems, information security, data mining, and scientific databases.



Sushil Jajodia received the PhD degree from the University of Oregon, Eugene. He is BDM Chair Professor and chairman of the Department of Information and Software Engineering and director of the Center for Secure Information Systems at George Mason University (GMU), Fairfax, Virginia. He joined GMU after serving as the director of the Database and Expert Systems Program at the National Science Foundation. Before that, he was the head of the Database and Distributed Systems Section in the Computer Science and Systems Branch at the Naval Research Laboratory, Washington. He has also been a visiting professor at the University of Milan, Italy, and at the Isaac Newton Institute for Mathematical Sciences, Cambridge University, England. His research interests include information security, temporal databases, and replicated databases. He has authored three books, including *Granularities in Databases, Data Mining, and Temporal Reasoning* (Springer-Verlag, 2000) and *Information Hiding: Stenography and Watermarking—Attacks and Countermeasures* (Kluwer, 2000), editied 17 books, and published more than 200 technical papers in refereed journals and conference proceedings. He received the 1996 Kristian Beckman award from IFIP TC 11 for his contributions to the discipline of Information Security and the 2000 Outstanding Research Faculty award from GMU's School of Information Technology and Engineering.

Dr. Jajodia has served in different capacities for various journals and conferences. He is the founding editor-in-chief of the *Journal of Computer Security*. He is on the editorial boards of *IEEE Concurrency*, *ACM Transactions on Information and Systems Security*, and the *International Journal of Cooperative Information Systems*. He is the consulting editor of the *Kluwer International Series on Advances in Information Security*. He serves as the program chair of the 2000 ACM Conference on Computer and Communications Security (CCS '00) and the 2001 International Conference on Conceptual Modeling (ER2001). He also serves on the Board of Directors of the National Colloquim for Information Security Education (NCISSE). He has been named a Golden Core member for his service to the IEEE Computer Society. He is a past chairman of the IEEE Computer Society Technical Committee on Data Engineering. He is a senior member of the IEEE and a member of the IEEE Computer Society and the ACM. The URL for his web page is http://isse.gmu.edu/csis/faculty/jajodia.html.