Assessing Query Privileges via Safe and Efficient Permission Composition

Sabrina De Capitani di Vimercati
DTI - Università di Milano
26013 Crema - Italy
decapita@dti.unimi.it

Sara Foresti
DTI - Università di Milano
26013 Crema - Italy
foresti@dti.unimi.it

Sushil Jajodia
CSIS - George Mason University
Fairfax, VA 22030-4444
jajodia@gmu.edu

Stefano Paraboschi
DIIMM - Università di Bergamo
24044 Dalmine - Italy
parabosc@unibg.it

Pierangela Samarati
DTI - Università di Milano
26013 Crema - Italy
samarati@dti.unimi.it

ABSTRACT

We propose an approach for the selective enforcement of access control restrictions in, possibly distributed, large data collections based on two basic concepts: *i)* flexible authorizations identify, in a declarative way, the data that can be released, and *ii)* queries are checked for execution not with respect to individual authorizations but rather evaluating whether the information release they (directly or indirectly) entail is allowed by the authorizations. Our solution is based on the definition of query profiles capturing the information content of a query and builds on a graph-based modeling of database schema, authorizations, and queries. Access control is then effectively modeled and efficiently executed in terms of graph coloring and composition and on traversal of graph paths. We then provide a polynomial composition algorithm for determining if a query is authorized.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; H.2.4 [**Database Management**]: Systems—*Relational databases*; H.2.4 [**Database Management**]: Systems—*Query processing*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Management

Keywords

Access control, authorization composition

1. INTRODUCTION

A crucial goal of research in computer security is the design of effective policies able to flexibly represent, at an abstract level, access privileges. Flexible and abstract policies facilitate the work of the security designer and reduce errors in the specification, decreasing direct and indirect costs of a security system. The deployment of flexible and expressive policy representations has often been hampered by the absence of efficient mechanisms for their evaluation. A critical success factor for an access control model is to be able to scale well to the size of modern information systems, with large numbers of resources that must be protected.

In this paper, we consider the protection of large data collections, possibly stored at different sites, which different parties (users, applications, servers) may be authorized to access selectively. In other words, different parties may be allowed to see different portions of the data [7]. The explicit reference of our model to relational databases is justified by the fact that relational databases have a central role in the management of large data collections today and promise to have an increasing role in future applications. We observe, however, that our solution is directly applicable to other data models and can be easily adapted to other scenarios, like web service integration or XML querying.

Current approaches for the specification and enforcement of authorizations in relational databases claim flexibility and expressiveness because of the possibilities of referring to views. Users can be given access to a specific portion of the data by the definition of the corresponding view (in the database schema) and the consequent granting of the authorization on the view to the user. It is then responsibility of the user to query the view itself. Queries on a table (base relation or view) are controlled with respect to authorizations specified on the tables themselves and are permitted only if the base tables are explicitly authorized. When the diversity of users and possible views is considerable and dynamic, such an approach clearly results limiting, because: *i)* it potentially requires to explicitly define a view for each possible access need and *ii)* it imposes on the user/application the burden of knowing and directly querying the view. The evaluation of query compliance in terms of existing authorization views has been considered in [14, 16, 17, 18].

We propose an expressive, flexible, and powerful, yet simple approach for the specification and enforcement of authorizations that overcomes such limitations. Our authorizations express privileges not on specific existing views but on stable components of the database schema, exploiting

CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

both *i)* relations and *ii)* joins between them; thus effectively identifying the specific portion of the data whose access is being authorized. Another important aspect of our approach is that we do not limit ourselves to a simple query-authorization control but allow a query to be executed whenever the information carried by the query (either directly in the result or indirectly due to the dependence of the result with other data not explicitly released) is legitimate according to the specified authorizations. This is an important paradigm shift with respect to current solutions, departing from the need of specifying views to identify the portion of the data to be authorized but explicitly supporting such a specification in the authorizations themselves.

The remainder of the paper is organized as follows. Section 2 illustrates the preliminary concepts of relational databases. Section 3 presents the basic components of our authorization model. Section 4 illustrates a graph-based representation of the components of our authorization model (database schema, queries, and permissions) which is then exploited for assessing if a query can be safely executed. Section 5 defines when a query is authorized (and therefore can be safely executed) by either an individual permission or a combination of permissions. Section 6 illustrates an algorithm to determine if a query can be considered authorized by a set of permissions avoiding computation of all possible permission compositions. Section 7 discusses related work. Finally, Section 8 draws some conclusions.

2. PRELIMINARY CONCEPTS

We consider a reference scenario where data are stored in a relational database against which *subjects*, which can be either users, servers, or processes, can execute queries. In the following, the set of subjects and relations in the reference scenario is denoted by $\mathcal{S}$ and $\mathcal{R}$, respectively. At the schema level, a relation is characterized by a name r and a set $\{a_1, \ldots, a_n\}$ of attributes and is denoted by $r(a_1,\ldots,a_n)$; $r.*$ refers to the set $\{a_1, \ldots, a_n\}$ of attributes in the relation. At the instance level, a relation $r(a_1,\ldots,a_n)$ is a set of *tuples* over set $\{a_1,\ldots,a_n\}$, where each tuple t is a mapping from attributes to *values* in their domain. For simplicity, when clear from the context, we will use r to denote either the schema or the instance of relation r. Given an attribute a and a set A of attributes, $t[a]$ denotes the value of attribute a in t and $t[A]$ the sub-tuple composed of all values of attributes in A.

Each relation has a *primary key* which is the attribute, or the set of attributes, that uniquely identifies each tuple in the relation. Given a relation r_i, $K_i \subseteq r_i.*$ denotes r_i's primary key attributes. Primary key attributes cannot assume NULL values and two tuples in the relation cannot assume the same value for the primary key. This latter condition implies the existence of a *functional dependency* between the primary key of a relation and any other attribute in the relation. Given a relation $r(a_1,\ldots,a_n)$ and two non-empty subsets A_i and A_j of the attributes $\{a_1,\ldots,a_n\}$, there is a *functional dependency* on r between A_i and A_j if for each pair of tuples t_l, t_m of r with the same values on attributes in A_i, t_l and t_m have also the same values on attributes in A_j. Without loss of generality, we assume that only functional dependencies given by the primary key hold in the relations. This assumption does not limit the applicability of our solution since it is similar to the common database schema requirement that the relations satisfy the *Boyce-Codd Normal Form* (BCNF), to avoid redundancies and undesirable side-effects during update operations, and it is usually achievable using adequate decomposition procedures [3].

$\mathcal{R}$	Employee(ssn,job,salary) Patient(ssn,dob,race) Treatment(ssn,iddoc,type,cost,duration) Doctor(iddoc,name,specialty)
$\mathcal{I}$	⟨Treatment.ssn,Patient.ssn⟩ ⟨Treatment.iddoc,Doctor.iddoc⟩
$\mathcal{J}$	⟨Employee.ssn,Patient.ssn⟩

Figure 1: An example of relations, referential integrity constraints, and joins

Figure 1 illustrates an example of a set $\mathcal{R}$ of four relation schemas. Primary key attributes appear underlined.

The primary key K_i of a relation r_i can also appear in, or more precisely, be *referenced* by, a set of attributes F_j, in another relation r_j. In such a case, F_j, called *foreign key*, can assume only values that appear for K_i in the instance of r_i. This is formalized by the definition of *referential integrity* constraint which, assuming for simplicity absence of NULL values for the foreign key, is as follows.

DEFINITION 2.1 (REFERENTIAL INTEGRITY). *Given two relation schemas r_i,$r_j \in \mathcal{R}$ and a set of attributes $F_j \subseteq r_j.*$, there is a* referential integrity *constraint from F_j to K_i if and only if for any possible instance r_i' of r_i and r_j' of r_j, $\forall t_j \in r_j'$ there exists a tuple $t_i \in r_i'$ such that $t_j[F_j] = t_i[K_i]$.*

In the following, we use $\langle F_j, K_i \rangle$ to denote a referential integrity constraint between F_j and K_i. Also, $\mathcal{I}$ denotes the set of all referential integrity constraints defined over $\mathcal{R}$.

For instance, with respect to the relation schemas in Figure 1, there are two referential integrity constraints: ⟨`Treatment.ssn`,`Patient.ssn`⟩, implying that treatments can only be given to patients (values appearing for `ssn` in `Treatment` can be only values appearing for `ssn` in `Patient`), and ⟨`Treatment.iddoc`,`Doctor.iddoc`⟩, implying that treatments can only be prescribed by doctors (values appearing for `iddoc` in `Treatment` can be only values appearing for `iddoc` in `Doctor`).

Information in different relations can be combined by using the join operation, which allows the combination of tuples belonging to different relations imposing conditions on how tuples can be combined. For simplicity of exposition, we assume that attributes that can be joined appear with the same name in the different relations, and consider then all joins to be *natural joins*, that is, joins whose conditions are conjunctions of equality conditions that compare the value of two attributes with the same name. We denote a conjunction of equality conditions with a pair $\langle A_l, A_r \rangle$, where A_l (A_r, resp.) is the list of attributes of the left (right, resp.) operand of the join. Note that while possible joins obviously include all referential integrity constraints, other joins are possible; in the following we denote with $\mathcal{J}$ the set of pairs representing the equality conditions of such additional joins. As an example, with respect to the relations in Figure 1, `Employee` and `Patient` can be joined over attribute `ssn` (retrieving all people that are both employees and patients). Like the set of relations and the referential integrity constraints, possible joins are also specified at the time of database design [3].

We assume all attributes in the different relations to have distinct names, apart from attributes that can be joined, which appear instead with the same name. The intuitive rationale behind such a homonymity is that attributes that can be joined actually represent the same concept of the real world. For instance, `ssn` denotes social security numbers of people, who can then appear, for example, as patients or employees. We adopt the usual dot notation, when necessary, to distinguish the attribute in a specific relation. For instance, `Employee.ssn` denotes the social security numbers of employees and `Patient.ssn` denotes the social security numbers of patients.

Different join operations can also be used to combine tuples belonging to more than two relations. The following definition introduces a *join path* as a sequence of natural join conditions.

DEFINITION 2.2 (JOIN PATH). *A* join path *over a sequence of relation schemas $r_1,\ldots,r_n$ is a sequence of $n-1$ joins $J_1,\ldots,J_n$ such that $\forall i=1,\ldots,n-1$, $J_i=\langle A_{li},A_{ri}\rangle \in (\mathcal{I}\cup\mathcal{J})$ and A_{li} are attributes of a relation appearing in a join J_k, with $k<i$.*

While noting that the permission model we propose in the next section can be applied to any schema, in this paper we assume that the schema is *acyclic* and *lossless* [1, 2]. Acyclicity implies that the join path over any subset of the relations $\{r_1,\ldots,r_n\}$ in the schema, denoted $joinpath(\{r_1,\ldots,r_n\})$, is unique. Acyclicity rules out schemas that present recursion or multiple independent join conditions among the same relations. Acyclicity can be immediately evaluated on the schema graph (see Section 4), considering arcs without orientation. Losslessness of the schema guarantees that joins among relations produce only correct information (according to the real world). Intuitively, two relations produce a *lossless join* if the join among them does not produce spurious tuples. Losslessness can be evaluated by means of attribute intersections and functional dependencies (see Section 4). Acyclicity and losslessness assumptions are often used in relational databases, because they permit the realization of simple and efficient procedures on the data, at the same time capturing the requirements of most real-word situations [1].

We consider select-from-where queries of the form: "SELECT *Attributes* FROM *Joined relations* WHERE *Conditions*", retrieving a set of *Attributes* from the tuples in the relation resulting from joining the specified *Joined relations* that satisfy the given *Conditions*.

EXAMPLE 2.1. *Consider the relations in Figure 1. Query:*

```
SELECT Employee.ssn, salary
FROM   Employee JOIN Patient
       ON Employee.ssn=Patient.ssn
       JOIN Treatment ON Treatment.ssn=Patient.ssn
WHERE cost> 250
```

retrieves the `ssn` (*of employee*) *and* `salary` *for all hospitalized employees under some treatment whose* `cost` *is greater than $250.*

3. SECURITY MODEL

We first present our simple, while expressive, permissions, regulating how data can be released to each subject. We then characterize the information content of queries by introducing the concept of query profile.

p_1: [(ssn,dob,race),(Patient)] →Alice
p_2: [(ssn,type,cost,duration),(Treatment)] →Alice
p_3: [(race,specialty),(Treatment,Patient,Doctor)] →Alice
p_4: [(ssn,job,salary),(Employee)] →Alice
p_5: [(name),(Treatment,Doctor)] →Alice

Figure 2: Examples of permissions

3.1 Permissions

Different subjects in the system may be authorized to view different portions of the whole database content. We consider permissions in a simple, yet powerful form, specifying visibility *permissions* for subjects to view certain schema components. Formally, permissions are defined as follow.

DEFINITION 3.1 (PERMISSION). *A* permission p *is a rule of the form* [A, R]→*S where:*

- A *is a set of attributes, belonging to one or more relations, whose release is being authorized;*
- R *is a set of relations such that for every attribute in* A *there is a relation including it;*
- S *is a subject in* $\mathcal{S}$.

Permission $[A, R]\rightarrow S$ states that subject S can view the sub-tuples over the set of attributes A belonging to the join among relations R (on conditions $joinpath(R)$).

Note that, according to the definition, only attribute names (without indication of the relation) appear in the first component of the permission, whereas the relation (or relations) to which the attribute belongs is specified in the second component. This occurs even when the attribute appears in more than one relation (specified in R), consistently with the semantics that the two occurrences represent the same entity in the real world.

EXAMPLE 3.1. *Figure 2 illustrates some permissions on the relations in Figure 1 that give* `Alice` *the visibility of:*

- *ssn, date of birth, and race of all patients (p_1);*
- *ssn of treated patients, together with type, cost, and duration of their treatments (p_2);*
- *race of patients and specialty of their caring doctors (p_3);*
- *ssn, job, and salary of all employees (p_4);*
- *name of doctors who have prescribed some treatment (p_5).*

Note that the *presence of a relation* (and therefore the enforcement of the corresponding join conditions) in a permission may *decrease the set of tuples* that are made visible (to only those tuples that participate in the join). However, such an elimination of tuples does not correspond to less information, rather it *adds information* on the fact that the visible tuples actually join with (i.e., have values appearing in) other tuples of the joined relations. For instance, permission p_5 while restricting the set of doctor's names visible to `Alice` to only the names of the doctors who have prescribed treatments, it allows `Alice` to see that such doctors have

prescribed treatments (i.e., they appear in relation `Treatment`).

The only case where including an additional relation in the permission does not influence the result, and therefore does not imply an indirect information disclosure, occurs when the additional relations are reachable via referential integrity constraints (from the foreign key to the primary key it references). For instance, permissions p_2 in Figure 2 and a permission with the same first component as p_2 and having (`Treatment`,`Patient`,`Doctor`) as a second component, are completely equivalent as they permit (direct or indirect) release of exactly the same information. Indeed, given the existing referential integrity constraints (see $\mathcal{I}$ in Figure 1), all `ssn` and all `iddoc` appearing in `Treatment` also appear in `Patient` and `Doctor`, respectively. The added joins are therefore ineffective.

Note also that the set R of relations in a permission may also include relations whose attributes do not appear in the set A of attributes. This may be due to either:

- *connectivity constraints*, where these relations are needed to build the association among attributes of other relations. For instance, in permission p_3, relation `Treatment` establishes the associations between attributes of patients and of their caring doctors, but none of its attributes is released.
- *instance-based restrictions*, where these relations are needed to restrict the attributes to be released to only those values appearing in tuples that can be associated with such relations. For instance, in permission p_5 as just commented.

3.2 Queries

Permissions restrict the data (view) that can be released to a subject. To determine whether a subject can execute a query, we need then to check the subject's permissions against the query to determine whether the query execution would allow the subject to view information that s/he should not be allowed to view (according to the specified permissions). To this purpose, we introduce the concept of *query profile*, which captures the information content of the query, as follows.

Definition 3.2 (Query profile). *Given a select-from-where query q, the* query profile *of q is a pair* $[\mathrm{A}_q,\mathrm{R}_q]$, *where* A_q *is the set of attributes appearing in the select and/or in the where clauses and* R_q *is the set of relations appearing in the from clause.*

Intuitively, the semantics of a query profile $[A_q,R_q]$ is that the query result bears information (and therefore needs permission for execution) on attributes A_q appearing in the relation obtained by joining the relations R_q. As an example, the profile of the query in Example 2.1 is [(`ssn`,`salary`,`cost`), (`Employee`,`Patient`,`Treatment`)].

The reason why both, *i)* the attributes being returned as result (i.e., the attributes in the SELECT clause) and *ii)* the attributes on which the query imposes conditions (i.e., the attributes in the WHERE clause), appear in the profile reflects the fact that the query result returns indeed information on both (or, equivalently, the subject needs permissions to view both for query execution).

Note also that, like for permissions, only attribute names (without indication of the relation) appear in the first component of the query profile, where the relation (or relations) to which the attributes belong is specified in the second component. Indeed, if an attribute belongs to more than one relation (and therefore participates in the join), the common values of such an attribute in all relations are released by the query, regardless of the specific relation mentioned in the SELECT clause, which is needed for disambiguating attribute names. The consideration of the attribute names allows us to conveniently capture this aspect regardless of the specific way in which the query has been written. For instance, with respect to the query in Example 2.1, the set of social security numbers released by the query is the intersection of the set of `ssn` values of patients and employees, as captured in the profile. As a matter of fact, a query equal to the query in Example 2.1 but releasing `Patient.ssn` instead of `Employee.ssn`, while slightly different in the syntax, would carry exactly the same information content and, consequently, would have the same profile.

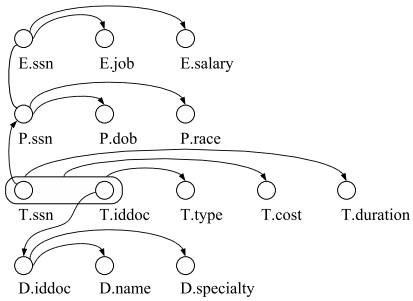

Figure 3: Schema graph for the relations in Figure 1

Our goal in this paper is to evaluate a query issued by a subject against her permissions and to determine if the query can be executed. In the following sections, we first introduce a graph-based model for representing the database schema, permissions, and queries. We then present an approach based on such a graph-based model for the evaluation of permissions.

4. GRAPH-BASED MODEL

We model database schema, permissions, and queries via mixed graphs, that is, graphs with both undirected and directed arcs.

The *schema graph* of a set $\mathcal{R}$ of relations is a mixed graph whose nodes correspond to the different attributes of the relations and whose arcs correspond to: non-oriented arcs representing the possible joins ($\mathcal{J}$); and oriented arcs representing the referential integrity constraints ($\mathcal{I}$) and the functional dependencies between the primary key of a relation and its non-key attributes. Attributes appearing with the same name in more than one relation appear as different nodes. To disambiguate, nodes are identified with the usual dot notation by the pair *relation.attribute*. This is formalized by the following definition.

Definition 4.1 (Schema graph). *Given a set $\mathcal{R}$ of relations, a set $\mathcal{I}$ of referential integrity constraints over $\mathcal{R}$, and a set $\mathcal{J}$ of join conditions over $\mathcal{R}$, a* schema graph *is a graph $G(\mathcal{N},\mathcal{E})$ where:*

- $\mathcal{N} = \{r_i.* \mid r_i \in \mathcal{R}\}$
- $\mathcal{E} = \mathcal{J} \cup \mathcal{I} \cup \{(r_i.K, r_i.a) \mid r_i \in \mathcal{R} \wedge a \notin K\}$

Figure 3 represents the schema graph corresponding to the set of relations, referential integrity constraints, and join conditions in Figure 1 (for simplicity, we only report the initials of the relations).

In the following, we refer our discussion to permissions and queries of a specific subject and omit the subject component of permissions in the formalization. Permissions and queries correspond to views over the set $\mathcal{R}$ of relations and, as already discussed, are characterized by a pair $[A,R]$, corresponding to attributes/relations appearing in the permissions and in the query profiles, respectively.

In the characterization of views, we take into consideration the fact that referential integrity constraints can be used to extend the relations in R to include all relations reachable from the ones appearing in R by following referential integrity connections from a foreign key to the referenced primary key and without adding information (see Section 3.1). We can then include such relations in the set R. Given a set R of relations, R^* denotes the relations obtained by closing R with respect to referential integrity constraints. For instance, with respect to the schema graph in Figure 3, the closure of R={`Treatment`} is R^*={`Treatment`, `Patient`, `Doctor`}.

Given a query/permission, we graphically represent the view entailed by it as a *view graph* obtained by coloring the original schema graph with three colors: *black* for information that the view carries (i.e., it explicitly contains or indirectly conveys); *white* for all the non-black attributes belonging to relations in R^* and the arcs connecting them to the primary key; and *clear* for any other attribute or arc. Intuitively, clear nodes/arcs are attributes/arcs belonging to the original graph that are ineffective with respect to the evaluation and composition of permissions. The reason for maintaining them in the view graphs is so that every query/permission is a coloring (in contrast to a subgraph) of the schema graph. View graph is formally defined as follows.

Definition 4.2 (View graph). *Given a set $\mathcal{R}$ of relations characterized by schema graph $G(\mathcal{N},\mathcal{E})$ and a permission/query profile V = [A,R] on it, the* view graph *of V over G is a graph $G_V(\mathcal{N},\mathcal{E},\lambda_V)$, where $\lambda_V : \{\mathcal{N} \cup \mathcal{E}\} \rightarrow$ {black,white,clear} is a coloring function defined as follows.*

$$\lambda_V(n)=\begin{cases} black, & n=r.a,\ r \in R^* \wedge a \in \mathrm{A} \\ white, & n=r.a,\ r \in R^* \wedge a \notin \mathrm{A} \\ clear, & otherwise \end{cases}$$

$$\lambda_V(n_i,n_j)=\begin{cases} black, & (n_i,n_j) \in \mathrm{joinpath}(R^*) \vee \\ & (n_i=r.K,\ n_j=r.a,\ r \in R^*, \\ & (a \in \mathrm{A} \vee r.a \ appears\ in\ \mathrm{joinpath}(R^*))) \\ white, & n_i=r.K,\ n_j=r.a,\ r \in R^*, \\ & \neg(a \in \mathrm{A} \vee r.a \ appears\ in\ \mathrm{joinpath}(R^*))) \\ clear, & otherwise \end{cases}$$

According to this definition, a node is colored as: *black* if it appears in A, *white* if it is not black and it belongs to a relation appearing in R^*, and *clear* otherwise. An arc is colored: *black* if either it belongs to *joinpath*(R^*) or it is an arc going from the key of a relation in R^* to an attribute which either belongs to A or appears in *joinpath*(R^*); *white* if it is an arc from the key of a relation in R^* to one of its attributes which neither belongs to A nor appears in *joinpath*(R^*); *clear* otherwise.

COLOR_GRAPH(G,[A,R])
$\mathcal{N}_V := \mathcal{N}$
$\mathcal{E}_V := \mathcal{E}$
for each $n \in \mathcal{N}_V$ **do** $\lambda_V(n)$:= clear
for each $(n_i,n_j) \in \mathcal{E}_V$ **do** $\lambda_V(n_i,n_j)$:= clear
for each $r \in R^*$ **do**
 for each $a \in r.*$ **do** /* color nodes */
  if $a \in A$ **then** $\lambda_V(r.a)$:= black
  else $\lambda_V(r.a)$:= white
for each $(n_i,n_j) \in joinpath(R^*)$ **do** /* color the join path */
 $\lambda_V(n_i,n_j)$:= black
for each $(n_i,n_j) \in \{(n_i,n_j) | \exists r \in R^*,\ n_i=r.K \wedge n_j \subseteq r.*\}$ **do**
 if $\lambda_V(n_j)$=black $\vee$ n_j appears in *joinpath*(R^*) **then**
  $\lambda_V(n_i,n_j)$:= black
 else $\lambda_V(n_i,n_j)$:= white
$G_V := (\mathcal{N}_V,\mathcal{E}_V,\lambda_V)$
return(G_V)

Figure 4: Function for coloring a view graph

Figure 4 illustrates the **Color_Graph** function that given the schema graph G and a pair $[A,R]$ denoting either a permission or a query profile, implements Definition 4.2 and returns the corresponding view graph. **Color_Graph**, whose interpretation is immediate, starts by assigning a clear color to all nodes and arcs and proceeds coloring black and white arcs and nodes as prescribed by the definition.

Figure 5 reports the view graphs corresponding to the permissions in Figure 2. Here, black nodes and arcs are represented by filled nodes and bold lines, white nodes and arcs are represented by continuous nodes and lines, and clear nodes and arcs are represented by dotted nodes and lines. Figure 6 reports some examples of queries over the schema of Figure 3 together with their corresponding view graphs.

Before closing this section, we introduce two dominance relationships between view graphs that will be used in the remainder of the paper.

Definition 4.3 ($\preceq_N$, $\preceq_{NE}$). *Given a schema graph $G(\mathcal{N},\mathcal{E})$, and two view graphs $G_{V_i}(\mathcal{N},\mathcal{E},\lambda_{V_i})$ and $G_{V_j}(\mathcal{N},\mathcal{E},\lambda_{V_j})$ over G, the following dominance relationships are defined:*

- $G_{V_i} \preceq_N G_{V_j}$, *when* $\forall n \in \mathcal{N}$ *and* $\forall (n_h,n_k) \in (\mathcal{J} \cup \mathcal{I})$*:*
  - $\lambda_{V_i}(n) = black \Longrightarrow \lambda_{V_j}(n)$=*black, and*
  - $\lambda_{G_i}(n_h,n_k) = black \Longleftrightarrow \lambda_{G_j}(n_h,n_k) = black$.
- $G_{V_i} \preceq_{NE} G_{V_j}$, *when* $\forall n \in \mathcal{N}$ *and* $\forall (n_h,n_k) \in \mathcal{E}$*:*
  - $\lambda_{V_i}(n) = black \Longrightarrow \lambda_{V_j}(n)$=*black, and*
  - $\lambda_{G_i}(n_h,n_k) = black \Longrightarrow \lambda_{G_j}(n_h,n_k) = black$.

According to this definition, given two graphs G_{V_i} and G_{V_j} on the same database schema, $G_{V_i} \preceq_N G_{V_j}$ if they have exactly the same black referential integrity and join arcs and the black nodes of G_{V_i} are a subset of the black nodes of G_{V_j}. $G_{V_i} \preceq_{NE} G_{V_j}$ if the black arcs and nodes of G_{V_i} are a subset of the black arcs and nodes of G_{V_j}. For instance, with reference to the view graphs in Figures 5 and 6, it is easy to see that: $G_{p_1} \preceq_{NE} G_{q_2}$ and that $G_{p_3} \preceq_N G_{q_3}$.

5. QUERY ANALYSIS

To evaluate a query requested by a subject against her permissions and to determine if the query can be executed, we implement the following intuitive concept.

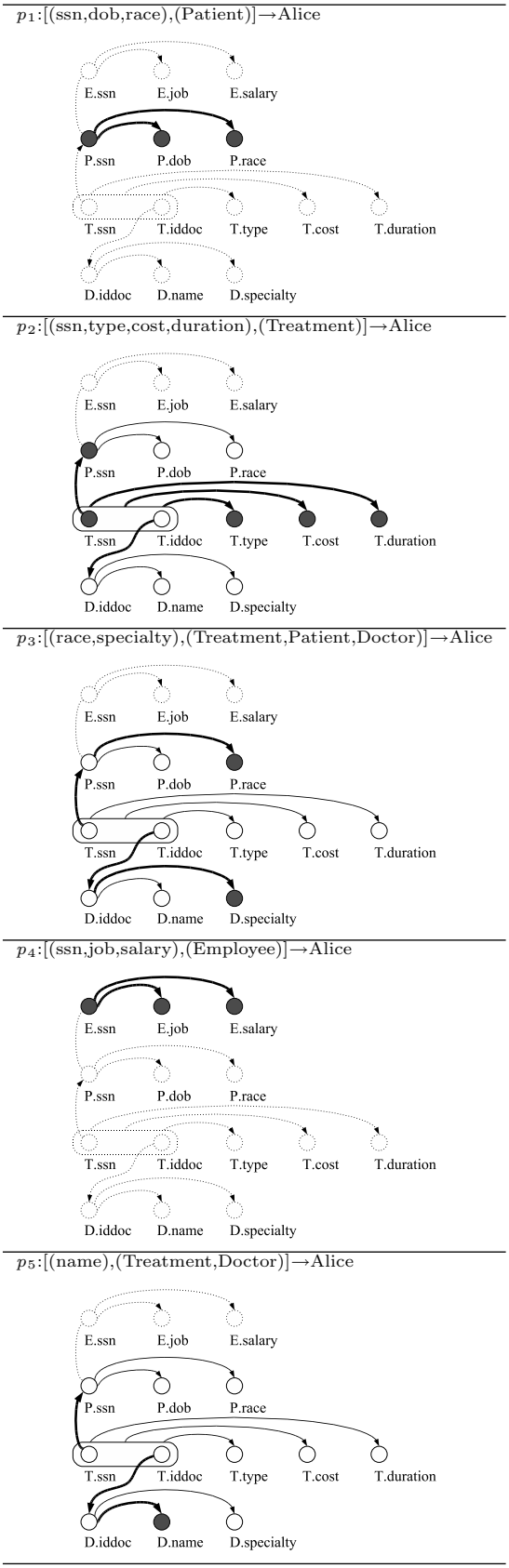

Figure 5: Examples of permissions and their view graphs

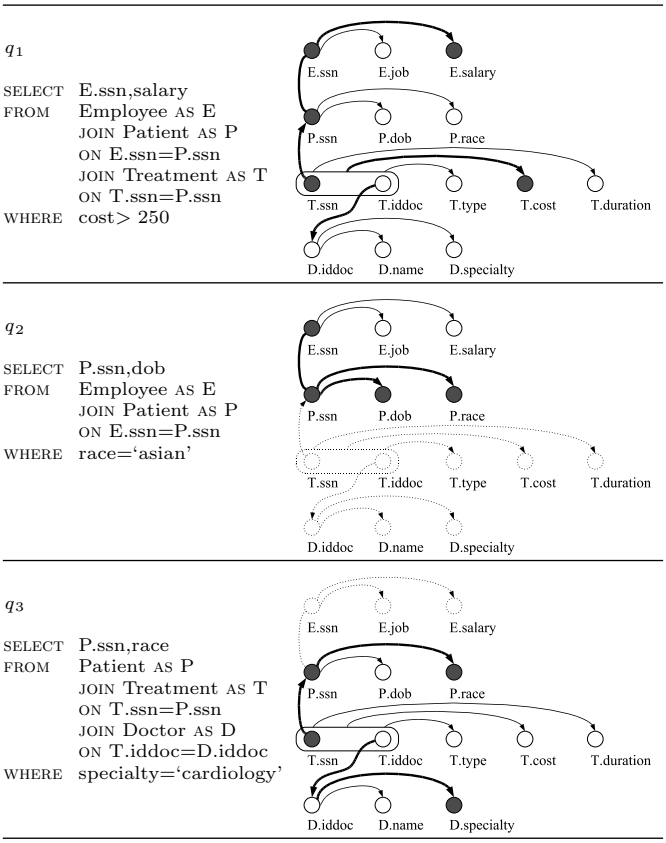

Figure 6: Examples of queries and their view graphs

PRINCIPLE 5.1. *A query can be executed if the subject has permissions to view the information content carried by the query.*

We first discuss when a permission authorizes a query execution. We will then address permission composition.

5.1 Authorizing permissions

Intuitively, a permission authorizes the execution of a query if and only if the information (directly or indirectly) released by the query is a subset of the information that the permission authorizes to view.

Note that this is different from saying that the query result should contain only data that are a subset of the data authorized by the permission, as this denotes only the information directly released. A correct enforcement should also ensure that no indirect release occurs. There are two main sources of indirect release: *i)* the presence, in the query, of conditions on attributes that are not returned (i.e., attributes that appear in the WHERE clause but do not appear in the SELECT clause); and *ii)* the presence of join conditions restricting the tuples returned by the query. The first aspect is easily taken into consideration as it is already captured by the inclusion, in the query profile (Definition 3.2), of all the attributes accessed by the query. To illustrate the problem of the second aspect, consider permission p_1 in Figure 5, which allows `Alice` to view the complete information in `Patient`, and therefore the whole tuples representing all patients. Permission p_1 by itself is then sufficient to grant `Alice` the ability to execute a query retrieving the data of

all patients ("SELECT P.ssn,dob FROM Patient AS P WHERE race='asian' "). Suppose instead that Alice issues query q_2 in Figure 6. This latter query returns a subset of all the tuples of patients, and therefore only tuples that Alice, according to p_1, is authorized to see. However, permission p_1 is not sufficient for granting such a query since the query result conveys the additional information that the returned tuples refer to patients who are also employees of the given company (information which permission p_1 does not authorize).

As already commented in Section 4, the only case when joins do not add information is when there is a referential integrity constraint among the involved relations. Consider, for example, permission p_2 authorizing the release of different attributes in Treatment. For instance, query "SELECT T.ssn FROM Treatment AS T" is clearly authorized by p_2. Consider then the same query containing, in the FROM clause, also relations Patient and Doctor with the corresponding joins. Despite the presence of the additional joins, such a query does not bear additional information (indirect release) and should therefore be authorized by p_2. As a matter of fact, because of the referential integrity constraints between the involved relations, all ssn's and iddoc's appearing in Treatment also appear in Patient and Doctor, respectively, and therefore the joins do not impose restrictions. The consideration of the peculiar characteristics of joins due to referential integrity constraints is easily taken into account as it is already captured by the coloring, in the view graph, of all the relations reachable from the ones appearing in the query, by following referential integrity constraints (Definition 4.2).

Let us then proceed to formally define when a permission authorizes a query. We start by identifying permissions applicable to a query. Intuitively, a permission applies to a query when it refers to the complete set of tuples requested by the query. Since tuple restriction is due to joins not following the direction from a foreign key to the referenced key in a referential integrity constraint (as commented above), this is equivalent to saying that the permission applies to a query if it does not contain additional joins (apart from those corresponding to referential integrity constraints). This is formalized by the following definition.

DEFINITION 5.1 (APPLICABLE). *A permission* $p = [\mathrm{A}_p,\mathrm{R}_p]$ *is applicable to a query* $q=[\mathrm{A}_q,\mathrm{R}_q]$ *iff* $R_p^* \subseteq R_q^*$.

In terms of view graphs, this definition is equivalent to say that the black and white nodes of G_p should be a subset of the black and white nodes of G_q.

According to the discussion above, a permission authorizes a query if and only if the permission applies to the query and authorizes the release, either direct of indirect, of information in the query. This means that the permission should include (at least) all attributes accessed by the query as well as all the join conditions. In terms of the view graphs, this is equivalent to say that the view graph G_q of the query and the view graph G_p of the permission have exactly the same black arcs and that all nodes that are black in the view graph of the query are also black in the view graph of the permission, that is, $G_q \preceq_N G_p$. This is formally captured by the following definition.

DEFINITION 5.2 (AUTHORIZING PERMISSION). *Given a permission* $p=[\mathrm{A}_p,\mathrm{R}_p]$ *applicable to a query* $q=[\mathrm{A}_q,\mathrm{R}_q]$, p authorizes q *iff* $G_q \preceq_N G_p$.

COMPOSE(G,p_i,p_j)
$p := [A_i \cup A_j, R_i \cup R_j]$
$\mathcal{N}_p := \mathcal{N}$
$\mathcal{E}_p := \mathcal{E}$
for each $n \in \mathcal{N}_p$ **do** $\lambda_V(n) :=$ clear
for each $(n_i,n_j) \in \mathcal{E}_p$ **do** $\lambda_V(n_i,n_j) :=$ clear
for each $n \in \mathcal{N}_p$ **do**
 if $\lambda_{p_i}(n)$=black$\vee\lambda_{p_j}(n)$=black **then** $\lambda_p(n)$=black
 else if $\lambda_{p_i}(n)$=white$\vee\lambda_{p_j}(n)$=white **then**
  $\lambda_p(n)$=white
for each $(n_h,n_k) \in \mathcal{E}_p$ **do**
 if $\lambda_{p_i}(n_h,n_k)$=black$\vee\lambda_{p_j}(n_h,n_k)$=black$\vee$
  ($\lambda_p(n_h)$=black$\wedge\lambda_p(n_k)$=black) **then** $\lambda_p(n_h,n_k)$=black
 else if $\lambda_{p_i}(n_h,n_k)$=white$\vee\lambda_{p_j}(n_h,n_k)$=white **then**
  $\lambda_p(n_h,n_k)$=white
return(p)

Figure 7: Composition of two permissions

As an example, with reference to the authorizations in Figure 5 and query q_2 in Figure 6, the set of authorizations applicable includes p_1 and p_4. However, neither p_1 nor p_4 authorize the query. By contrast, considering query "SELECT P.ssn,dob FROM Patient AS P WHERE race='asian' ", permission p_1 is the only applicable permission that also authorizes the query.

5.2 Composition of permissions

Checking queries against individual permissions is not sufficient for a true enforcement of Principle 5.1. Indeed, it might be that for a query there is no permission that singularly taken authorizes the query, however information released (directly or indirectly) by the query is authorized. As an example, consider permissions p_1 and p_4 in Figure 5 and suppose that Alice requests query q_2 in Figure 6, returning the tuples associated with employees whose ssn appears also in the Patient relation. While neither p_1 nor p_4 authorize the query (as, for each of them, the query has the additional join condition that the permission does not authorize), it is clear that the query does not release any information that Alice is not authorized to see. As a matter of fact, Alice could indeed separately query both relations and then join the two results. In the spirit of Principle 5.1, Alice's query should therefore be authorized. To enforce this, we compose permissions and consider a query authorized if there exists a composition of permissions that authorizes it.

Composition of permissions must however be performed carefully to ensure that composition does not authorize additional queries that were authorized by neither of the original permissions. To illustrate, consider again the permissions in Figure 5 and suppose that Alice issues query q_3. One could think that such a query can be authorized by composing p_3 in Figure 5 (authorizing the release of race's and specialty's) and p_2 (authorizing, in particular, the release of ssn's of patients under treatment). However, such a composition does not authorize the query. Indeed, the query conveys the associations between a patient and her caring doctor, which neither of the individual permissions authorizes and which Alice would not be able to reconstruct by separately exploiting the privileges granted by the two permissions. The problem in this case is that the composition of the two permissions returns more information than that entailed by the two permissions. Therefore, the two permissions should not be composed.

To determine when two permissions can be composed, we exploit one of the foundational results of the theory of joins

for relational databases, expressed by the theorem presented in [2], which states that two relations produce a *lossless join* if and only if at least one of the two relations *functionally depends* from the intersection of their attributes. The relations that are considered in the theorem correspond to generic projections on the set of attributes that characterizes the "universal relation" obtained joining all the relations of our lossless acyclic schema; this means that each permission corresponds to a relation and that the composition of permissions is correct only if the above requirement is satisfied. For instance, consider the previous examples and the permissions in Figure 5. Permissions p_1 and p_4 can be combined because their intersection is represented by attribute `ssn`, which is a key for all the attributes in p_1 (and p_4). Permissions p_1 and p_3 cannot be combined because their intersection is represented by attribute `race`, and neither p_1 nor p_3 functionally depend on it.

The application of this basic result of the theory of joins in our scenario is slightly complicated by the fact that the views corresponding to given permissions may include attributes from different relations. (We note here that intersection of permissions is computed based only on the attribute names, without considering the relation they belong to, since attributes with the same name represent the same real world concept and natural joins impose them to be equal in all the resulting tuples.) Given two permissions p_i=[A_i,R_i] and p_j=[A_j,R_j] their composability depends on the intersection of their visible attributes (i.e., $A_i \cap A_j$) but the functional dependency of the visible attributes of one of the two permissions from the common attributes needs to be evaluated by taking into account also the referential integrity constraints. This concept can be easily captured by analyzing the view graphs G_{p_i} and G_{p_j} corresponding to the two permissions. The basic idea is that there is a dependence between p_i and p_j when there is a black path from nodes corresponding to the attributes that are listed both in A_i and in A_j to all the black nodes in G_{p_i} or in G_{p_j}. This intuitive concept of dependency is formalized as follows.

Definition 5.3 (Dependence). *Given two permissions* p_i=[A_i,R_i] *and* p_j=[A_j,R_j] *with view graphs* $G_{p_i}(\mathcal{N},\mathcal{E},\lambda_{p_i})$ *and* $G_{p_j}(\mathcal{N},\mathcal{E},\lambda_{p_j})$*, respectively, let* B_j *be the set of nodes corresponding to* $\{\mathrm{A}_i \cap \mathrm{A}_j\}$ *in* G_{p_j}*. We say that* p_j depends *on* p_i*, denoted* $p_i \rightarrow p_j$*, iff* $\forall n_j \in \mathcal{N}$ *such that* $\lambda_{p_j}(n_j)$=*black,* $\exists n \in B_j$ *such that there is a path of only directed black arcs from* n *to* n_j *in* G_{p_j}*.*

In the following, notation $p_i \leftrightarrow p_j$ denotes that both $p_i \rightarrow p_j$ and $p_j \rightarrow p_i$ hold. Similarly, $p_i \not\leftrightarrow p_j$ denotes that neither $p_i \rightarrow p_j$ nor $p_j \rightarrow p_i$ hold.

For instance, with reference to the permissions in Figure 5, as already noted, $p_2 \rightarrow p_1$, since common attribute `ssn` is key for the `Patient` relation authorized by p_1, and $p_1 \not\rightarrow p_2$, since the attributes released by p_2 depend on the pair of attributes `ssn` and `iddoc`. We also note that $p_1 \leftrightarrow p_4$, since the `ssn` attribute, common to the two permissions, is the key of both the `Patient` and `Employee` relations. On the contrary, as already pointed out, $p_1 \not\rightarrow p_3$.

If $p_i \rightarrow p_j$ (or $p_j \rightarrow p_i$, resp.), then the two permissions can be *safely composed*, as formally stated by the following definition.

Definition 5.4 (Safe composition). *Given two permissions* p_i=[A_i,R_i] *and* p_j=[A_j,R_j]*,* p_i *and* p_j *can be* safely composed *when* $p_i \rightarrow p_j$*,* $p_j \rightarrow p_i$*, or both.*

For instance, p_1 can be safely composed with p_2, since $p_2 \rightarrow p_1$. Also, since $p_1 \leftrightarrow p_4$, p_1 can be safely composed with p_4.

Similarly to the composition of relations presented in the theory of normal forms for relational databases, the composition of p_i with p_j generates a new permission that combines the viewing privileges of the two, as stated by the following definition.

Definition 5.5 (Composed permission). *Given two permissions* p_i=[A_i,R_i] *and* p_j=[A_j,R_j]*, their composition is the permission* $p_i \otimes p_j$=[$\mathrm{A}_i \cup \mathrm{A}_j$,$\mathrm{R}_i \cup \mathrm{R}_j$]*.*

It is easy to see that the view graph of the resulting composed permission is obtained from the view graphs of the components as follows. A node in $G_{p_i \otimes p_j}$ is: *black* if it is black in either G_{p_i} or G_{p_j}; *white* if it is not black and it is white in either G_{p_i} or G_{p_j}; it is *clear* otherwise. An arc in $G_{p_i \otimes p_j}$ is: *black* if it is black in either G_{p_i} or G_{p_j} or if it is incident on only black nodes in $G_{p_i \otimes p_j}$; *white* if it is not black and is white in either G_{p_i} or G_{p_j}; it is *clear* otherwise. Figure 8 represents the view graphs resulting from a subset of the safe compositions of the privileges in Figure 5, that is, $p_1 \otimes p_2$, $p_1 \otimes p_4$, and $p_1 \otimes p_2 \otimes p_4$.

It might be that the permission $p_i \otimes p_j$ obtained by composing permissions p_i and p_j can be composed with a permission p_k that did not satisfy the composition requirements with p_i nor with p_j. In general, each new permission produces new opportunities for composition that have to be considered. The consideration of all the potential compositions is modeled by the following concept.

Definition 5.6 (Composition closure). *Given a set of permissions* $\mathcal{P}$*, the* closure on composition *of* $\mathcal{P}$*, denoted* $\mathcal{P}^{\otimes}$*, is the set of permissions obtained as a fixpoint by the procedure which repeatedly extends* $\mathcal{P}$ *with all permissions obtained by the safe composition of the permissions in* $\mathcal{P}$*.*

For instance, with reference to the set of permissions in Figure 5, their closure is $\mathcal{P}^{\otimes}$=$\{p_1,\ p_2,\ p_3,\ p_4,\ p_5,\ p_1 \otimes p_2,\ p_1 \otimes p_4,\ p_2 \otimes p_4,\ p_1 \otimes p_2 \otimes p_4\}$.

The closure represents the greatest representation of the permissions available to a subject. This concept permits to identify in a complete way if a specific query is authorized for a subject.

Definition 5.7 (Authorized query). *Given a set* $\mathcal{P}$ *of permissions applicable to a query* q=[A_q,R_q]*,* $\mathcal{P}$ *authorizes* q *iff* $\exists p \in \mathcal{P}^{\otimes}$ *such that* p *authorizes* q (*according to Definition 5.2*)*.*

The computation of the closure on composition of permissions is potentially an expensive procedure. In the next section, we present an efficient algorithm that avoids computing the composition closure while ensuring completeness of control needed to evaluate if a query is authorized.

6. ALGORITHM

Given a set $\mathcal{P}$ of n permissions applicable to a query q, the composition operation does not require to compute all the possible $2^n - 1$ permission compositions, since given two permissions p_i and p_j, if $p_j \rightarrow p_i$ then p_j is subsumed by $p_i \otimes p_j$, and whenever a permission p_k can be composed with p_j, p_k can also be composed with $p_i \otimes p_j$, as stated by the following theorem.

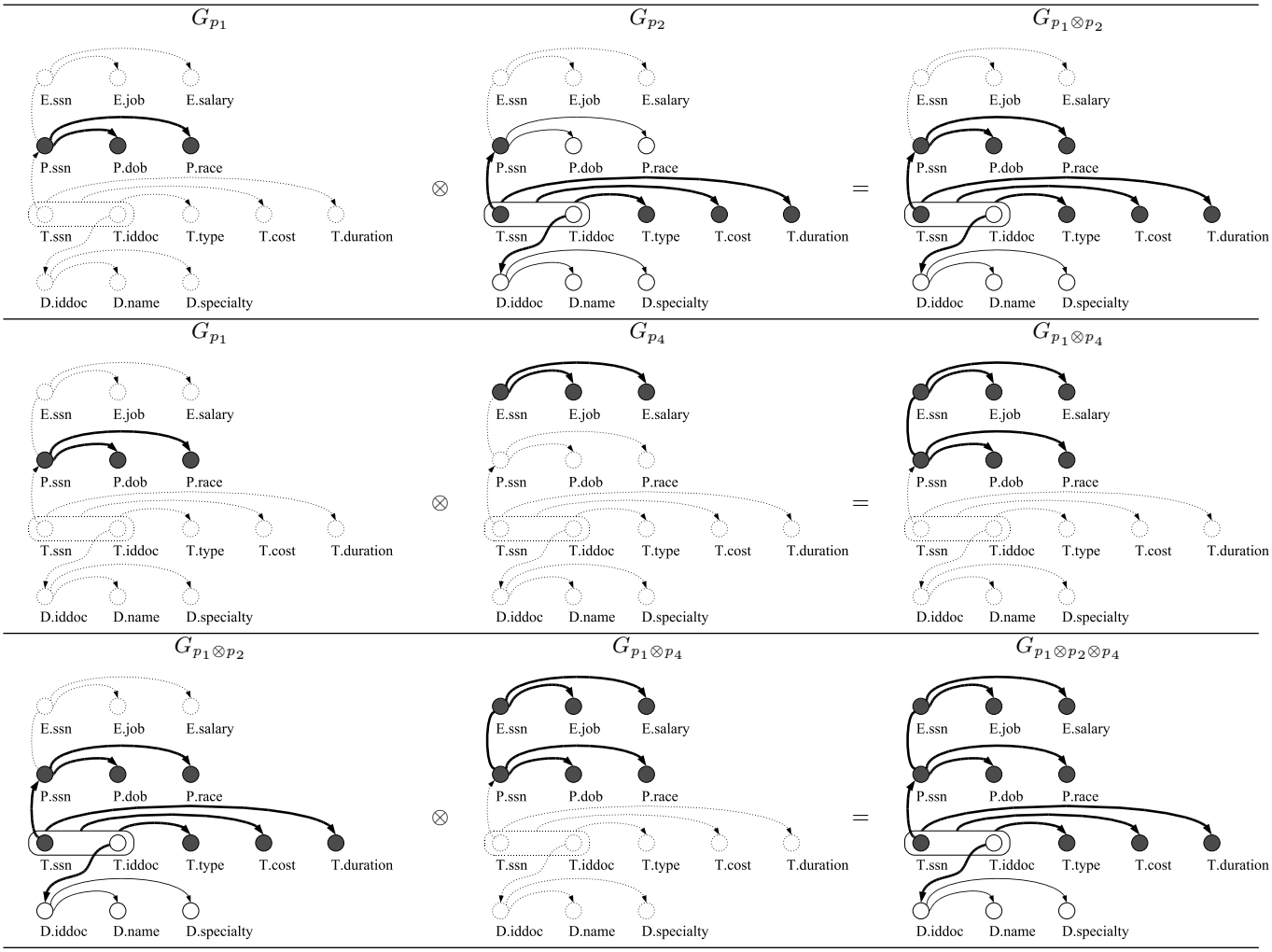

Figure 8: Examples of permission compositions

THEOREM 6.1 (PERMISSION IMPLICATION). *Given two permissions* p_i=[A_i,R_i], p_j=[A_j,R_j] $\in \mathcal{P}$ *such that* $p_j \rightarrow p_i$, $\forall p_k$=[A_k,R_k] $\in \mathcal{P}$*:*

1. $p_j \rightarrow p_k \Rightarrow (p_i \otimes p_j) \rightarrow p_k$;
2. $p_k \rightarrow p_j \Rightarrow p_k \rightarrow (p_i \otimes p_j)$.

This theorem implies that when adding $p_i \otimes p_j$, permission p_j can be removed from set $\mathcal{P}$ without compromising the composition process. It is also easy to see that since the composed permission is again applicable to q, the set of permissions to be composed always contains at most n permissions (i.e., the composed permission substitutes one, or both, of the composing permissions). Function **Authorized** in Figure 9 applies this observation to check whether a query q is authorized. The function takes as input the view graph G_q representing the profile of query q and, on the basis of the set of applicable permissions, returns TRUE or FALSE depending on whether or not the query is authorized.

Initially, **Authorized** determines the set *Applicable* of applicable permissions and checks if one of these permissions dominates ($\preceq_N$) G_q. If this is the case, function **Authorized** returns TRUE. Otherwise, the function starts the composition process that exploits Theorem 6.1. The applicable permissions are first ordered according to a numeric identifier *id*, ranging from 1 to |*Applicable*|, which is associated with each permission. In the **repeat-until** loop, each permission p_i is compared with a permission p_j such that $p_i.id < p_j.id$. If the set of black nodes and arcs of G_{p_i} is not a subset of the set of black nodes and arcs of G_{p_j} (i.e., $G_{p_i} \not\preceq_{NE} G_{p_j}$, meaning that p_j has not been computed in a previous iteration by composing p_i with another authorization) and vice versa, function **Authorized** checks whether p_i and p_j can be composed (i.e., $p_j \rightarrow p_i$ or $p_i \rightarrow p_j$). If so, the identifier of the resulting composed permission becomes equal to the current maximum identifier (*maxid*) incremented by one. Each permission p has also a variable $p.maxcfr$, which keeps track of the highest identifier of the permissions compared with p. This variable avoids to check the same pair of permissions more than once. The composition process terminates when *maxcfr* of all permissions is equal to the highest identifier *maxid*. The function then checks if any of the permissions in *Applicable* dominates ($\preceq_N$) G_q. If this is the case, function **Authorized** returns TRUE; otherwise it returns FALSE.

EXAMPLE 6.1. *Consider the schema graph in Figure 3,*

AUTHORIZED(G_q)
Let *Applicable* be the set of permissions $p \in \mathcal{P}$ such that:
$\{n \in \mathcal{N}_p | \lambda_p(n) = \text{black} \vee \text{white}\} \subseteq \{n \in \mathcal{N}_q | \lambda_q(n) = \text{black} \vee \text{white}\}$
/* check individual permissions */
for each $p \in Applicable$ **do**
 if $G_q \preceq_N G_p$ **then return**(TRUE)
/* compose permissions */
$maxid := |Applicable|$
$counter := 1$
for each $p \in Applicable$ **do**
 $p.id := counter$
 $p.maxcfr := counter$
 $counter := counter + 1$
$idmin_i := 1$
repeat
  Let p_i be the permission with $p_i.id = idmin_i$
  $idmin_j := \textbf{Min}(\{p.id | p \in Applicable \wedge p_i.maxcfr < p.id\})$
  Let p_j be the permission with $p_j.id = idmin_j$
  $dominated :=$ NULL
  if $(G_{p_i} \not\preceq_{NE} G_{p_j}) \wedge (G_{p_j} \not\preceq_{NE} G_{p_i})$ **then**
    if $p_j \rightarrow p_i$ **then** $dominated := dominated \cup \{p_j\}$
    if $p_i \rightarrow p_j$ **then** $dominated := dominated \cup \{p_i\}$
  $p_i.maxcfr := p_j.id$
  if $dominated \neq$ NULL **then**
    $maxid := maxid + 1$
    $p_{maxid} := \textbf{Compose}(G, p_i, p_j)$
    $p_{maxid}.id := maxid$
    $p_{maxid}.maxcfr := maxid$
    $Applicable := Applicable - dominated \cup \{p_{maxid}\}$
  $idmin_i := \textbf{Min}(\{p.id | p \in Applicable \wedge p.maxcfr < maxid\})$
until $idmin_i =$ NULL
/* check resulting permissions */
for each $p \in Applicable$ **do**
 if $G_q \preceq_N G_p$ **then return**(TRUE)
return(FALSE)

Figure 9: Access control

id	1	2	3	4	5	6	7	8
	p_1	p_2	p_3	p_4	p_5			
initialization	1	2	3	4	5			
$p_2 \rightarrow p_1$	*1*	*2*	3	4	5	$p_1 \otimes p_2$		
$p_1 \not\rightarrow p_3$	*2*		*3*	4	5	6		
$p_1 \leftrightarrow p_4$	*3*		3	*4*	5	6	$p_1 \otimes p_4$	
$p_3 \not\rightarrow p_5$			*3*		*5*	6	7	
$p_3 \not\rightarrow (p_1 \otimes p_2)$			*5*		5	*6*	7	
$p_5 \not\rightarrow (p_1 \otimes p_2)$			6		*5*	*6*	7	
$p_3 \not\rightarrow (p_1 \otimes p_4)$			*6*		6	6	*7*	
$p_5 \not\rightarrow (p_1 \otimes p_4)$			7		*6*	6	*7*	
$(p_1 \otimes p_2) \rightarrow (p_1 \otimes p_4)$			7		7	*6*	*7*	$p_1 \otimes p_2 \otimes p_4$
$p_3 \not\rightarrow (p_1 \otimes p_2 \otimes p_4)$			*7*		7		7	*8*
$p_5 \not\rightarrow (p_1 \otimes p_2 \otimes p_4)$			8		*7*		7	*8*
$G_{p_1 \otimes p_4} \preceq_{NE} G_{p_1 \otimes p_2 \otimes p_4}$			8		8		*7*	*8*
			8		8		8	8

Figure 10: An example of algorithm execution

the set of permissions in Figure 5, and query q_1 in Figure 6. All the five permissions are applicable to q_1. The table in Figure 10 represents the execution, step by step, of function **Authorized** *on G_{q_1} by reporting the evolution of variable* p.maxcfr *for both original and composed permissions. Each column in the table corresponds to a permission, whose identifier is the label of the column itself. Each row in the table represents an iteration of the* **repeat-until** *loop, reporting both the dependence relationship between the composing permissions and the* maxcfr *for all permissions. Also, in each row the* maxcfr *of the permissions checked for a possible composition are reported in italic. When a permission is removed from* Applicable *(because subsumed by an added composed permission), its* maxcfr *is not reported anymore. Figure 8 illustrates the view graphs of the composed permissions generated by the algorithm.*

The following theorems state the correctness and complexity of function **Authorized**. Proofs are omitted due to space constraints.

THEOREM 6.2 (TERMINATION AND CORRECTNESS). *Given a query q and a set Applicable of applicable permissions, function* **Authorized** *terminates and returns* TRUE *iff q is authorized by $Applicable^{\otimes}$.*

THEOREM 6.3 (COMPLEXITY). *Given a query q and a set Applicable of n applicable permissions, the computational complexity of function* **Authorized** *is $O(n^3)$.*

7. RELATED WORK

In light of the crucial role that security has in the construction of future large-scale distributed applications, a significant amount of research has recently focused on the problem of processing distributed queries under access restrictions. Most of these works [5, 8, 9, 11, 12, 15] are based on the concept of *access pattern*, a profile associated with each relation/view where each attribute has a value that may either be *i* or *o* (i.e., input or output). When accessing a relation, the values for all *i* attributes must be supplied, to obtain the corresponding values of *o* attributes. Also, queries are represented in terms of Datalog, a query language based on the logic programming paradigm. The main goal of all these works is identifying the classes of queries that a given set of access patterns can support. There are several differences between the work on access patterns and the approach proposed in this paper. First, our proposal focuses on the definition of privileges on the components of a schema, naturally extending the approach normally used to describe privileges in a relational database; instead, access patterns describe authorizations as special formulas in a logic programming language for data access, outside the expertise of database and security administrators. Second, the work on access patterns typically requires the invocation of a processor of Datalog queries to identify if a query can be computed on the available access patterns, in general with exponential cost, compared to the polynomial complexity exhibited here. Finally, it is easier to adapt the model presented in this paper to scenarios where multiple subjects cooperate in query execution; we also observe that the idea of having input parameters can be immediately modeled in our proposal defining an ad-hoc join path with a relation under the subject control, demonstrating the flexibility of our approach. The models based on access patterns are not as easy to extend, since it is known that cooperative computation of logical queries presents several difficult issues.

Other related work is represented by classical studies on chase and data dependencies [1, 2, 4, 13]. The chase process exploits a specific data structure, called *tableau*, to represent a query or a relation. It is usually adopted to study and identify functional dependencies within a relation schema, to check if a decomposition is lossy or lossless, to evaluate if the result of a query q_i is contained in the result of another query q_j (or vice versa) without explicitly computing the queries. The algorithm we present in the paper can be considered a variant of chase computation for acyclic queries that adapts previous results to a configuration where each query corresponds to a security constraint (security aspects are not considered in the work on data dependencies).

A model with some similarity to the approach introduced in this paper has been presented in [16]. An interesting similarity lies in the exploitation of referential integrity constraints for the automatic identification of security compliance of queries with respect to views, but the modes of application in the two approaches are clearly distinct. In general, compared to our proposal, the approach in [16] operates at a lower level; it analyzes the integration with a relational DBMS optimizer and focuses on the consideration of "instantiated" queries, i.e., queries that present predicates that force attributes to assume specific values, aiming to evaluate compatibility of the instantiated queries with the authorized views. Our model instead operates at a more abstract level: we focus on a data integration scenario and provide an overall data-model characterization of the views; query instantiation with parameter is modeled by the addition of tables to the schema. The advantages of our approach are: (1) the model is applicable to a wider variety of contexts, and (2) the security requirements are expressed in a clearer way, with benefits for the security administrator and the reasoning system. In particular, we provide a complete technique for the identification of query compliance with authorizations and formally demonstrate the correct behavior of an efficient composition of authorizations, whereas in [16] no algorithm is provided for the evaluation of view composition, relying instead on a potentially expensive application of a few heuristics by the query optimizer. Overall, the two approaches complement each other and the exploration of their integration appears a fruitful line of research.

Acyclicity of the database schema allows us to obtain a limited computational complexity in authorization composition. It is well-known in the database community that cyclic schemas significantly increase the complexity of query processing and optimization. Recent research (e.g., [10]) has shown that it is possible to identify restrictions on the degree of schema cyclicity that permit to keep at polynomial complexity several query processing and schema design problems. It appears promising to extend those results to the scenario presented in this paper.

8. CONCLUSIONS

The evolution of networking and Web technology offers opportunities for interaction among separate information systems that will be exploited in many important applications and will produce great benefits to society. A preliminary condition for the realization of these benefits is the flexible and efficient support of access permissions. Our proposal represents a step in this direction, with powerful, yet simple, permissions and their composition for assessing query privileges facilitates the construction of modern security-compliant integration mechanisms.

9. ACKNOWLEDGMENTS

This work was supported in part by the EU, within the 7FP project, under grant agreement 216483 "PrimeLife" and by the Italian MIUR, within PRIN 2006, under project 2006099978 "Basi di dati crittografate". The work of Sushil Jajodia was partially supported by National Science Foundation under grants CT-0716567, CT-0627493, and IIS-0430402 and by Air Force Office of Scientific Research under grant FA9550-07-1-0527.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM TODS*, 4(3):297–314, 1979.

[3] P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database Systems - Concepts, Languages and Architectures*. McGraw-Hill Book Company, 1999.

[4] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[5] A. Calì and D. Martinenghi. Querying data under access limitations. In *Proc. of ICDE 2008*, Cancun, Mexico, April 2008.

[6] S. Dawson, S. De Capitani di Vimercati, P. Lincoln, and P. Samarati. Maximizing sharing of protected information. *Journal of Computer and System Sciences*, 64(3):496–541, May 2002.

[7] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Controlled information sharing in collaborative distributed query processing. In *Proc. of ICDCS 2008*, Beijing, China, June 2008.

[8] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. In *Proc. of ICDT 2005*, Edinburgh, UK, January 2005.

[9] D. Florescu, A. Y. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proc. of SIGMOD 1999*, Philadelphia, PA, June 1999.

[10] G. Gottlob. Computing cores for data exchange: new algorithms and practical solutions. In *Proc. of the PODS 2005*, Baltimore, MD, June 2005.

[11] G. Gottlob and A. Nash. Data exchange: Computing cores in polynomial time. In *Proc. of PODS 2006*, Chicago, IL, June 2006.

[12] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3):211–227 , 2003.

[13] D. Maier, A. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. In *Proc. of the SIGMOD 1979*, Boston, MA, June 1979.

[14] A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *Proc. of the ICDE89*, Los Angeles, CA, February 1989.

[15] A. Nash and A. Deutsch. Privacy in GLAV information integration. In *Proc. of ICDT 2007*, Barcelona, Spain, January 2007.

[16] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. of the SIGMOD 2004*, Paris, France, 2004.

[17] A. Rosenthal and E. Sciore. View security as the basis for data warehouse security. In *Proc. of DMDW'2000*, Stockholm, Sweden, June 2000.

[18] A. Rosenthal and E. Sciore. Administering permissions for distributed data: factoring and automated inference. In *Proc. of the IFIP 11.3 Working Conference in Database Security*, Niagara, Ontario, Canada, July 2001.