Chapter 9

DEDUCTIVE DATABASE SECURITY

Steve Barker

Abstract An approach is described for protecting the information in deductive databases from unauthorized retrieval and update requests. The approach we propose combines meta-programming and partial evaluation techniques, has a number of attractive technical results associated with it, and can be used as a basis for efficient implementations of protected forms of deductive database.

Keywords: Role-based access control, deductive databases, meta-programming, partial evaluation

1. Introduction

Deductive databases are predicted to become increasingly significant in the future [9]. However, in order for deductive databases to make an impact, it is important to consider practical matters like ensuring the security of the information they contain.

Despite the long recognized importance of database security, the issues relating to the security of deductive databases have hitherto been largely neglected. We address this gap in the literature by showing how a range of access control policies may be represented by using logic programs, and how these programs may be used to define an access policy to protect a deductive database from unauthorized access requests made by authenticated users.

In the approach we describe, logic programs are used by a *security administrator (SA)* to define *role-based access control (RBAC)* policies [11] that specify authorized forms of access to a deductive database. The logic programs we describe are based on the *Hierarchical (level 2A) RBAC* model from [11]. Henceforth, we refer to these logic programs as *RBAC_{H2A} programs*. In addition to the assignment of users and access privileges on objects to roles, *RBAC_{H2A}* programs include a definition of a *role hierarchy* [11].

In [4], modal logics are considered for specifying confidentiality requirements on deductive databases to preserve secrets. As such, the work described

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35697-6_26](https://doi.org/10.1007/978-0-387-35697-6_26)

E. Gudes et al. (eds.), *Research Directions in Data and Applications Security*

© IFIP International Federation for Information Processing 2003

in [4] addresses different issues to those we consider. In other (loosely) related work, Jamil [7] describes a query language which enables a user of a multi-level secure deductive database to reason about the beliefs held by users with different security clearances. In our approach, RBAC policies rather than MLS policies are specified to protect a deductive database, theorem-proving techniques are used to retrieve the information a user is authorized to see (rather than being used to reason about the beliefs of users), and updates are considered as well as retrievals.

An approach for protecting deductive databases using RBAC policies expressed as stratified logic programs was first described in [2] (see also [3]). The proposal in [2] is based on the use of a *holds* predicate to “compile” access control information into a deductive database. This approach enables the database to be specialized such that users of the database are only able to retrieve from and update authorized subsets of the database. Compiling access control information into a database enables the technique of *partial evaluation* [6] to be used to efficiently perform access requests.

Unfortunately, in the form of protected deductive databases described in [2], it is not possible to distinguish between information that is false in the database and information that is false because a user is not authorized to know that the information is true. Moreover, retrievals and updates are not treated in a uniform way, and the formulation of the protected form of a deductive database requires that non-stratified logic be used if the database includes some clause with a negative condition.

In contrast, in this paper we describe an approach which enables:

- the negative information as well as the positive information in a deductive database to be specified as being protected from unauthorized retrieval requests.
- a user to distinguish between information which is true or false in a deductive database and information that the user is not permitted to know the truth value of due to access control restrictions.
- the retrieval and update of the information in a deductive database to be treated in a uniform manner.

By combining partial evaluation and meta-programming techniques, it is possible to avoid using the large number of *holds* conditions that are required by the encoding used in [2], and without loss of expressive power or compromising performance measures.

The rest of the paper is organized thus. In Section 2, some basic notions in deductive databases, access control and RBAC are briefly described. In Section 3, we show how RBAC models may be represented as *locally stratified logic programs* [13]. In Section 4, we describe our proposal for formulating

deductive databases so that unauthorized retrieval requests are not permitted. In Section 5, we present a variant of the approach we describe in Section 4 which enables deductive databases to be protected from unauthorized update requests. Finally, in Section 6, some conclusions are drawn and suggestions for further work are made.

2. Basic Concepts

The deductive databases, $RBAC_{H2A}$ programs, and meta-programs that we describe in the sequel, consist of a finite set of *normal clauses* [8]. A normal clause takes the form:

$$H \leftarrow L1, L2, \dots, Lm \quad (m \geq 0).$$

The *head* of the clause, H , is an *atom* and $L1, L2, \dots, Lm$ is a conjunction of *literals* that constitutes the *body* of the clause. The conjunction of literals $L1, L2, \dots, Lm$ must be true (proven) in order for H to be true (proven). A literal is an atomic formula (a positive literal) or its negation (a negative literal); negation is *negation as failure*, and the negation of the atom A is denoted by *not* A . A clause with an empty body is an *assertion* or a *fact*.

To protect a deductive database D from unauthorized access requests, we use meta-programming and partial evaluation techniques to express D in a protected form D^* that ensures that access requests are only possible if an $RBAC_{H2A}$ program (see Section 3) which is associated with D^* specifies that the requested access is authorized. In the sequel, we will use D^* to denote an arbitrary protected database; ΔD^* is used to denote an updated state of D^* , and we use S^* to denote an arbitrary $RBAC_{H2A}$ program. The meta-programs we will use to define the semantics of authorized retrieval and authorized updates on D^* will be respectively denoted by $\mathcal{M}_R$ and $\mathcal{M}_U$.

An $RBAC_{H2A}$ program S^* is defined on a domain of discourse that includes: a set U of *users*, a set O of *objects*, a set A of *access privileges*, and a set R of *roles*.

The access privileges of interest to us in this paper are defined by the set:

$$A = \{true, false, insert, delete\}.$$

The semantics of the access privileges in A are defined thus:

- If a user u ($u \in U$) has the privilege *true* on an atomic formula $F \in HB(D)$ then u is permitted to know that F is true in D^* ($HB(D)$ is the *Herbrand base* of D [8]).
- If a user u ($u \in U$) has the privilege *false* on an atomic formula $F \in HB(D)$ then u is permitted to know that F is false in D^* .

- If a user u ($u \in U$) has the privilege *insert* on a ground instance of an atomic formula $F \in HB(D)$ then u is permitted to insert F into D^* .
- If a user u ($u \in U$) has the privilege *delete* on a ground instance of an atomic formula $F \in HB(D)$ then u is permitted to delete F from D^* .

The U , O , A and R sets respectively define the (disjoint and finite) sets of user, object, access privilege and role identifiers that form part of the universe of discourse for an $RBAC_{H2A}$ program. In this framework we have:

Definition 2.1 *An authorization is a triple (u, a, o) which denotes that a user u ($u \in U$) has the a access privilege ($a \in A$) on object o ($o \in O$).*

Definition 2.2 *If a is an access privilege and o is an object then a permission is a pair (a, o) denoting that a access is permitted on o .*

Definition 2.3 *A permission-role assignment is a triple (a, o, r) which denotes that the permission (a, o) is assigned to the role r .*

Definition 2.4 *A user-role assignment is a pair (u, r) which denotes that the user u is assigned to the role r .*

In this paper, we assume that an $RBAC_{H2A}$ program which defines a *closed policy* with no *session management* [11] is to be used to protect a deductive database. The implementation of sessions, open policies, and any number of hybrid access policies for protecting D^* necessitates that only minor modifications be made to the approach we describe.

In the sequel, constants in clauses will be denoted by symbols that appear in the lower case; variables will be denoted by using upper case symbols. Moreover, we will use the constants u , o , a and r to denote a, distinct and arbitrary, user, object, access privilege and role, respectively. We use the variables U , O , A and R to respectively denote the sets of users, objects, access privileges and roles in the *Herbrand Domain* of $D^* \cup S^*$ [8]. It should also be noted that $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ is completely hidden from “ordinary” users in order to restrict a user’s scope for drawing inferences about information which is secret in D^* .

Since $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ is a locally stratified program, it follows that the *well-founded semantics* [13] may be used for the declarative semantics for $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$. The restricted form of *SLG-resolution* [5] that is sufficient for goal evaluation on locally stratified programs may be used for the corresponding procedural semantics (the type of deductive database we consider protecting is an XSB database [10]). When SLG-resolution is used with $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$, a *search forest* is constructed starting from the *SLG-tree* with its root node labeled by the *goal clause* $Q \leftarrow Q$ [5]. From the

soundness and (search space) completeness of SLG-resolution (for flounder-free computations), we have:

Proposition 2.1 *$Q \in WFM(D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U)$ iff there is an SLG-derivation for $Q \leftarrow Q$ on $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ that terminates with the answer clause $Q \leftarrow$ (where $WFM(D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U)$ is the well-founded model of $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$).*

Since $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ is a set of locally stratified clauses we also have:

Proposition 2.2 *An $RBAC_{H2A}$ program has a unique 2-valued well-founded model which coincides with the perfect model of $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ [5].*

It immediately follows that:

Corollary 2.1 *$RBAC_{H2A}$ programs define a consistent and unambiguous set of authorizations.*

Although we use functions in our representation of $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$, our use of these functions does not result in unbounded terms being generated during computation. Hence:

Proposition 2.3 *$D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ satisfies the bounded-term-size property.*

An immediate corollary of the previous proposition is:

Corollary 2.2 *SLG-derivations with respect to $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ are guaranteed to be terminating.*

For implementations, an equivalent function-free form of $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ may be used. For that, the structured terms that appear in $D^* \cup S^* \cup \mathcal{M}_R \cup \mathcal{M}_U$ may simply be “flattened”. Moreover, if an $RBAC_{H2A}$ program is expressed using built-in comparison or mathematical operators then we assume that a SA will express these definitions in a *safe* form [12] such that the arguments of a condition involving these operators are bound prior to evaluating the condition.

3. $RBAC_{H2A}$ Programs

The access control programs we describe in this section are based on the specification of $RBAC_{H2A}$ programs from [1]. In [1], a user U is specified as being assigned to a role R by a SA defining a 2-place $ura(U, R)$ predicate. To record that the A access privilege on an object O is assigned to a role R , definitions of a 3-place $pra(A, O, R)$ predicate are used.

In a deductive database, the objects to be protected are n -ary EDB and IDB predicates; access privileges are from the set A for the $RBAC_{H2A}$ programs we consider. For example, $pra(true, q(V, Y, Z), r1) \leftarrow V \neq a$ may be used to specify that role $r1$ is assigned the permission to know that any instance of $q(V, Y, Z)$ such that $V \neq a$ is true. Similarly, $pra(delete, r(c, Y), R) \leftarrow Y < 20$ specifies that all roles are permitted to delete any instance of r such that the first argument of r is equal to c , and the second argument of r is a number less than 20.

To simplify the representation of S^* , we distinguish between permission-role assignments for retrieval requests and permission-role assignments for update requests. For the former we use a 3-place pra_R predicate and for the latter we use a 3-place pra_U predicate. To further simplify the representation of S^* we make use of the surjective function, $f : A \mapsto \{+, -\}$, defined thus:

$$f(x) = \begin{cases} +, & \text{if } x = true \text{ or } x = delete; \\ -, & \text{if } x = false \text{ or } x = insert. \end{cases}$$

The function f is used to map elements from A that may appear in the first argument of pra to $+$ or $-$. For instance, $pra(true, q(V, Y, Z), r1) \leftarrow V \neq a$ is expressed in a practical implementation of S^* by $pra_R(+, q(V, Y, Z), r1) \leftarrow V \neq a$ and $pra(delete, r(c, Y), R) \leftarrow Y < 20$ is expressed in a practical implementation of S^* by $pra_U(-, r(c, Y), R) \leftarrow Y < 20$.

An $RBAC_{H2A}$ role hierarchy is represented by a partial order $(R, \geq)$ that defines a seniority ordering $\geq$ on a set of roles R . If $r_i \in R$, $r_j \in R$, and $r_i \geq r_j$ then r_i inherits the permissions assigned to r_j (assuming that there is no permission overriding).

In [1], an $RBAC_{H2A}$ role hierarchy is expressed by a set of clauses that define a 2-place *senior_to* predicate (i.e., $\geq$) as the reflexive-transitive closure of an irreflexive-intransitive 2-place *ds* predicate, to wit:

$$\begin{aligned} senior_to(R1, R1) &\leftarrow ds(R1, _). \\ senior_to(R1, R1) &\leftarrow ds(_, R1). \\ senior_to(R1, R2) &\leftarrow ds(R1, R2). \\ senior_to(R1, R2) &\leftarrow ds(R1, R3), senior_to(R3, R2). \end{aligned}$$

The set of authorization triples defined by S^* are expressed by the following pair of clauses:

$$\begin{aligned} permitted_R(U, A, O) &\leftarrow ura(U, R1), senior_to(R1, R2), pra_R(A, O, R2). \\ permitted_U(U, A, O) &\leftarrow ura(U, R1), senior_to(R1, R2), pra_U(A, O, R2). \end{aligned}$$

The $permitted_R$ clause expresses that user U has A retrieval access on object O if U is assigned to a role $R1$ that is senior to the role $R2$ in an

$RBAC_{H2A}$ role hierarchy associated with the $RBAC_{H2A}$ program, and $R2$ has been assigned the A access privilege on O . The $permitted_U$ clause specifies the set of authorized updates defined by $S^* \cup \mathcal{M}_U$ on D^* .

4. Authorized Retrievals

In our proposal, $EDB(D^*)$ is always a set of ground atomic assertions, and $IDB(D)$ is represented by using a meta-predicate idb . That is, for each IDB clause of the form $I \leftarrow A1, \dots, Am, not B1, \dots, not Bn$ in D , a clause of the following form is included in D^* to define when I is true:

$$idb((I, +), ((A1, +), \dots, (Am, +), (B1, -), \dots, (Bn, -))).$$

Moreover, the following set of clauses is included in D^* to define when I is false:

$$\begin{aligned} & idb((I, -), ((A1, -))). \\ & \dots \\ & idb((I, -), ((Am, -))). \\ & idb((I, -), ((B1, +))). \\ & \dots, \\ & idb((I, -), ((Bn, +))). \end{aligned}$$

The $\mathcal{M}_R$ program is defined thus:

- (c1) $retrieve(U, (X, +)) \leftarrow permitted_R(U, X, +), not idb((X, +), -), X.$
- (c2) $retrieve(U, (X, -)) \leftarrow permitted_R(U, X, -), not idb((X, -), -), not X.$
- (c3) $retrieve(U, ((X, A1), (Y, A2))) \leftarrow permitted_R(U, X, A1),$
 $retrieve(U, (X, A1)), permitted_R(U, Y, A2), retrieve(U, (Y, A2)).$
- (c4) $retrieve(U, (X, A)) \leftarrow permitted_R(U, X, A), idb((X, A), Y), retrieve(U, Y).$

The (c4) clause expresses that a user's request to retrieve a logical consequence X from D^* will succeed if the user U has the access privilege A which permits U to know that X is true (provable) in D^* , and the conditions in the body Y of a clause defining X are provable from the subset of D^* that U is authorized to retrieve from. The (c3) clause specifies how conjoined conditions in the body of a clause are demonstrated. The (c1) clause defines when positive facts (true facts) may be disclosed to a user. To know that X is true in D^* , U must be permitted, by S^* , to know that X is true in U 's authorized view of D^* , and X must be true in D^* . Similarly, the (c2) clause is used to define when negative facts (facts which are false) may be disclosed to a user. For U to know that X is false (not provable) in D^* , U must be permitted, by S^* , to know that X is false in D^* , and X must be false in U 's authorized view of D^* .

The following results apply in the case of retrieval requests with respect to $D^* \cup S^* \cup \mathcal{M}_R$:

Proposition 4.1 $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (I, +))$ iff user u is permitted to know I holds in D^* from S^* , and $\forall Ai \in \{1, \dots, m\}$, $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (Ai, +))$ and, $\forall Bj \in \{1, \dots, n\}$, $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (Bj, -))$.

Proposition 4.2 $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (I, -))$ iff u is permitted to know that I is false in S^* and $\exists Ai \in \{1, \dots, m\}$ such that $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (Ai, -))$ or $\exists Bj \in \{1, \dots, n\}$ such that $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (Bj, +))$.

Proposition 4.3 Let u denote an authenticated user and E denote an arbitrary EDB assertion in D^* then $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (E, +))$ iff u is permitted to know that E holds from S^* and $WFM(D^* \cup S \cup M_R) \models E$.

Proposition 4.4 Let u denote an authenticated user and E denote an arbitrary EDB assertion in D^* then $WFM(D^* \cup S \cup M_R) \models \text{retrieve}(u, (E, -))$ iff u is permitted to know that E holds from S^* and $WFM(D^* \cup S \cup M_R) \not\models E$.

The key technical results for query evaluation on $D^* \cup S^* \cup \mathcal{M}_R$, involving an arbitrary predicate q follow directly from the soundness and (flounder-free) completeness results for SLG-resolution applicable to locally stratified theories:

Proposition 4.5 If $D^* \cup S^* \cup M_R \vdash_{SLG} \text{retrieve}(u, (q, +))$ then $WFM(D^* \cup S^* \cup M_R) \models \text{retrieve}(u, (q, +))$.

Proposition 4.6 If $D^* \cup S^* \cup M_R \vdash_{SLG} \text{retrieve}(u, (q, -))$ then $WFM(D^* \cup S^* \cup M_R) \models \text{retrieve}(u, (q, -))$.

Proposition 4.7 If $WFM(D^* \cup S^* \cup M_R) \models \text{retrieve}(u, (q, +))$ then $D^* \cup S^* \cup M_R \vdash_{SLG} \text{retrieve}(u, (q, +))$.

Proposition 4.8 If $WFM(D^* \cup S^* \cup M_R) \models \text{retrieve}(u, ((q, -)))$ then $D^* \cup S^* \cup M_R \vdash_{SLG} \text{retrieve}(u, (q, -))$.

The results that follow are corollaries of the soundness and completeness results for retrieval request evaluation by SLG-resolution on $\mathcal{M}_R$:

Proposition 4.9 (Safety) Let q be an arbitrary predicate symbol, u be an arbitrary user, and σ be an element of $\{+, -\}$. If $WFM(D^* \cup S^* \cup M_R) \not\models \text{retrieve}(u, (q, \sigma))$ then $D^* \cup S^* \cup M_R \not\vdash_{SLG} \text{retrieve}(u, (q, \sigma))$.

Proposition 4.10 (*Availability*) *Let q be an arbitrary predicate symbol, u be an arbitrary user, and σ be an element of $\{+, -\}$. If $WFM(D* \cup S* \cup M_R) \models \text{retrieve}(u, (q, \sigma))$ then $D* \cup S* \cup M_R \vdash_{SLG} \text{retrieve}(u, (q, \sigma))$.*

Safety guarantees that users can only retrieve the logical consequences that they are authorized to retrieve from $D* \cup M_R$ by $S*$; availability guarantees that a user can retrieve all logical consequences that he or she is authorized from $S* \cup M_R$ to retrieve from $D*$.

5. Authorized Updating

In our proposal, “ordinary” users may only change ground instances of EDB or IDB atoms. Moreover, to avoid view update problems, change requests may only be satisfied by modifying $EDB(D*)$. If an update request $\mathcal{R}$ cannot be satisfied by making changes to $EDB(D*)$ then $\mathcal{R}$ cannot be performed on $D*$. Only a SA or other trusted users should be permitted to make changes other than changes to $EDB(D*)$.

The meta-program $\mathcal{M}_U$, which controls access to $D*$, is specified thus:

- (c5) $\text{update}(U, (X, +)) \leftarrow \text{permitted}_U(U, X, +), \text{not idb}(X, (+, -))$.
- (c6) $\text{update}(U, (X, -)) \leftarrow \text{permitted}_U(U, X, -), \text{not idb}(X, (-, -))$.
- (c7) $\text{update}(U, ((X, A1), (Y, A2))) \leftarrow \text{permitted}_U(U, X, A1),$
 $\text{update}(U, (X, A1)), \text{permitted}_U(U, Y, A2), \text{update}(U, (Y, A2))$.
- (c8) $\text{update}(U, (X, A)) \leftarrow \text{permitted}_U(U, X, A), \text{idb}((X, A), Y), \text{update}(U, Y)$.

The clause (c5) specifies that user U can insert an instance X of an EDB predicate into $D*$ if U is permitted by $S*$ to insert X into $D*$. Conversely, (c6) expresses that U can delete an instance X of an EDB predicate from $D*$ if U is permitted by $S*$ to delete X from $D*$. The clause (c8) specifies that U may perform a change involving an instance X of an IDB predicate if U has the authorization to perform the change on X , and to perform the changes to make the set of literals Y true in U 's authorized view of $\Delta D*$. The set of literals Y (and hence X) is made true in $\Delta D*$ if U is permitted to change $EDB(D*)$ such that the conjunction of literals in Y is true in U 's authorized view of $\Delta D*$. The clause (c7) is used to reduce a conjunction of literals in the body of an IDB predicate to a set of changes to $EDB(D*)$ which is sufficient to satisfy the request to change an instance of the IDB predicate.

The semantics of updates we implement for $D*$ may be expressed thus (where o is a ground instance of an n -ary EDB or IDB atom):

Proposition 5.1 *A user u inserts o into $D*$ iff $WFM(D* \cup S* \cup \mathcal{M}_U) \models \text{update}(u, (o, +))$.*

Proposition 5.2 *A user u deletes o from D^* iff $WFM(D^* \cup S^* \cup \mathcal{M}_U) \models \text{update}(u, (o, -))$.*

Moreover, we include the following pair of clauses in D^* to specify that insert (delete) privilege on object O implies the privilege to know that O is true (false) in ΔD^* :

$$\begin{aligned} \text{pra}_R(+, O, R) &\leftarrow \text{pra}_U(+, O, R). \\ \text{pra}_R(-, O, R) &\leftarrow \text{pra}_U(-, O, R). \end{aligned}$$

Since update permissions imply read permissions we have:

Proposition 5.3 *If $WFM(D^* \cup S^* \cup \mathcal{M}_U) \models \text{update}(u, (o, +))$ then $WFM(\Delta D^* \cup S^* \cup \mathcal{M}_R) \models \text{retrieve}(u, (o, +))$.*

Proposition 5.4 *If $WFM(D^* \cup S^* \cup \mathcal{M}_U) \models \text{update}(u, (o, -))$ then $WFM(\Delta D^* \cup S^* \cup \mathcal{M}_R) \models \text{retrieve}(u, (o, -))$.*

That is, a request by a user u to insert o into D^* will be satisfied in the updated state ΔD^* of D^* iff o is true in u 's authorized view of ΔD^* . Similarly, a request by u to delete o from D^* will be satisfied in ΔD^* iff o is false in u 's authorized view of ΔD^* .

In terms of SLG-resolution we have:

Theorem 5.1 *If $\text{update}(u, (o, +)) \leftarrow$ is an answer clause by SLG-resolution on $D^* \cup S^* \cup \mathcal{M}_U$ then $\Delta D^* \cup S^* \cup \mathcal{M}_R \vdash_{SLG} \text{retrieve}(u, (o, +))$.*

Theorem 5.2 *If $\text{update}(u, (o, -)) \leftarrow$ is an answer clause by SLG-resolution on $D^* \cup S^* \cup \mathcal{M}_U$ then $\Delta D^* \cup S^* \cup \mathcal{M}_R \vdash_{SLG} \text{retrieve}(u, (o, -))$.*

From each successful SLG-derivation, $\mathcal{Z}$, for the evaluation of an access request with respect to $D^* \cup S^* \cup \mathcal{M}_U$, the conjunction of *permitted* resolvents processed in $\mathcal{Z}$ defines a candidate authorized change transaction on D^* that will satisfy a user's update request.

Proposition 5.5 *There will be m authorized change transactions iff there are m successful SLG-derivations for the evaluation of an update request with respect to $D^* \cup S^* \cup \mathcal{M}_U$.*

Each of the m authorized change transactions generated from $D^* \cup S^* \cup \mathcal{M}_U$ includes n sets of pairs of the form (OP_i, E_i) ($i \in \{1, \dots, n\}$) where OP_i is an element from the set $\{+, -\}$, and E_i is a ground instance of an EDB predicate that must be inserted into $EDB(D^*)$ (if $OP = '+'$) or must be deleted from $EDB(D^*)$ (if $OP = '-'$) in order to satisfy a user's change request.

Given some authorized change transaction τ consisting of a set of n (OP_i, E_i) pairs viz. $\{(OP_1, E_1), (OP_2, E_2), \dots, (OP_i, E_i)\}$, the set of changes required on D^* is the conjunction of operations in τ . That is, the OP_1 operation must be performed on E_1 and the OP_2 operation must be performed on E_2 and $\dots$ and the OP_i operation must be performed on E_i .

The set $\mathcal{T}$ of m authorized change transactions $\{T_1, T_2 \dots T_m\}$ is used by an update routine which will perform the operations in one T_i transaction in $\mathcal{T}$ ($i \in \{1..m\}$). If $|\mathcal{T}| > 1$ then a view update or belief revision problem arises, and some criterion must be used to select a preferred authorised change transaction. These problems are well known, and their discussion is well beyond the scope of this paper.

6. Conclusions and Further Work

The encoding we have described, for protecting deductive databases from unauthorized access requests, is simpler than that used in [2]. However, expressive power is not compromised and technical results are preserved. Moreover, the meta-programs we use for protecting deductive databases may be efficiently implemented.

In future work, we intend to investigate how our work on temporal authorization models (e.g., [1]) may be incorporated into the approach we have described here.

References

- [1] Barker, S., Data protection by logic programming, *Proceedings of the First International Conference on Computational Logic (LNAI 1861)*, Springer, pp. 1300-1313, 2000.
- [2] Barker, S., Protecting deductive databases from unauthorized retrieval requests, *Proceedings of the Fourteenth IFIP WG 11.3 Conference on Database and Applications Security*, 2000; also published in B. Thuraisingham et al., (eds.), *Data and Applications Security*, Kluwer, pp. 185-196, 2001.
- [3] Barker, S., Protecting deductive databases from unauthorized retrieval and update requests, *Journal of Data and Knowledge Engineering*, 2002.
- [4] Bonatti, P., Kraus, S. and Subrahmanian, V., Foundations of secure deductive databases, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7(3), pp. 406-422, 1995.
- [5] Chen, W. and Warren, D., Tabled evaluation with delaying for general logic programs, *Journal of the ACM*, vol. 43(1), pp. 20-74, 1996.
- [6] Hogger, C., *Essentials of Logic Programming*, Oxford University Press, 1990.

- [7] Jamil, H., Belief reasoning in MLS deductive databases, *Proceedings of the ACM SIGMOD Conference*, pp. 109-120, 1999.
- [8] Lloyd, J., *Foundations of Logic Programming*, Springer, 1987.
- [9] Minker, J., Logic and databases: A 20 year retrospective, *Proceedings of the First International Workshop on Logic in Databases (LNCS 1154)*, Springer, pp. 3-57, 1996.
- [10] Sagonas, K., Swift, T. and Warren, D., XSB: An efficient deductive database system, *Proceedings of the ACM SIGMOD Conference*, pp. 442-453, 1994.
- [11] Sandhu, R., Ferraiolo, D. and Kuhn, R., The NIST model for role-based access control: Towards a unified standard, *Proceedings of the Fourth ACM Workshop on Role-Based Access Control*, pp. 47-61, 2000.
- [12] Ullman, J., *Principles of Database and Knowledge-Base Systems: Volume 1*, Computer Science Press, 1990.
- [13] Van Gelder, A., Ross, K. and Schlipf, J., The well-founded semantics for general logic programs, *Journal of the ACM*, vol. 38(3), pp. 620-650, 1991.