Algorithms for Maintaining Authorization Base

William Brooks, V. Wiktor Marek, Miroslaw Truszczynski

Department of Computer of Science
University of Kentucky
Lexington, KY 40506-0046
{bbrooks|marek|mirek}@cs.engr.uky.edu

Abstract

We present algorithms for access rights control in multiuser, object-oriented databases. These algorithms follow the model of authorization introduced in [6]. We show how the three basic operations: *AccessControl*, *Grant*, and *Revoke* can be efficiently implemented using techniques for manipulating partially ordered sets. Several techniques for maintenance of authorization bases are presented and the complexity of the algorithms is established.

1 Introduction

Multiuser databases require a mechanism to control access rights to objects that are stored in them. Different users may and will have different access rights to different objects in the database. A proposal for a mechanism to grant access rights on objects to the users is described in [6]. It is based on the concept of an *authorization base*. The authors identify three sets: a set S of subjects (users of a database), a set O of objects stored in the database, and a set A of access types. A triple $(s, o, +a)$, where $s \in S$, $o \in O$ and $a \in A$, is called an *explicit strong positive authorization*. The intuition is that user s has access a to object o. Similarly, a triple $(s, o, -a)$ is called an *explicit strong negative authorization* and its informal meaning is that user s does *not* have access a to object o (though it may have access of some other type to this object). An *authorization*

base is a collection AB of explicit strong positive and negative authorizations satisfying some additional closure constraints.

To describe these constraints, observe that all three sets S, O and A are endowed with a hierarchy (partial ordering). For instance, the hierarchy of users may reflect their positions in the organizational structure of a company that owns the database. Similarly, there is a natural hierarchy of objects, the *accumulation* hierarchy, in which objects serving as attribute values of other objects precede them. Finally, orderings are imposed on sets of access types. For instance, *write* access to an object implies *read* access to the object. Users higher in their hierarchy have at least the same access rights as their subordinates. Once the user has access of type a to an object o, s/he has also at least the same access to all objects lower than o in the object hierarchy. Finally, once the user has access of type a to an object o, it has also accesses of weaker types (in the hierarchy of access types) to the object o. These rules are concerned with positive authorizations. They have their dual forms for negative authorizations. For instance, subordinates of a user with negative authorization for the access of type a to an object o (that is, with no access a to object o) must also have such negative authorization (no access a to o). Authorization bases are those sets of positive and negative authorizations that are closed under these rules.

The fundamental role of an authorization base is to provide decisions concerning the access rights of users to objects. Given a triple (s, o, a), we can have one of the following four situations:

1. the authorization base has no information on whether user s has access a to object o
2. the authorization base contains (and, thus, implies) only the positive authorization for s to access a on o
3. the authorization base contains (and, thus, implies) only the negative authorization for s to access a on o
4. the authorization base contains both authorizations, positive and negative.

The possibilities (1) and (4) are undesirable. Possibility (1) means that (s, o, a) is underdefined and possibility (4) means that (s, o, a) is overdefined. Hence, they do not provide a unique answer to whether the access right should be granted. The goal is to propose a system in which for every triple (s, o, a) either (2) or (3) holds. This need was recognized in [6] and some solutions were proposed there.

In this paper, we propose an abstract model of the concept of an authorization base exploiting properties of partial orderings. We propose to use the Closed World Assumption of Reiter [7] to simplify the problem of representation of an authorization base and to enforce that for every triple (s, o, a), exactly one of the possibilities (2) and (3) will hold. We will further study the issue of representability of authorization bases. We will consider several different approaches to this problem. In one of them, an authorization base is represented by means of an antichain. Another one is somewhat related to the original proposal of [6]. The idea is to represent an authorization base as a *sequence* (rather than a set) of positive and negative authorizations.

Each representation mechanism must support algorithms for three basic operations on authorization bases:

1. *AccessControl*: a procedure that given an authorization base AB and a triple (s, o, a) returns TRUE if user s has access of type a to object o, and FALSE, otherwise.

2. *Grant*: a procedure that adds to authorization base AB a new authorization x and all authorizations implied by x. No other authorizations are added. The result will be denoted by $AB + x$.

3. *Revoke*: a procedure that eliminates from an authorization base AB an authorization x and all authorizations that imply x. No other authorizations are removed. The result will be denoted by $AB - x$.

In the paper we will describe these procedures for each of our representations of authorization bases.

2 Formal model of authorization

As in [6] (see also [1] and [5]) we consider the following three sets: the set S of subjects, O of objects and A of access types (called authorizations in [6]). Subsets AB of $S \times O \times A$ (satisfying some additional conditions discussed below) are called *authorization bases*. We interpret the fact that $(s, o, a) \in AB$ as: "user s has access type a on object o". In other words, authorization bases in our sense consist only of positive authorizations. The negative authorizations are obtained by Closed World Assumption [7]. That is, AB implies a negative authorization $(s, o, -a)$ if and only if $(s, o, a) \notin AB$. Consequently, for

every triple (s, o, a), either a positive or negative authorization is implied by AB (but not both).

We will now formalize the closure properties required of authorization bases. Let us recall that each of the sets S, O and A has its own hierarchy. In [6] these hierarchies are modeled by partial orderings: $\preceq_S, \preceq_O$, and $\preceq_A$, respectively. Consider an authorization (s', o', a'). It means that user s' has access a' to object o'. Consider now user s such that $s' \preceq_S s$ (that is, user s is higher than s' in the hierarchy of users). Then user s also must have access a' to object o'. Even more, if for some object o, $o \preceq_O o'$ (that is, o is a "part" of o'), then s should have access a' to o, as well. Finally, if access type a is weaker than a', $a \preceq_A a'$, then s should have access a to o. In other words, *if s is higher in the hierarchy of users than s' (or $s = s'$), and o is a part of o' (or $o = o'$), and a is weaker than a' (or $a = a'$), then if s' has access a' to o' then s has access a to o.* This intuition motivates the following partial ordering on the set of authorizations $S \times O \times A$:

$$(s', o', a') \preceq (s, o, a) \text{ if and only if } (s' \preceq_S s) \wedge (o \preceq_O o') \wedge (a \preceq_A a')$$

The idea is that authorizations imply all authorizations that are greater or equal in this ordering $\preceq$.

This ordering and the Closed World Assumption allow us to replace the closure principles discussed in the introduction with a single rule (upward closure rule UC):

UC: if $(s', o', a') \in AB$ and $(s', o', a') \preceq (s, o, a)$, then $(s, o, a) \in AB$.

We are now ready to provide a complete formal definition of authorization bases. By an *authorization space* we mean the partially ordered set $(S \times O \times A, \preceq)$, and by an authorization base we mean any subset AB of the authorization space which satisfies the rule UC. The rule UC guarantees that authorization bases satisfy the three closure rules for positive authorizations introduced in [6] and discussed in the introduction. In fact, this rule is equivalent to their conjunction. In addition, the rule UC implies that authorization bases satisfy also the following downward closure rule (DC):

DC: if $(s', o', a') \notin AB$ and $(s, o, a) \preceq (s', o', a')$, then $(s, o, a) \notin AB$.

This observation implies that if Closed World Assumption is used to provide information on negative authorizations, then the set of negative authorizations implied by AB satisfies the three closure rules for negative authorizations mentioned in the introduction.

In this paper we will discuss methods to represent authorization bases, that is, subsets of $S \times O \times A$ satisfying the rule UC. We will also describe algorithms for the procedures

ControlAccess, *Grant* and *Revoke*. To this end, we will use a more general setting of arbitrary partial orders (we will call it an *abstract* setting for authorization bases). At the end of the paper, in Section 5, we will further simplify the algorithms by utilizing the fact that the authorization space is the product of three smaller sets S, O and A.

3 Abstract model of authorization bases

We are now in a position to formulate the abstract version of the model of authorization bases. In this abstract model, an authorization space is an arbitrary partially ordered set $(U, \preceq)$ (for $U = S \times O \times A$ and $\preceq$ defined as above, we obtain the situation discussed in the previous section).

A subset $T \subseteq U$ is called *upward closed* (UC, in short) if

$$\forall_{x,y}((x \in T \wedge x \preceq y) \Rightarrow y \in T).$$

Similarly, a subset $T \subseteq U$ is called *downward closed* (DC, in short) if

$$\forall_{x,y}((x \in T \wedge y \preceq x) \Rightarrow y \in T).$$

UC sets capture the intuition of authorization bases as introduced above (in fact, we will often refer to UC sets as authorization bases). Similarly, DC sets describe the complements of the authorization bases. The DC sets will be needed in the discussion of the process of revoking authorizations.

We define now closure operators $Cl{\uparrow}(T)$ and $Cl{\downarrow}(T)$ as follows:

$$Cl{\uparrow}(T) = \{x : \exists_{y \in T} y \preceq x\}$$

$$Cl{\downarrow}(T) = \{x : \exists_{y \in T} x \preceq y\}.$$

Informally, $Cl{\uparrow}(T)$ consists of these authorizations that have to be granted once authorizations from T are granted, and $Cl{\downarrow}(T)$ consists of these authorizations that must be revoked once the authorizations from T are revoked.

We now list without proof a number of properties of upward and downward closed sets and of the closure operators.

Proposition 3.1 *1. If X, Y are UC sets then $X \cap Y$ and $X \cup Y$ are UC sets. Similarly for DC sets.*

2. *If X is a UC set and Y is a DC set, then $X \setminus Y$ is a UC set and $Y \setminus X$ is a DC set.*

3. *For every $X \subseteq U$, $Cl\uparrow(X)$ is a UC set and $Cl\downarrow(X)$ is a DC set.*

It follows from Proposition 3.1 that if X is a UC set then $X \cup Cl\uparrow(\{x\})$ is a UC set, as well. In fact, it is the least UC set containing $X \cup \{x\}$. We will denote it by $X + x$. If X is an authorization base and x is an authorization to be granted, it is clear that $X + x$ is exactly the intended result of *Grant*(X, x). Similarly, $X \setminus Cl\downarrow(\{x\})$ is the largest UC set included in X and not containing x. We will denote it by $X - x$. It is the intended result of *Revoke*(X, x).

A straightforward approach to the problem of maintaining authorization bases is to explicitly store the whole UC set (authorization base) X. Under this representation, *ControlAccess*(X, x) is a procedure returning TRUE if $x \in X$ and FALSE, otherwise. The procedure *Grant*(X, x) replaces X by $X + x$, that is, by $X \cup Cl\uparrow(\{x\})$. Similarly, the procedure *Revoke*(X, x) replaces X by $X - x$, that is, by $X \setminus Cl\downarrow(\{x\})$.

These procedures can be implemented as follows. Let us assume that we maintain the Hasse diagram H of $(U, \preceq)$. That is, assume that for each vertex $u \in U$ we maintain two lists: $in(u)$ containing all immediate predecessors of u, and $out(u)$ containing all immediate successors of u. Let us denote the size of this representation of H by h. We will use this notation throughout the paper.

We maintain an authorization base X by marking all nodes of X by *black*. Under this representation, *ControlAccess*(X, x) takes $O(1)$ (assuming that we can access each node from U in H in constant time). *Grant*(X, x) can be implemented by performing a depth-first search from x, using lists $in(u)$, and marking each visited vertex *black*. Similarly, *Revoke*(X, x) can be implemented by performing a depth-first search from x, using the lists $out(u)$, and unmarking each visited *black* vertex. Hence, these procedures take linear time in the size of the Hasse diagram.

A major problem with this approach is that it requires that the Hasse diagram H, which may be very large, be explicitly maintained. In the next section we will introduce two different proposals. In the case when $U = S \times O \times A$, they can be implemented using a much more economical representation of the Hasse diagram H.

4 Techniques for maintenance of authorization bases

Let us start with an example. Let $S = \{bill, victor, mirek\}$, $O = \{o_1, \ldots, o_7\}$ and $A = \{sc, r, w\}$, where sc denotes the right to see the scheme of the object (record), r is the *read* access and w is the *write* access. Assume also that these sets are endowed with the partial orderings whose Hasse diagrams are shown in Figure 1.

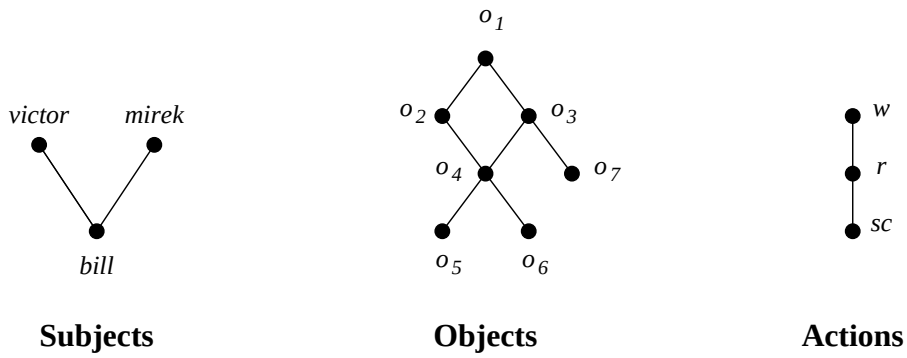

Figure 1: Hasse diagrams of $\preceq_S$, $\preceq_O$ and $\preceq_A$

Let us consider the authorization base in which *mirek* has complete access to all objects, *victor* has r and sc access to o_2, o_4, o_5 and o_6 and, sc access to o_7, and finally, *bill* has r access to o_5 and o_6 and sc access to o_2, o_4, o_5 and o_6. In other words, the authorization base AB consists of the following elements:

$(mirek, o_1, w), (mirek, o_2, w), \ldots, (mirek, o_7, w),$
$(mirek, o_1, r), (mirek, o_2, r), \ldots, (mirek, o_7, r),$
$(mirek, o_1, sc), (mirek, o_2, sc), \ldots, (mirek, o_7, sc),$

$(victor, o_2, r)$, $(victor, o_4, r)$, $(victor, o_5, r)$, $(victor, o_6, r)$,
$(victor, o_2, sc)$, $(victor, o_4, sc)$, $(victor, o_5, sc)$, $(victor, o_6, sc)$, $(victor, o_7, sc)$,

$(bill, o_5, r)$, $(bill, o_6, r)$,
$(bill, o_2, sc)$, $(bill, o_4, sc)$, $(bill, o_5, sc)$, $(bill, o_6, sc)$,

This set consists of 36 elements. However, it is uniquely determined by a much smaller set, say D, of its minimal elements, which contains just 8 elements:

$(mirek, o_1, w)$,

$(victor, o_2, r)$, $(victor, o_4, r)$,
$(victor, o_7, sc)$,

$(bill, o_5, r)$, $(bill, o_6, r)$,
$(bill, o_2, sc)$, $(bill, o_4, sc)$.

It can be checked that $Cl\uparrow(D) = AB$. Hence, to decide whether $x \in AB$, it is enough to decide whether $x \in Cl\uparrow(D)$, that is, whether there is an element $d \in D$ such that $d \preceq x$. For example, assume that we have a query $(victor, o_6, sc)$. Since $(victor, o_2, r)$ is in D and $(victor, o_2, r) \preceq (victor, o_6, sc)$, access sc to object o_6 should be approved for *victor*.

On the other hand, $(bill, o_2, r)$ is *not* approved since there is no tuple $(s, o, a) \in D$ such that $(s, o, a) \preceq (bill, o_2, r)$.

In the following subsections we will describe two techniques for maintenance of authorization bases using the ideas employed in the example above.

4.1 Representing authorization bases by antichains

An *antichain* is a subset D of U such that for all $x, y \in D$, if $x \neq y$ then $\neg(x \preceq y) \wedge \neg(y \preceq x)$. We have the following proposition showing that UC sets can be represented as upward closures of antichains. A dual result holds for DC sets.

Proposition 4.1 *Let $(U, \preceq)$ be a finite partially ordered set. A subset $T \subseteq U$ is a UC set if and only if there exists an antichain D such that $T = Cl\uparrow(D)$. Moreover, such an antichain D is unique and consists precisely of the minimal elements of T. Similarly, a set $X \subseteq U$ is a DCS if and only if there exists an antichain E such that $X = Cl\downarrow(E)$. Moreover, such antichain E is unique and consists of the maximal elements in X.*

Proposition 4.1 states that for every authorization base AB there is a unique antichain D such that $AB = Cl\uparrow(D)$. Moreover, D consists of all minimal elements in AB. Hence, authorization bases can be represented by antichains of their minimal elements.

We will use this representation now. This will require modifications in the procedures *ControlAccess*, *Grant* and *Revoke*.

In the procedure *ControlAccess*(D, x), we assume that D is the antichain of all minimal elements of an authorization base AB, that is, $AB = Cl{\uparrow}(D)$. Moreover, x is an element of an authorization space. The procedure returns TRUE if $x \in AB$ and FALSE otherwise.

ControlAccess(D, x)
if $x \in Cl{\uparrow}(D)$ **then return**{TRUE}
else return{FALSE}

Let us assume that we maintain the Hasse diagram H of the authorization space $(U, \preceq)$ by means of two lists: $in(u)$, $u \in U$, of immediate predecessors of u, and $out(u)$, $u \in U$, of immediate successors of u. Then, the test to check if $x \in Cl\uparrow(D)$ can be performed by executing the depth-first search from D upwards (using the lists $in(u)$). Clearly, this search takes linear time in h (recall that h denotes the size of H). Hence, the procedure *ControlAccess* can be implemented to run in time $O(h)$.

The problem of adding a new authorization is not much more complex. The corresponding procedure *Grant*(D, x) is described below. We assume here D is the antichain of all minimal elements of an authorization base AB, that is, $AB = Cl{\uparrow}(D)$. Moreover, x is an element of an authorization space. The procedure replaces D by a new antichain of minimal elements of $AB + x$.

Grant(D, x)
if $x \in Cl{\uparrow}(D)$ **then stop**
else $D := (D \setminus Cl{\uparrow}(\{x\})) \cup \{x\}$

As before, if the Hasse diagram H of the authorization space $(U, \preceq)$ is maintained as two sets of lists $in(u)$ and $out(u)$, $u \in U$, the test whether $x \in Cl\uparrow(D)$ can be performed in linear time by using a depth-first search upward from D. In addition, if we mark the nodes from D in the Hasse diagram (which can be done in linear time), then $D \setminus Cl{\uparrow}(\{x\})$ can also be computed in linear time by running a depth-first search from x upwards. Hence, *Grant*(D, x) can be implemented to run in linear time in h.

Proposition 4.2 *The procedure Grant*(D, x) *is correct.*

Proof: Assume $x \in AB$. Then, $x \in Cl\uparrow(D)$. In this case, the same set D has to be returned and this is exactly what our procedure does. In the case when $x \notin AB$, then

the set of minimal elements of $AB + x$ is $(D \setminus Cl{\uparrow}(\{x\})) \cup \{x\}$. This is again precisely the effect of the procedure. □

Revoke is the most complex of the three operations. On input, D is an antichain that represents an authorization base AB and x is an authorization that is to be revoked. On output, the set D contains the antichain representing the updated authorization base, that is, the result of the deletion of x from AB: $AB \setminus Cl{\downarrow}(\{x\})$.

Revoke(D, x)
$ND := \emptyset$
for every $y \in Cl{\uparrow}(D) \setminus D$ **do**
  if $Cl{\uparrow}(D) \cap in(y) \subseteq Cl{\downarrow}(\{x\})$ **then** $ND := ND \cup \{y\}$
$D := ND \cup (D \setminus Cl{\downarrow}(\{x\}))$

Again assume that the Hasse diagram H is represented by the lists $in(u)$ and $out(u)$, $u \in U$. Then, $Cl{\uparrow}(D) \setminus D$ can be computed in linear time in h. By marking *black* all nodes of H which are in $Cl{\uparrow}(D)$ and by marking *red* all elements of H which are in $Cl{\downarrow}(\{x\})$ (both tasks can be accomplished by performing a depth-first search, up from D and down from x, respectively), the total time for all the tests $Cl{\uparrow}(D) \cap in(y) \subseteq Cl{\downarrow}(\{x\})$ is also linear in h. Consequently, the whole procedure can be implemented in time $O(h)$.

Proposition 4.3 *Procedure Revoke*(D, x) *is correct.*

Proof: Let D_i denote the antichain D on input and let D_o will be the result of the procedure. Clearly, any element in D_i which is not in $Cl{\downarrow}(\{x\})$ is minimal in the set $Cl{\uparrow}(D_i) \setminus Cl{\downarrow}(\{x\})$. All these elements are included in D_o by the procedure. All other elements of D_i must be removed. However, some new minimal elements still need to be added to D_o. They all belong to $Cl{\uparrow}(D_i) \setminus D_i$. An element $y \in Cl{\uparrow}(D_i) \setminus D_i$ is minimal in $Cl{\uparrow}(D_i) \setminus Cl{\downarrow}(\{x\})$ if and only if each its predecessor does not belong to $Cl{\uparrow}(D_i) \setminus Cl{\downarrow}(\{x\})$. To check this, it is enough to verify that each predecessor of y which belongs to $Cl{\uparrow}(D_i)$ belongs also to $Cl{\downarrow}(\{x\})$. But this is exactly what our procedure tests when generating the set ND. □

As it is now, this approach is worse than the approach proposed before. All procedures run in linear time in the size h of the Hasse diagram H, while in our first approach *ControlAccess* takes constant time and *Grant* and *Revoke* take linear time in h. In Section 5 we will show that at a cost of some preprocessing, the running time of the procedures presented in this section will be significantly reduced. Moreover, there will be no need to maintain the entire Hasse diagram.

4.2 Lazy maintenance of authorization bases

We will now present another technique for the maintenance of authorization bases. The key observation here is that, with an increase in the cost of *ControlAccess* we can reduce the cost of *Grant* and *Revoke* to constant.

By an *elementary update* we mean a pair (ϵ, x) where $\epsilon \in \{+, -\}$ and $x \in U$. Such pair (ϵ, x) can be treated as an operator on authorization bases. Specifically, the effect of (ϵ, x) on AB is $AB\epsilon x$. That is, if $\epsilon = +$ then the result is $AB + x$, and if $\epsilon = -$ then the effect is $AB - x$.

An authorization base AB can be represented as $Cl\uparrow(D)$, where $D = \{x_1, \ldots, x_n\}$ is the antichain of the minimal elements of AB. Consequently, we have

$$AB = ((\ldots(\emptyset + x_1) + x_2)\ldots) + x_n.$$

However, every sequence of elementary updates determines an authorization base. That is, for an arbitrary sequence of elementary updates

$$\mathbf{a} = ((\epsilon_1, x_1), \ldots, (\epsilon_n, x_n))$$

the corresponding authorization base is:

$$((\ldots(\emptyset\epsilon_1 x_1)\epsilon_2 x_2)\ldots)\epsilon_n x_n$$

Thus, for instance, if $\mathbf{a} = ((+, x), (+, y), (-, t))$ then the authorization base defined by $\mathbf{a}$ is $((\emptyset + x) + y) - t = (Cl\uparrow(\{x\}) \cup Cl\uparrow(\{y\})) \setminus Cl\downarrow(\{t\})$.

Notice that we just gave a semantics to the sequences of elementary updates. This semantics allows us to represent authorization bases as sequences of elementary updates. As a consequence, the *Grant* and *Revoke* procedures consist only of appending the sequence with a new elementary update and, thus, take constant time.

The price that we are going to pay is in the efficiency of *ControlAccess* procedure. We will describe this procedure now. We assume that $\mathbf{a} = ((\epsilon_1, x_1), \ldots, (\epsilon_n, x_n))$ is a sequence of elementary updates and x is an element of the authorization space.

ControlAccess$(\mathbf{a}, x)$
for $m = n$ **downto** 1 **do**
    if $\epsilon_m = +$ **and** $x_m \in Cl\downarrow(\{x\})$ **then return**{TRUE}
    if $\epsilon_m = -$ **and** $x_m \in Cl\uparrow(\{x\})$ **then return**{FALSE}
return{FALSE}

Proposition 4.4 *Procedure ControlAccess is correct.*

Proof: By induction on n. If $n = 0$, then the authorization base determined by $\mathbf{a}$ is empty. Consequently, only the last instruction is invoked and the procedure returns FALSE, as needed.

Let us now assume that the statement is valid for sequences of length n. Observe that

$$(((\ldots((\emptyset\epsilon_1 x_1)\epsilon_2 x_2)\ldots)\epsilon_n x_n)\epsilon_{n+1}, x_{n+1}) = AB'\epsilon_{n+1}x_{n+1},$$

where

$$AB' = ((\ldots((\emptyset\epsilon_1 x_1)\epsilon_2 x_2)\ldots)\epsilon_n x_n$$

Thus, all we need to do is to see the effect of one step in our algorithm. When $\epsilon_{n+1} = +$ then if $x_{n+1} \in Cl{\downarrow}(\{x\})$ we answer TRUE, otherwise we use the inductive assumption and test if x belongs to $((\ldots((\emptyset\epsilon_1 x_1)\epsilon_2 x_2)\ldots)\epsilon_n x_n$. Now, observe that $x \in AB' + x_{n+1}$ if and only if $x \in AB'$ or $x_{n+1} \in Cl{\downarrow}(\{x\})$. Similarly, if $\epsilon_{n+1} = -$ then x does not belong to $AB' - x_{n+1}$ if and only if $x_{n+1} \in Cl{\uparrow}(\{x\})$ or $x \notin AB'$. □

Notice that the complexity of the algorithm *ControlAccess* is $O(nh)$, where n is the length of the sequence of updates $\mathbf{a}$ and h is the size of the Hasse diagram H.

This indicates that as the number n grows, there will be a moment when it is worth to replace $\mathbf{a}$ by a shorter sequence of updates representing the same authorization base, for instance, the one given by the antichain of minimal elements of AB.

One should also note that our algorithm gives the priority to checking most recent updates. That is, we always check which of the most recent updates affected the status of x and fall back on previous states of the base if x was nor affected.

To summarize, under the lazy maintenance approach *ControlAccess* takes $O(nh)$ steps and *Grant* and *Revoke* take constant time. The performance of the *ControlAccess* procedure will be further improved in the next section.

5 Implementations when $U = S \times O \times A$

Until now we used an abstract representation of the authorization space U and its ordering $\preceq$ without taking into account the fact that in the context of the model presented in Section 2, U is the product of S, O, and A. Let us recall that in Section 2 $\preceq$ is defined by reference to orderings $\preceq_S$, $\preceq_O$, and $\preceq_A$:

$$(s, o, a) \preceq (s', o', a') \text{ if and only if } (s \preceq_S s') \wedge (o' \preceq_O o) \wedge (a' \preceq_A a)$$

The ordering $\preceq$ is the product of orderings $\preceq_S$, $\preceq_O^{-1}$ and $\preceq_A^{-1}$ (see [3]). Given Hasse diagrams for $\preceq_O$ and $\preceq_A$, the Hasse diagrams for $\preceq_O^{-1}$ and $\preceq_A^{-1}$ can be easily obtained by switching the roles of *in* and *out* lists.

Given the Hasse diagrams of the orderings $\preceq_S$, $\preceq_O^{-1}$ and $\preceq_A^{-1}$, the Hasse diagram of the ordering $\preceq$ can be easily computed. But it should be clear that we do *not* want to maintain the Hasse diagram of $\preceq$ due to its big size. We will see below that maintaining only the Hasse diagrams of the orderings $\preceq_S$, $\preceq_O$ and $\preceq_A$ provides us with enough information to perform depth-first searches needed in the procedures described earlier but is substantially more efficient. We will denote by H_S, H_O and H_A the Hasse diagrams of $\preceq_S$, $\preceq_O$ and $\preceq_A$, respectively. By h_S, h_O and h_A, we denote the sizes of their representations by means of adjacency lists in_S and out_S, in_O and out_O and in_A and out_A, respectively.

We will now describe the relationship between the Hasse diagrams of $\preceq_S$, $\preceq_O$ and $\preceq_A$ and the Hasse diagram of $\preceq$. Namely,

$$(s,o,a) \in in(s',o',a') \text{ if and only if } \quad \begin{array}{l} (s \in in_S(s') \wedge o = o' \wedge a = a') \\ \vee (s = s' \wedge o' \in in_O(o) \wedge a = a') \\ \vee (s = s' \wedge o = o' \wedge a' \in in_A(a)). \end{array}$$

A dual relationship holds for the *out* lists.

It follows that the Hasse diagrams H_S, H_A and H_S allow us to reproduce the lists *in* and *out* for H, in time linear in the size of these lists. In the same time, the total size $h_S + h_O + h_A$ of the representations of H_S, H_A and H_S is substantially smaller than h. It is easy to see that h is of the order

$$O(h_S \cdot h_O \cdot k_A + h_S \cdot k_O \cdot h_A + k_S \cdot h_O \cdot h_A + h_S \cdot h_O \cdot h_A),$$

where $|S| = k_S$, $|O| = k_O$, and $|A| = k_A$. Thus the fact that H may be large is not a problem — all we need to do is to maintain H_S, H_O and H_A.

We will now present a method to improve the efficiency of the algorithms presented earlier in Section 4. Our approach is to precompute the transitive closures of the Hasse diagrams of $\preceq_S$, $\preceq_O$ and $\preceq_A$. This can be done in time $O(k_S \cdot h_S)$ for $\preceq_S$, $O(k_O \cdot h_O)$ for $\preceq_O$ and $O(k_A \cdot h_A)$ for $\preceq_A$. The resulting transitive closures can be stored as adjacency matrices at the total space cost $O(k_S^2 + k_O^2 + k_A^2)$. From this point on we will assume that comparisons $s \preceq_S s'$, $o \preceq_O o'$, and $a \preceq_A a'$ take constant time.

We will now present versions of the algorithms introduced earlier. We will first

consider the case when authorization bases are represented as antichains. Recall that $U = S \times O \times A$ and thus D consists of triples of the form (s, o, a).

ControlAccess$(D, (s', o', a'))$
for $(s, o, a) \in D$ **do**
    if $s \preceq_S s'$ **and** $o' \preceq_O o$ **and** $a' \preceq_A a$ **then return**{TRUE}
return{FALSE}

This algorithm runs in time $O(|D|)$, a substantial improvement over the general version.

Grant$(D, (s', o', a'))$
for $(s, o, a) \in D$ **do**
    if $s \preceq_S s'$ **and** $o' \preceq_O o$ **and** $a' \preceq_A a$ **then stop**
for $(s, o, a) \in D$ **do**
    if $s' \preceq_S s$ **and** $o \preceq_O o'$ **and** $a \preceq_A a'$ **then** $D := D \setminus \{(s, o, a)\}$
$D := D \cup \{(s', o', a')\}$

Also the *Grant* procedure runs in time $O(|D|)$ (rather than in time linear in h).

As concerns the procedure *Revoke*, there is no gain in time efficiency. It still runs in time linear in the size h of the Hasse diagram of $\preceq$. It should be mentioned though that it can be implemented so that it requires only the Hasse diagrams H_S, H_O and H_A.

Finally, we will show that in the case of lazy maintenance, the performance of the procedure *ControlAccess* also improves substantially. We assume here that $\mathbf{a} = ((\epsilon_1, (s_1, o_1, a_1)), \ldots, (\epsilon_n, (s_n, o_n, a_n)))$.

ControlAccess$(\mathbf{a}, (s', o', a'))$
for $m = n$ **downto** 1 **do**
    if $\epsilon_m = +$ **and** $s_m \preceq_S s'$ **and** $o' \preceq_O o_m$ **and** $a' \preceq_A a_m$ **then return**{TRUE}
    if $\epsilon_m = -$ **and** $s' \preceq_S s_m$ **and** $o_m \preceq_O o'$ **and** $a_m \preceq_A a'$ **then return**{FALSE}
return{FALSE}

It is clear that this procedure runs in time $O(n)$ (and not $O(nh)$), as before.

6 Further research

In [6], the authors considered two classifications of authorizations: into positive and negative and into strong and weak. The weak authorizations allow us to handle exceptions. The idea is that weak authorizations are inherited *provided* there is no strong authorization blocking the inheritance. It is possible to provide a semantics for both strong and weak authorizations using Reiter's default logic [8]. We will deal with this problem in a separate paper.

Acknowledgments

Research of the second and third authors has been partially supported by NSF grant IRI-9400568.

References

[1] E. Bertino and L. Martino. Object-Oriented Database Systems: Concepts and Architectures Addison-Wesley Publishing Company, 1993.

[2] E. Bertino, and H. Weigand. An approach to authorization modeling in object-oriented database systems. Data and Knowledge Engineering 12, 1994.

[3] D.A. Davey and H.A. Priestley. Introduction to Lattices and Order. Cambridge University Press, 1990.

[4] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. Database Security. ACM Press, Addison-Wesley Publishing Company, 1995.

[5] T. Lunt. Authorization in Object-Oriented Databases. In: W. Kim, *Modern Database Systems*, pp. 130–145, Addison Wesley, Reading, MA., 1994.

[6] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation databases. ACM Transaction on Database Systems 16:88–131, 1991

[7] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and data bases*, pages 55–76. New York, NY: Plenum Press, 1978.

[8] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.