Decentralized Administration for a Temporal Access Control Model

Elisa Bertino Claudio Bettini
Elena Ferrari Pierangela Samarati

Dipartimento di Scienze dell'Informazione

Università di Milano

Milano, Italy
Email: {ebertino,bettini,ferrarie,samarati}@dsi.unimi.it

Technical Report

176-96

Abstract

In this report we present a temporal access control model that provides for decentralized administration of authorizations. Each access authorization, negative or positive, is associated with a time interval limiting its validity. When the interval expires, the authorization is automatically revoked. The model also permits the specification of rules, based on four different temporal operators, to derive additional authorizations from the presence or absence of other authorizations. Users creating objects can retain complete control over their objects or delegate other users the privilege of administering accesses on the objects. Delegation can also be selectively enforced with reference to specific access modes or time intervals. The resulting model provides a high degree of flexibility and allows to express several protection requirements which cannot be expressed in traditional access control models.

1 Introduction

Access control models available today allow the specification of authorizations stating permissions for users to exercise operations on objects. When a user must be allowed for an operation on an object, an authorization is granted to him. The user is then allowed for the access until the authorization is explicitly removed. This simple paradigm of authorization model, on which almost all the models available today are based, does not fit the real-life situations where more complex security requirements may need to be specified. Organizational policies call for more flexibility and expressiveness in authorization specification. In this respect, exception handling, temporal authorizations, dependency rules among authorizations, explicit denials, task-based authorizations, roles are all needed facilities. In this report we make a step in this direction with respect to the treatment of temporal aspects in the specification of authorizations.

Access control mechanisms provided as part of commercial data management systems and research prototypes [6, 7] do not have temporal capabilities. For example, in a typical Relational DBMS (RDBMS) it is not possible to specify, by using the authorization command language, that a user may access a relation only for a day or a month. If such a need arises, authorization management and access control must be implemented as application programs, thus making authorization management very difficult. However, in many application environments, there is a strong need for authorizations with temporal validity. Consider for example the case of our University. Students which attend a given course must be authorized to access the DBMS but only for the time of their practice.

A temporal authorization model has been presented by us in a previous paper [3]. In that model, authorizations have a temporal dimension, in that each authorization has an associated time interval representing the set of time instants for which the authorization is given. Moreover the model supports the specification of derivation rules expressing temporal dependencies among authorizations. For example, a derivation rule can state that a user can read an object as long as another user can read the same object. Derivation rules allow to derive new authorizations on the basis of the presence (or absence) of other authorizations. Like authorizations, derivation rules have an associated time interval representing the set of time instants in which the derivation rule is applied. The model supports both positive authorizations and negative authorizations. A user owning a negative authorization for an access mode on an object can neither access the object nor grant or revoke other users the access on the object. Like positive authorizations, negative authorizations for an access mode on an object can be granted by the owner of the object as well as by any other user who owns an authorization on the object with the grant option. The capability of supporting explicit denials, provided by negative authorizations, is very useful in a decentralized model [5], where other users, besides the owner of an object, can grant access authorizations on the object.

In this report we address the issue of decentralized administration of authorizations. In our model, the owner can retain complete control over his objects or delegate other users the privilege of administering accesses on the object. Delegation can be enforced by either giving a user the privilege to administer the object, in which case the user can grant and revoke any authorization (explicitly or through rules) on the object, or selectively, by granting authorizations with the grant option. The grant option allows to delegate administration only with reference to specific privileges and to specific time intervals. Although delegation of administrative privileges necessarily implies some loss of control from the owners of the objects, our model still allows the owner to retain some control on the privileges that others will allow to execute on his object: The owner can always issue specific negative authorizations and derivation rules. Thus, the resulting model provides a high degree of flexibility by supporting decentralized authorization administration, coupled with

the possibility of enforcing stricter controls on particular crucial data items, as we will illustrate in Section 6. Revocation of authorizations is recursive: whenever a user revokes an authorization for an object to another user, the authorizations that the revokee has granted thanks to the revoked authorization may need to be removed or their time intervals modified. The revocation is iteratively applied to all the users that received the access authorization from the revokee. The semantics of recursive revocation, proposed by Griffiths and Wade in the framework of the System R database model [10] and formally defined by Fagin [9], is used by current commercial DBMSs [8, 13, 15]. [1] We extend the semantics of the recursive revocation to the consideration of temporal authorizations. Since each authorization has a time interval, a revoke request can cause not only the deletion of other authorizations, besides the ones whose deletion is explicitly required, but also the modification of their time intervals or the splitting of one authorization in several authorizations.

Related work

The need to include time in security specifications and to provide more powerful languages for the specification of authorizations has been recognized by other researchers. Temporal issues have been first addressed in the authentication control of the Kerberos operating system [16]. In this system, the ticket issued to clients for presentation to servers and for verifying that the sender has been recently authenticated by Kerberos, includes an expiration time after which the ticket is not valid anymore. However, time is used in Kerberos only to save the clients from having to require a new ticket at each service request. By contrast, in our approach, time is included in the specification of authorizations and derivation rules against which access control is enforced. Thomas and Sandhu [17] recognize the need, in real-life situations, of expressing rules for the derivation of authorizations on the basis of temporal relationships among authorizations. However, they do not provide any model or framework for the specification of these dependencies. A general language for expressing authorization rules has been proposed by Woo and Lam in [18]. Although their language does not explicitly provide time and temporal relationships, these concepts could be modeled by their rules. Unfortunately, the generality of this language, which has almost the same expressive power of first order logic, impacts efficiency. The tradeoff between expressiveness and efficiency seems to be strongly unbalanced in their approach. A different logic language, based on modal logic, has been proposed by Abadi et Al. in [1]. However, their logic is mainly used to model concepts such as roles and delegation of authorities and their framework does not provide any mechanism to express temporal operators for authorization derivation. In [3] we have presented an authorization model where authorizations, either explicitly specified or derived through rules, have a time interval associated with them. In that model a very limited form of authorization administration was considered. In this report we extend that model with a decentralized policy for the administration of authorizations and formally define the operation for granting and revoking authorizations and derivation rules.

Organization of the report

The remainder of the report is organized as follows. Section 2 illustrates the authorization model. Section 3 deals with the formal semantics of authorizations and rules. The administrative operations supported by our model are described in Section 4. Section 5 formally defines the semantics of the revoke operations in terms of its effects on the authorization state and provides an algorithm to enforce revocation of authorizations. Section 6 illustrates how our model can be used to express,

[1] Although with some variations in the different DBMSs.

in a flexible way, different protection requirements. Section 7 presents some final remarks. Finally, Appendix A proves the correctness of the revoke algorithm presented in the report.

2 The authorization model

In this section we illustrate our authorization model. We do not make any assumption on the underlying data model against which accesses must be controlled and on the access modes users can exercise on the objects of the data model. The choice of the data model and the access modes executable on the objects is to be made when the system is initialized. This generality makes our authorization model applicable to the protection of information in different data models.

In the following U denotes the set of users, O the set of objects, and M the set of access modes. We consider as users the identifiers with which users can connect to the system. We suppose identifiers can refer to single users (e.g., `Ann` or `Bob`) or to user groups (e.g., `staff` or `manager`).

In our model, authorizations do not always exist from the time they are granted to the time of their revocation. We associate with each authorization a temporal constraint representing the set of time instants in which the authorization holds. We refer to authorizations together with their temporal constraint as *temporal authorization*. Beside the specification of (explicit) temporal authorizations, our model allows to specify *derivation rules* from which other authorizations can be derived on the basis of the existing authorizations. Like authorizations, derivation rules have an associated time interval, representing the set of instants in which the derivation rule can be applied.

In the following we assume time to be discrete. In particular, we take as our model of time the natural numbers $\mathbb{N}$ with the total order relation $<$. The results that we show using natural numbers are valid for any set isomorphic to natural numbers.

We start by defining explicit authorizations. We then illustrate the rules for the derivation of authorizations.

2.1 Temporal authorizations

In our model, authorizations can be positive or negative. A positive authorization represents a permission for a user to exercise a privilege on an object. A negative authorization represents a denial for a user to exercise a privilege on an object. Positive authorizations can also be granted with the *grant option*. If a user holds an authorization for a privilege on an object with the grant option, the user can also grant (and revoke) other users authorizations, positive or negative, for the privilege on the object.[2] Authorizations are defined as follows.

Definition 2.1 (Authorization) *An authorization is a 6-tuple* `(s,o,m,pn,g,go)` *where*

`s` $\in U$ *is the subject, i.e., the user to whom the authorization is granted;*

`o` $\in O$ *is the object to which the authorization refers;*

`m` $\in M$ *is the access mode, or privilege, for which the authorization is granted;*

`pn` $\in \{+, -\}$ *indicates whether the authorization is positive (+) or negative (-);*

`g` $\in U$ *is the user who granted the authorization;*

[2] For simplicity, we consider that the grant option can only be associated to positive authorizations and that users holding a positive authorization for a privilege on an object with the grant option can also grant negative authorizations for the privilege on the object. The model could be easily extended to the consideration of two different types of authorizations for the administration of positive or negative authorizations.

go $\in$ {yes, no} *indicates whether* s *has the grant option for* m *on* o.

Tuple (s,o,m,pn,g,go) states that user s can exercise (if pn="+") or cannot exercise (if pn="−") access mode m on object o and that this authorization was granted by user g. If go="yes", s can also grant/revoke other users authorizations (positive or negative), and the grant option on them, for m on o. Since only positive authorizations can be granted with the grant option, authorizations with pn = "−" have necessarily go = "no".

For instance, tuple (Ann,o_1,write,+,Tom,yes) states that Ann is authorized by Tom to write object o_1. Ann is also entitled to grant other users the write access mode on o_1 as well as the grant option on it, and to prevent other users from writing o_1, by granting them a negative authorization. Tuple (Bob,o_1,read,−,Ann,no) states that Bob is forbidden by Ann to read object o_1.

As already stated, our model considers temporal authorizations, i.e., authorizations together with a time interval of validity. Temporal authorizations are defined as follows.

Definition 2.2 (Temporal authorization) *A temporal authorization is a triple* (ts, time, auth)*, where* ts $\in \mathbb{N}$ *is the time at which the authorization was granted* [3]*,* time *is a time interval* [t_i,t_j]*, with* $t_i \in \mathbb{N}$, $t_j \in \mathbb{N} \cup \infty$, ts $\leq t_i \leq t_j$*, and* auth *is an authorization.*

Temporal authorization (ts,[t_i,t_j],auth) states that authorization auth, specified at time ts is granted for the time interval [t_i,t_j]. We require the starting time of the authorization to be greater than or equal to the time at which the authorization is granted ($t_i \geq$ts), i.e., it is not possible to specify retroactive authorizations.

In the following, given a temporal authorization A = (ts,[t_i,t_j],(s,o,m,pn,g,go)), s(A), o(A),m(A),pn(A),g(A),go(A) denote respectively the subject, the object, the privilege, the sign, the grantor, and the grant option in A. Moreover, ts(A) denotes the time when A has been granted and [t_i(A),t_j(A)] the temporal validity of A.

For example the temporal authorization (5,[10,40],(Alice,o_1,write,+,Bob,yes)) states that at time 5 Bob granted Alice the authorization to write object o_1 between instants 10 and 40. Since the authorization is with the grant option, Alice can also grant other users positive or negative authorizations for the write privilege on object o_1 for time intervals in [10,40].

Note that a user can only grant privileges he owns. Then, if a user holds an authorization for an access mode on an object with the grant option for a time interval [t_i,t_j], his privilege to authorize or deny other users to exercise the access mode on the object is limited to the interval [t_i,t_j]. For instance, with reference to the abovementioned authorization, all authorizations to read o_1 granted by Alice must have a time interval included in [10,40]. The time interval associated with an authorization is specified by the grantor at the time the authorization is issued. If nothing is specified the whole interval for which the authorization can be granted is taken. For instance, with reference to the authorization above, the time interval [10,40] will be considered in case no further restriction is specified by Alice.

2.2 Derivation rules

Additional authorizations can be derived from the authorizations explicitly specified. The derivation of authorizations is based on temporal propositions, used as rules, which allow new temporal authorizations to be derived on the basis of the presence or the absence of other temporal authorizations. Derivation rules can be applied to positive as well as to negative authorizations. Like

[3] Timestamps are introduced to prevent cycles among authorizations [10].

authorizations, derivation rules have a time interval. The time interval associated with a derivation rule indicates the set of time instants for which the rule applies.

Derivation rules are defined as follows.

Definition 2.3 (Derivation rule) *A derivation rule is defined as* $([t_i,t_j],A_l\langle op\rangle A_r)$*, where* $[t_i,t_j]$ *is a time interval,* $t_i \in \mathbb{N}$, $t_j \in \mathbb{N} \cup \infty$, $t_i \leq t_j$, A_l *and* A_r *are authorizations,* $g(A_l)$ *is the user who specifies the rule,* $go(A_l) =$ "no"*, and* $\langle op\rangle$ *is one of the following operators:* WHENEVER, ASLONGAS, WHENEVERNOT, UNLESS.

Authorizations derived from derivation rules have as grantor the user who specifies the rule. Note that for sake of simplicity, we restrict rules to the derivation of authorizations without the grant option. Indeed, allowing grant option in rules would make authorization management cumbersome.

Derivation rules can also be parametric, meaning that not all elements in the two authorizations of the rules are specified. The only element that cannot be parameterized is the sign of authorizations. If a rule is parametric with respect to an element, the metacharacter "*", meaning any value, appears instead of a specific value for the element. We require that each time the symbol "*" appears for either the subject, the object, or the access mode, of any of the two authorizations in a rule, then it must also appear for the corresponding element of the other authorization. By contrast, since we require derived authorizations to be without grant option and the grantor of a derived authorization to be the user who specified the rule, symbol "*" cannot appear for grantor and grant-option in the authorization on the left of the operator. However, it can be used in the authorization on the right of the operator to denote any value for grantor or grant-option. Each parametric rule is resolved by the system in several derivation rules for each possible value of the elements for which symbol "*" is used. In the derived derivation rule, every time a value is substituted for "*" with reference to either the subject, the object, or the access mode, the same value is substituted for the same element in the other authorization.

The intuitive semantics of derivation rules is as follows:

- $([t_i,t_j],A_1$ WHENEVER $A_2)$. We can derive A_1 for each instant in $[t_i,t_j]$ for which A_2 is given or derived. For example, rule R_1 in Figure 1, specified by `Tom`, states that every time in `[10,90] staff` can read object `bulletin`, thanks to an authorization granted by `Tom`, also `secretarial-staff` can read `bulletin`.

- $([t_i,t_j],A_1$ ASLONGAS $A_2)$. We can derive A_1 for each instant t in $[t_i,t_j]$ such that A_2 is either given or derived for each instant from t_i to t. Unlike the WHENEVER operator, the ASLONGAS operator does not allow to derive A_1 at an instant t in $[t_i,t_j]$ if there exists an instant t', $t_i \leq t' \leq t$, such that A_2 is not given and cannot be derived at t'. For example, rule R_2 in Figure 1, specified by `Tom`, states that, until time `90`, `temporary-staff` is authorized on object `bulletin` for all access modes for which `staff` has been continuously authorized from time `10`.

- $([t_i,t_j],A_1$ WHENEVERNOT $A_2)$. We can derive A_1 for each instant in $[t_i,t_j]$ for which A_2 is neither given nor derived. For example, rule R_3 in Figure 1, specified by `Tom`, states that `staff-A` can write object `staff-document` for every instant in $[30,\infty]$ in which `staff-B` is not authorized to write it by `Tom`.

- $([t_i,t_j],A_1$ UNLESS $A_2)$. We can derive A_1 for each instant t in $[t_i,t_j]$ such that A_2 is neither given nor can be derived for each instant from t_i to t. Unlike the WHENEVERNOT, the UNLESS operator does not allow to derive A_1 at an instant t in $[t_i,t_j]$ if there exists an

instant t', $\mathtt{t}_i \leq t' \leq t$, such that $\mathtt{A}_2$ is given or derived at t'. For example, suppose that at time `40` `new-staff` cannot read object `worksheet`. Then rule $\mathtt{R}_4$ in Figure 1 states that `staff` can read object `worksheet` until the minimum between time `300` and the time at which the authorization for `new-staff` will start to hold.

Note that, the temporal operators WHENEVER and WHENEVERNOT allow to derive an authorization at a given time instant on the basis of the presence (or absence) of another temporal authorization at the same instant, whereas ASLONGAS and UNLESS allow to derive an authorization at a given instant on the basis of the presence (or absence) of another authorization at the same and in past time instants. The authorization must be present (or absent) continuously from the starting time of the derivation rule to the time of the derivation.

In the following, we also refer to WHENEVERNOT and UNLESS rules, which allow to derive authorizations from the absence of other authorizations, as negative rules.

```
(A1) (5,([10,40], (staff,bulletin,read,+,Tom,yes)))

(A2) (10,([10,50], (staff-B,staff-document,write,+,Tom,yes)))

(A3) (10,([80,90], (staff-B,staff-document,write,+,Tom,yes)))

(A4) (20,([50,100], (staff,bulletin,read,+,Tom,yes)))

(A5) (60,([120,∞], (new-staff,worksheet,write,+,Bob,yes)))

(R1) ([10,90], (secretarial-staff,bulletin,read,+,Tom,no) WHENEVER
  (staff,bulletin,read,+,Tom,*))

(R2) ([10,90], (temporary-staff,bulletin,*,+,Tom,no) ASLONGAS
  (staff,bulletin,*,+,Tom,*))

(R3) ([30,∞],(staff-A,staff-document,write,+,Tom,no) WHENEVERNOT
  (staff-B,staff-document,write,+,Tom,*))

(R4) ([40,300], (staff,worksheet,write,+,Bob,no) UNLESS
  (new-staff,worksheet,write,+,*,*))

(R5) ([20,∞], (consultant,*,*,-,Bob,no) WHENEVER (temporary-staff,*,*,+,Tom,*))
```

Figure 1: An example of authorizations and derivation rules

Example 2.1 Consider the authorizations and derivation rules illustrated in Figure 1. The following temporal authorizations can be derived:

- `([10,40], (secretarial-staff,bulletin,read,+,Tom,no))`, and `([50,90], (secretarial-staff,bulletin,read,+,Tom,no))`, from authorizations $\mathtt{A}_1$ and $\mathtt{A}_4$ and rule $\mathtt{R}_1$.
- `([10,40], (temporary-staff,bulletin,read,+,Tom,no))` from authorization $\mathtt{A}_1$ and rule $\mathtt{R}_2$.

- ([51,79], (staff-A,staff-document,write,+,Tom,no)), and
  ([91,∞], (staff-A,staff-document,write,+,Tom,no)), from authorizations A_2 and A_3 and rule R_3.
- ([40,119], (staff,worksheet,write,+,Bob,no)) from authorization A_5 and rule R_4.
- ([20,40], (consultant,bulletin,read,-,Bob,no)) from authorization A_1 and rules R_2 and R_5.

□

3 Semantics of authorizations and rules

The possibility of specifying both positive and negative authorizations introduces potential conflicts among authorizations. A positive authorization states that a user can exercise a privilege on an object whereas a negative authorization states he cannot. A conflict therefore would arise in case a user holds both a positive and a negative authorization for a privilege on an object for a given time interval. We solve conflicts due to the simultaneous presence of positive and negative authorizations according to the denials-take-precedence principle. Then, whenever a user has both a positive and a negative authorization for a given access, the access will be denied. The positive authorization, although still present in the system cannot be used.

Example 3.1 Consider the two authorizations:
(A_1) (5,[40,100],(Bob,o_2,write,+,Ann,yes)) and
(A_2) (20,[50,70],(Bob,o_2,write,−,Tom,no))
In the time interval [50,70] the negative authorization overrides the positive authorization. Then, Bob will be allowed to write o_2 only in the time intervals [40,49] and [71,100]. □

Before illustrating the semantics of authorizations and rules we introduce the concept of *temporal authorization base* as the set of authorizations and rules present at a given time in the system.

Definition 3.1 (Temporal Authorization Base) *A Temporal Authorization Base* (TAB) *is a set of temporal authorizations and derivation rules.*

The semantics of a TAB is given as a set of clauses in a *general logic program* [12, page 52]. Table 1 illustrates the clause/set of clauses corresponding to each type of authorization/rule considered by our model.

The set of clauses reported in Table 1 represents an extension to the semantics given in [3]. All the results obtained in [3] are easily applied to the current authorization model. We use a logic with two sorts, the natural numbers ($\mathbb{N}$) as a temporal sort and a generic domain ($\mathcal{D}$) as the other sort. The language includes constant symbols $1, 2, \ldots$ for natural numbers, a finite set of constant symbols (e.g. $\mathtt{s}_1, \mathtt{o}_1, \mathtt{m}_1, \mathtt{g}_1, -, +, \mathtt{go}_1, \mathtt{s}_2, \ldots$) for elements in $\mathcal{D}$, and temporal variable symbols t, t', t''. Predicate symbols include the temporal predicate symbols $\leq$ and $<$ with the fixed interpretation of the corresponding order relation on natural numbers, and the predicate symbols $F()$,$F_N()$,$F_P()$ and $G()$. Intuitively, the predicate $F()$ is used to represent the authorizations at specific instants. The fact that $F(t,\mathtt{A})$ is true in an interpretation corresponds to the validity of A at instant t according to that interpretation. The predicates $G(), F_N()$ and $F_P()$ are auxiliary predicates, used to avoid quantification. Intuitively, $G(t, \mathtt{s}, \mathtt{o}, \mathtt{m})$ is true in an interpretation if there is at least one negative authorization, with the same s,o,m, valid at instant t according to

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s},\mathtt{o},\mathtt{m},-,\mathtt{g},\mathtt{no})$:
$F(t,\mathtt{s},\mathtt{o},\mathtt{m},-,\mathtt{g},\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s},\mathtt{o},\mathtt{m},+,\mathtt{g},\mathtt{go})$:
$F(t,\mathtt{s},\mathtt{o},\mathtt{m},+,\mathtt{g},\mathtt{go}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ \neg G(t,\mathtt{s},\mathtt{o},\mathtt{m})$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no})$ WHENEVER $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no})$ WHENEVER $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,\ \neg G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1)$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no})$ ASLONGAS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,\ \neg F_N(\mathtt{t_i},t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no})$ ASLONGAS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,$
$\neg F_N(\mathtt{t_i},t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,\ \neg G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1)$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no})$ WHENEVERNOT $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ \neg F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no})$ WHENEVERNOT $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ \neg F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,\ \neg G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1)$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no})$ UNLESS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ \neg F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,\ \neg F_P(\mathtt{t_i},t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$

$[\mathtt{t_i},\mathtt{t_j}]$, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no})$ UNLESS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1,\mathtt{no}) \leftarrow \mathtt{t_i} \leq t \leq \mathtt{t_j}\ ,\ \neg F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,$
$\neg F_P(\mathtt{t_i},t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2,\mathtt{go})\ ,\ \neg G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1)$

$F_P(t'',t,\mathtt{s},\mathtt{o},\mathtt{m},\mathtt{pn},\mathtt{g},\mathtt{go}) \leftarrow t'' \leq t' < t\ ,\ F(t',\mathtt{s},\mathtt{o},\mathtt{m},\mathtt{pn},\mathtt{g},\mathtt{go})$

$F_N(t'',t,\mathtt{s},\mathtt{o},\mathtt{m},\mathtt{pn},\mathtt{g},\mathtt{go}) \leftarrow t'' \leq t' < t\ ,\ \neg F(t',\mathtt{s},\mathtt{o},\mathtt{m},\mathtt{pn},\mathtt{g},\mathtt{go})$

$G(t,\mathtt{s},\mathtt{o},\mathtt{m}) \leftarrow F(t,\mathtt{s},\mathtt{o},\mathtt{m},\mathtt{-},\mathtt{g},\mathtt{no})$

Table 1: Semantics of temporal authorizations and rules

that interpretation. $F_N(t'', t, \mathtt{A})$ is true in an interpretation if there is at least an instant t', with $t'' \leq t' < t$, at which authorization A is false according to that interpretation. Finally, $F_P(t'', t, \mathtt{A})$ is true in an interpretation if there is at least an instant t', with $t'' \leq t' < t$, at which authorization A is true according to that interpretation. The resulting language is very similar to the temporal deductive language proposed in [2], the main difference being the negation in our rules.

The presence of negative derivation rules, that is, rules involving the WHENEVERNOT and UNLESS operator, introduces the problem of generating a unique set of authorizations from a given set of authorizations and rules. The set of derived authorizations could depend on the evaluation order, as illustrated by the following example.

Example 3.2 Consider a TAB containing the following rules:
($\mathtt{R}_1$) ($[10, 100], \mathtt{A}_1$ WHENEVERNOT $\mathtt{A}_2$)
($\mathtt{R}_2$) ($[10, 100], \mathtt{A}_2$ WHENEVERNOT $\mathtt{A}_1$)
Suppose that there are no explicit authorizations in TAB. If we consider first $\mathtt{R}_1$ and then $\mathtt{R}_2$ we derive $(\mathtt{[10,100]}, \mathtt{A}_1)$. By contrast, if we consider first $\mathtt{R}_2$ and then $\mathtt{R}_1$ we derive $(\mathtt{[10,100]}, \mathtt{A}_2)$.
□

From the point of view of the semantics, the property of always having a unique set of valid authorizations is guaranteed only if there exists a *unique* model of the program corresponding to the TAB. To solve these difficulties we have introduced appropriate syntactic restrictions on rules [3], to avoid set of rules as the one shown in the example above. Intuitively, those restrictions avoid recursion on negative rules. We have proposed an approach to stratify authorizations and derivation rules, that is, we have defined an evaluation order for authorizations and rules, and an algorithm to detect if a given TAB can be stratified. We have proved that if a TAB can be stratified, then the corresponding program has a *unique* model. The formal treatment given in [3] can be easily extended to the consideration of authorizations with the grant option.

4 Authorization administration

The user creating an object receives the `own` privilege on it. As owner, the the user to grant/revoke other users authorizations on the object either explicitly or through rules. The owner of an object can also delegate other users the privilege to administer authorizations on the object. Two different administrative privileges are considered: `refer` and `administer`. If a user has the `refer` privilege on an object, he can specify derivation rules in which the object appears in the authorization on the right of the temporal operator. If a user has the administer privilege on an object he can grant to and revoke from other users authorizations (negative or positive and with or without the grant option) on the object either explicitly or through rules.

Decentralized administration of authorizations can also be selectively granted on single privileges, through the use of the grant option. Note however that users holding the grant option for a privilege on an object can grant only explicit authorizations for the privilege on the object; they are not allowed to specify rules for the derivation of these authorizations. The reason for this restriction is that rules can be very powerful and computationally expensive. Rules for the derivation of authorizations on an object can be specified only by users holding either the `own` or the `administer` privilege on the object.

Granting and revoking authorizations and administrative privileges are enforced through administrative operations. Administrative operations allow users to add and remove temporal authorizations and derivation rules, and to grant or revoke other users administrative privileges on the objects. Each temporal authorization and each derivation rule in the TAB is identified by a

unique label assigned by the system at the time of its insertion. The label allows the user to refer to a specific temporal authorization or derivation rule upon execution of administrative operations. In the following we discuss the administrative operations supported by our model. The syntax of the operations in BNF form is given in Figure 2. With reference to the figure, non terminal symbols ⟨subject⟩, ⟨object⟩,⟨access-mode⟩, ⟨auth-t⟩, ⟨grant-option⟩, and ⟨nat-number⟩ represent elements of the domains U, O, M, $\{+,-\}$, $\{\texttt{yes},\texttt{no}\}$ and $\mathbb{N}$ respectively. Non terminal symbols ⟨aid⟩ and ⟨rid⟩ represent system labels. Symbol # can be used in the specification of the starting time for an authorization/rule to indicate the time at which the administrative request is submitted to the system.

⟨grant⟩	::=	GRANT ⟨access-mode⟩ ON ⟨object⟩ TO ⟨subject⟩ FROMTIME ⟨start-time⟩ TOTIME ⟨end-time⟩
⟨deny⟩	::=	DENY ⟨access-mode⟩ ON ⟨object⟩ TO ⟨subject⟩ FROMTIME ⟨start-time⟩ TOTIME ⟨end-time⟩
⟨revoke⟩	::=	REVOKE ⟨aid⟩ | REVOKE ⟨access-mode⟩ ON ⟨object⟩ FROM ⟨subject⟩ FROMTIME ⟨start-time⟩ TOTIME ⟨end-time⟩ REVOKE NEGATION ⟨access-mode⟩ ON ⟨object⟩ FROM ⟨subject⟩ FROMTIME ⟨start-time⟩ TOTIME ⟨end-time⟩
⟨add-rule⟩	::=	ADDRULE ⟨subj⟩ ⟨obj⟩ ⟨acc-mod⟩ ⟨auth-t⟩ ⟨temp-operator⟩ ⟨subj⟩ ⟨obj⟩ ⟨acc-mod⟩ ⟨auth-t⟩ ⟨subj⟩ ⟨grant-op⟩ FROMTIME ⟨start-time⟩ TOTIME ⟨end-time⟩
⟨drop-rule⟩	::=	DROPRULE ⟨rid⟩
⟨grant-adm⟩	::=	GRANTADM ON ⟨object⟩ TO ⟨subject⟩
⟨revoke-adm⟩	::=	REVOKEADM ON ⟨object⟩ FROM ⟨subject⟩
⟨grant-ref⟩	::=	GRANTREF ON ⟨object⟩ TO ⟨subject⟩
⟨revoke-ref⟩	::=	REVOKEREF ON ⟨object⟩ FROM ⟨subject⟩
⟨temp-operator⟩	::=	WHENEVER | ASLONGAS | WHENEVERNOT | UNLESS
⟨subj⟩	::=	subject | *
⟨obj⟩	::=	object | *
⟨acc-mod⟩	::=	access-mode | *
⟨grant-op⟩	::=	grant-option | *
⟨start-time⟩	::=	# | ⟨nat-number⟩
⟨end-time⟩	::=	∞ | ⟨nat-number⟩ | +⟨nat-number⟩

Figure 2: Syntax of administrative operations

Administrative operations can be classified into three groups:

- **Operations involving explicit authorizations**
  These operations allow users to grant or revoke explicit authorizations on an object. The user issuing these operations them must have the `own` or the `administer` privilege on the object or an authorization for the privilege on the object with the grant option. Note that, if a user owns the authorization for an access mode on an object with the grant option for the interval $[t_i, t_j]$, he can authorize or deny other users the access mode on the object but only for the interval $[t_i, t_j]$. Moreover, a user can revoke only authorizations and rules he granted. The grant and revoke operations are as follows.

Grant privilege: GRANT m ON o TO s FROMTIME t_i TOTIME t_j

To grant subject s the authorization for the access mode m on object o for the time interval [t_i,t_j]. The starting time of the authorization must be greater than or equal to the time at which the authorization is inserted, i.e., it is not possible to specify retroactive authorizations.

Deny privilege: DENY m ON o TO s FROMTIME t_i TOTIME t_j

To deny subject s the access mode m on object o for the time interval [t_i,t_j]. The deny operation results in the addition of a new explicit temporal negative authorization.

Revoke authorization: REVOKE aid

To revoke the temporal authorizations whose label is aid. The specified authorization is removed from TAB. The user issuing the revoke request must be the user appearing as grantor on the authorization.

Revoke privilege: REVOKE m ON o FROM s FROMTIME t_i TOTIME t_j

To revoke from subject s access mode m on object o for the time interval [t_i,t_j]. It results in the deletion or modification (to exclude interval [t_i,t_j]) of all the temporal authorizations of s for m on o, granted by the user requesting the revoke operation. If the authorizations revoked or modified are with the grant option, the authorizations granted by the revokee may need to be reconsidered. We discuss the semantics of the revocation and the methods to implement it in Section 5.

Revoke negation: REVOKE NEGATION m ON o FROM s FROMTIME t_i TOTIME t_j

To revoke from subject s the negation for the access mode m on object o for the time interval [t_i,t_j]. The revoke operation results in the deletion or modification (to exclude the interval [t_i,t_j]) of all the negative temporal authorizations of s for m on o, granted by the user issuing the revoke operation. Since negative authorizations cannot be granted with the grant option, there is no need to propagate the effect of the revocation.

- **Operations involving rules**

These are requests for creating or deleting rules. The user invoking these operations must have either the own or the administer privilege on the object appearing at the left of the temporal operator and either the own, administer, or refer privilege on the object appearing at the right of the temporal operator.

Add rule: ADDRULE s_1 o_1 m_1 pn_1 operator s_2 o_2 m_2 pn_2 g go FROMTIME t_i TOTIME t_j

To add rule ([t_i,t_j],(s_1,o_1,m_1,pn_1,g_1,no) operator (s_2,o_2,m_2,pn,g,go)) to TAB, where g_1 is the user who issued the request. The grantor of the authorization appearing at the left of the temporal operator, that is, g_1 identifies the user inserting the rule. Like for authorizations, the starting time of the interval associated with the rule must be greater than the time at which the request is specified.

Drop rule: DROPRULE rid

To drop the derivation rule labeled rid. The user issuing the request must be the user appearing as grantor in the authorization on the left of the operator in the rule.

- **Operations involving administrative privileges**

These are requests for granting or revoking administrative privileges on an object. They can be executed only by the owner of the object.

Grant administer: GRANTADM ON o TO s

To grant the `administer` privilege on object `o` to subject `s`.

Revoke administer: REVOKEADM ON o FROM s

To revoke the `administer` privilege on object `o` to subject `s`. All the authorizations on `o` and all the derivation rules where `o` appears in the authorization at the left of the operator specified by `s` are deleted. If `s` does not have the reference privilege on `o`, also the derivation rules where `o` appears in the authorization at the right of the operator are deleted.

Grant refer: GRANTREF ON o TO s

To grant the `refer` privilege on object `o` to subject `s`.

Revoke refer: REVOKEREF ON o FROM s

To revoke the `refer` privilege on object `o` from subject `s`. All the derivation rules granted by `s` where `o` appears in the authorization at the right of the operator are deleted, if `s` does not have the administer privilege on `o`.

5 Revocation of authorizations

In our model, the revoke operation can be required for a single authorization, by specifying its label, or for an access mode on an object, with respect to a given time interval. In the following we consider the case of the revocation of an access mode on an object for a given time interval. All the results shown for this case apply to the revocation of specific authorizations as well.

Suppose a user revokes an access mode on an object for a given time interval from another user. The TAB resulting from the revoke operation has to be as if the revokee had never received by the revoker an authorization for the access mode on the object for the interval specified in the revoke request. More precisely, the semantics of the revocation of access mode `m` on object `o` from user `y` by user `x` in the interval $[t_1, t_2]$ is:

(i) to modify or revoke the authorizations that `x` had granted to `y` to exclude the interval $[t_1, t_2]$, and

(ii) to modify or revoke the authorizations in TAB to exclude from their time intervals the time instants in which they would not have existed if `x` had never granted to `y` an authorization for `m` on `o`, for the time instants eliminated by step *(i)*.

In the following we represent the sequence of grant operations for an access mode on an object by a labeled graph, where each node represents a user and an arc between node u_1 and u_2 indicates that user u_1 granted u_2 the access mode. Every arc is labeled with a 5-tuple `(id,timestamp,interval, sign,grant-op)`, where: `id` is the identifier of the authorization; `timestamp` is the time when the authorization was granted; `interval` is the time interval of the authorization; `sign` is the sign ('+','−') of the authorization; and `grant-op` indicates whether the authorization is with the grant option. We use symbol '`g`' to indicate that the authorization is with the grant option and nothing if the authorization is without grant option. A node with no incoming arcs denotes the owner of the object or one of its administrators.

The revocation of an authorization may imply: deleting temporal authorizations, modifying the time interval associated with authorizations, or splitting temporal authorizations in several temporal authorizations (the last operation can be necessary when a subset of the instants associated with a temporal authorization needs to be excluded). The following example illustrates a case of revoke operation.

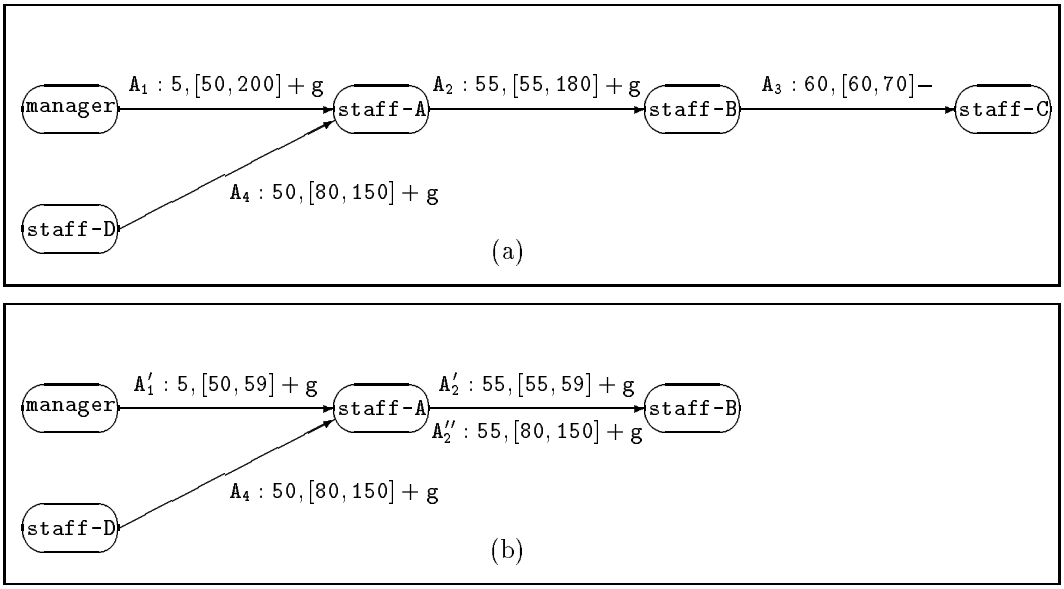

Figure 3: An example of revoke operation.

Example 5.1 Consider a TAB consisting of the following authorizations:

(A_1) (5,[50,200],(staff-A,o,read,+,manager,yes))
(A_2) (55,[55,180],(staff-B,o,read,+,staff-A,yes))
(A_3) (60,[60,70],(staff-C,o,read,-,staff-B,no))
(A_4) (50,[80,150],(staff-A,o,read,+,staff-D,yes))

The corresponding graph is illustrated in Figure 3(a). Suppose that, at time 52, `manager` issues the following command:

REVOKE `read` ON `o` FROM `staff-A` FROMTIME 60 TOTIME 200

According to the semantics of the revoke operation, interval `[60,200]` must be removed from all positive authorizations for the `read` privilege on `o` granted by `manager` to `staff-A`. In case the modified authorizations are with the grant option, the revocation has to be propagated also on the authorizations that `staff-A` granted, and recursively on the authorizations granted by any users for which some authorizations with the grant option are modified/deleted. The resulting effect on the TAB has to be as if `staff-A` had never received the `read` privilege on `o` from `manager` for the interval `[60,200]`.

Let us now illustrate the effect of the revoke operation. The only authorization for the `read` privilege on `o` granted by `manager` to `staff-A` is authorization A_1. As required by `manager`, this authorization is modified to exclude time interval `[60,200]`. Since the authorization is with the grant option, also the authorizations granted by `staff-A` have to be reconsidered. In doing so, the fact that `staff-A` has also other authorizations for the `read` privilege on `o` with the grant option (authorization A_4 granted by `staff-D`) must be taken into account. After A_1 is modified, `staff-A` still remains with the privilege of granting authorizations for the `read` privilege on `o` but only for the intervals `[50,59]` (thanks to `manager`) and `[80,150]` (thanks to `staff-D`)[4]. The authorizations granted by `staff-A` must therefore be restricted to these intervals. Accordingly,

[4]Note that if `manager` wants to strictly forbid `staff-A` to read `o` in interval `[60,200]`, he can enter a negative authorization.

authorization A_2 is modified to exclude time intervals [60,79] and [151,180]. This causes the splitting of the authorization in two authorizations, one for interval [55,59] and the other for the interval [80,150]. Since authorization A_2 was with the grant option, again the effect must be propagated and the time instants deleted from A_2 must be deleted from the authorizations staff-B has granted. The only authorization granted by staff-B is authorization A_3 whose interval must be completely excluded. As a consequence the authorization is deleted. The TAB' resulting after the revoke operation, illustrated in Figure 3(b), is composed of the following authorizations:

(A_1') (5,[50,59],(staff-A,o,read,+,manager,yes))
(A_2') (55,[55,59],(staff-B,o,read,+,staff-A,yes))
(A_2'') (55,[80,150],(staff-B,o,read,+,staff-A,yes))
(A_4) (50,[80,150],(staff-A,o,read,+,staff-D,yes)). □

Before formally introducing the semantics of revocation, we need some preliminary definitions on authorizations of TAB.

Definition 5.1 (Supporting authorization) *Let* A_1 *and* A_2 *be two authorizations. We say that* A_1 *supports* A_2 *at time* t, $t \in [t_i(A_2), t_j(A_2)]$, *(written* $A_1 \xrightarrow{t} A_2$*), iff:*

- A_1 *and* A_2 *are authorizations for the same access mode on the same object:* $m(A_1) = m(A_2)$, $o(A_1) = o(A_2)$;
- *the subject of* A_1 *is the grantor of* A_2*:* $s(A_1) = g(A_2)$;
- *the timestamp of* A_1 *is smaller than the timestamp of* A_2*:* $ts(A_1) < ts(A_2)$;
- *authorization* A_1 *is with the grant option:* $go(A_1) =$ "yes";
- *time instant* t *belongs to the time interval of authorization* A_1*:* $t \in [t_i(A_1), t_j(A_1)]$.

With reference to the TAB illustrated in Figure 3(a), $\forall t \in [55, 180]$ $A_1 \xrightarrow{t} A_2$, $\forall t \in [60, 70]$ $A_2 \xrightarrow{t} A_3$, and $\forall t \in [80, 150]$ $A_4 \xrightarrow{t} A_2$.

A sequence of authorizations each one supporting the next is called chain and is defined as follows.

Definition 5.2 (Supporting chain) *Assume that* $A_1, \ldots, A_n$ *are authorizations for access mode* m *on object* o. *We say that* $\langle A_1, \ldots, A_n \rangle^t$, $n \geq 1$, *is a supporting chain for* A_n *at time* t, *iff the grantor of* A_1 *has either the own or the administer privilege on object* o, *and* $A_1 \xrightarrow{t} A_2 \xrightarrow{t}, \ldots, \xrightarrow{t} A_n$.

With reference to Figure 3(a), $\forall t \in [55, 180]$ $\langle A_1, A_2 \rangle^t$ is a supporting chain for A_2, $\forall t \in [80, 150]$ $\langle A_4, A_2 \rangle^t$ is a supporting chain for A_2, $\forall t \in [60, 70]$ $\langle A_1, A_2, A_3 \rangle^t$ is a supporting chain for A_3.

Each authorization A for a privilege on an object must either be granted by the object's owner, by any of the object's administrators, or by a user who holds the authorization for the privilege on the object with the grant option for all time instants in $[t_i(A), t_j(A)]$. An authorization satisfying these requirements is said to be *legal*.

Definition 5.3 (Legal authorization) *An authorization* A *in* TAB *is legal, iff* $\forall t \in [t_i(A), t_j(A)]$ *there exists a supporting chain* $\langle A_1, \ldots, A_n, A \rangle^t$ *for* A *with* $A_1, \ldots, A_n$ *in* TAB.

Note that several supporting chains can be present to make a single authorization legal.

In the following, we use a set of disjoint[5] intervals $T = \{[t_i, t_j], \ldots, [t_r, t_s]\}$ as a compact notation for the set of natural numbers included in these intervals. Hence, the operations of union ($T_1 \cup T_2$), intersection ($T_1 \cap T_2$), difference ($T_1 \setminus T_2$), and inclusion ($I_1 \subseteq I_2$) have the usual semantics of set operations. However, we implement those operations so that they can be performed using intervals and giving the result as a set of disjoint intervals. We use two kinds of set membership: $t \in T$ is true if t is one of the natural numbers represented by T, $[t_i, t_j] \in T$ is true if the interval $[t_i, t_j]$ is exactly one of the disjoint intervals of T.

To formalize the semantics of the revoke operation we use function '*Delete*()' that takes as argument two sets of authorizations S_1 and S_2. For each authorization A in S_2, the function checks if an authorization A′ exists in S_1 having the same timestamp, subject, object, access mode, sign, grantor and grant option as A, and whose time interval is not disjoint from that of A. If the time intervals of A and A′ coincide, then A′ is removed from S_1. Otherwise, A′ is replaced by a set of authorizations which differ from A′ only for their time intervals, which are the elements of $\{[t_i(A'), t_j(A')] \setminus [t_i(A), t_j(A)]\}$.

In the following we use the notation $\langle x, m, o, y, t_1, t_2 \rangle$ to denote a request by user x to revoke access mode m on object o from user y, from time t_1 to time t_2. We formalize the semantics of the revoke operation by a function named '*rvk*()', defined as follows.

Definition 5.4 (Rvk function) *Given a* TAB *containing only legal authorization, let* $\langle x, m, o, y, t1, t2 \rangle$ *be a request for revocation of access mode* m *on object* o. *Function rvk*() *generates a new temporal authorization base* TAB′ *defined as:*
TAB′ = *Delete(*TAB, *(REV(*TAB*)*$\cup$*RREV(*TAB*))), where:*
*REV(*TAB*)* = $\{(ts, [t'_i, t'_j], auth) \mid \exists\, (ts, [t_i, t_j], auth) \in$ TAB, *where subject,object, access mode, sign and grantor in* auth *are* y,o,m,'+',x, *respectively, and* $[t'_i, t'_j] = ([t_i, t_j] \cap [t1, t2]) \neq \emptyset\}$.
*RREV(*TAB*)* = $\{(ts, [t'_i, t'_j], auth) \mid \exists\, A=(ts, [t_i, t_j], auth) \in$ TAB, $[t'_i, t'_j] \subseteq [t_i, t_j]$, $[t'_i, t'_j] \neq \emptyset$, *and* $\forall t \in [t'_i, t'_j] \nexists\, \langle A_1, \ldots, A_n, A \rangle^t$, *with* $A_1, \ldots, A_n, A \in$ *Delete(*TAB, *REV(*TAB*))*$\}$.

REV(TAB) denotes the set of authorizations in TAB whose revocation is explicitly required, whereas *RREV*(TAB) (Recursive *REV*) denotes the set of authorizations that, after deleting the authorizations in *REV*(TAB), are not legal in TAB.

Example 5.2 Consider the TAB and the revoke operation of Example 5.1.

REV(TAB) = {(5,[60,200],(staff-A,o,read,+,manager,yes))}.
RREV(TAB) = {(55,[60,79],(staff-B,o,read,+,staff-A,yes)),
(55,[151,180],(staff-B,o,read,+,staff-A,yes)),
(60,[60,70],(staff-C,o,read,−,staff-B,no))}.
TAB′ = {(5,[50,59],(staff-A,o,read,+,manager,yes)),
(55,[55,59](staff-B,o,read,+,staff-A,yes)),
(55,[80,150],(staff-B,o,read,+,staff-A,yes)),
(50,[80,150],(staff-A,o,read,+,staff-D,yes))}.

□

An algorithm implementing function *rvk()* is illustrated in Figure 4. The algorithm works as follows. Suppose that user x revokes access mode m on object o from user y for the interval

[5] Two intervals are considered disjoint if they cannot be collapsed into a single one (note that $[1, 2]$ and $[3, 4]$ are not disjoint).

Algorithm 5.1 *Revoke Algorithm*

INPUT: 1) A TAB.
2) A revoke request ⟨revoker,acc-mode,object,revokee,t1,t2⟩.
OUTPUT: TAB′ = *rvk*(TAB,(revoker,acc-mode,object,revokee,t1,t2)).
METHOD:
(1) **for** each authorization in TAB with $t_j = \infty$, substitute t_j with t_{max}
(2) **if** t1 = ∞ **then** substitute t1 with t_{max}
(3) T is initialized to be empty
(4) **for** each A ∈ TAB such that s(A)=revokee,o(A)=object, m(A)=acc-mode,
pn(A)='+',g(A)=revoker, [t_i(A),t_j(A)]∩[t1,t2]≠ ∅ **do**
(a) *Delete*(TAB,{(ts(A),[t_i(A),t_j(A)]∩[t1,t2],(s(A),o(A),m(A),pn(A),g(A),go(A))}
(b) **if** go(A) = "yes" **then** $T := T \cup$ {[t_i(A),t_j(A)]∩[t1,t2]}
endfor
(5) **if** $T \neq \emptyset$ **then** *casc_revoke*(revokee,object,acc-mode,T)
(6) **for** each authorization in TAB with $t_j = t_{max}$, substitute t_j with ∞

casc_revoke(*user,obj,mode,T*)

(1) T' and I are initialized to be empty
(2) **for** each $s_i \in S$ such that exists A ∈ TAB, s(A)=s_i,o(A)=*obj*,
m(A)=*mode*,g(A)=*user*, {[t_i(A),t_j(A)]}∩$T \neq \emptyset$ **do**
(a) **for** each A_k ∈ TAB such that s(A_k)=s_i,o(A_k)=*obj*,
m(A_k)=*mode*, g(A_k)=*user*,{[t_i(A_k),t_j(A_k)]}∩$T \neq \emptyset$ **do**
(1) I := {[t_i(A_k),t_j(A_k)]} ∩ T
(2) **for** each A_u ∈ TAB such that s(A_u)=g(A_k),o(A_u)=o(A_k), m(A_u)=m(A_k),
pn(A_u)='+',go(A_u)="yes",ts(A_u)<ts(A_k), {[t_i(A_u),t_j(A_u)]}∩$I \neq \emptyset$, **do**
$I := I \setminus$ {[t_i(A_u),t_j(A_u)]}
endfor
(3) *Delete*(TAB,{(ts(A_k),[t_i,t_j],(s(A_k),o(A_k),m(A_k), pn(A_k),g(A_k),go(A_k)))|[t_i,t_j]∈ I}
(4) **if** go(A_k)="yes" **then** $T' := T' \cup I$
endfor
(b) **if** $T' \neq \emptyset$ **then** *casc_revoke*(s_i,*obj,mode*,T')
endfor

Figure 4: Revoke Algorithm

[t1,t2]. In steps 1 and 2, t_{max} is substituted for each occurrence of symbol '∞' in time intervals associated with the authorizations in TAB and in the revoke request. We define t_{max} to be the first instant greater than the maximum temporal constant appearing in a temporal authorization in TAB. There is no need to consider all time instants up to ∞ as, for instants greater than t_{max}, the authorizations that are legal remain unchanged (this fact is formally proved as part of the proof of Theorem 5.1). Step 4 iteratively considers all the authorizations A in TAB for m on o granted by x to y whose time interval contains an instant in [t1,t2] (step 4) and deletes or modifies it to exclude the interval [t1,t2] (step 4a). If authorization A is with the grant option then the interval in which authorization A has been revoked is added to T (step 4b). If T is empty the algorithm terminates, since the revoke operation does not have any other effect on the authorizations in TAB. Otherwise the Revoke algorithm calls procedure '*casc_revoke()*' (step 5). Procedure '*casc_revoke()*' is a recursive procedure that determines which other authorizations have to be deleted or modified upon the revoke request. For this reason, step 2 of the procedure iteratively considers all the users s_i who had received an authorization for m on o from y for an instant $t \in T$. Step 2.a verifies which authorization A for m on o granted by y to s_i has to be revoked or modified. All the authorizations that support A in the time interval $[t_i(A),t_j(A)] \cap T$ are considered by step 2.a.2. After the execution of this step, I contains the time intervals representing the time instants in $[t_i(A),t_j(A)] \cap T$ in which A does not have a supporting authorization. Then, I is removed from the time interval of A by step 2.a.3. The process is then repeated for every user s_i whose authorizations have been modified. The final step of the main algorithm (step 6), replaces each value t_{max} appearing in the authorizations of TAB with the symbol '∞'. The following example illustrates an application of the Revoke Algorithm.

Example 5.3 Suppose that the Revoke algorithm receives as input the TAB and the revoke operation of Example 5.1. Symbol '∞' does not appear in any authorizations in TAB, nor in the revoke request; thus steps 1 and 2 are not executed. Step 4 of the algorithm considers all the positive authorizations in TAB for the read access mode on o granted by manager to staff-A whose time interval is not disjoint from the interval [60,200]. The only authorization satisfying these conditions is authorization A_1.
Thus, A_1 is replaced by the authorization A_1'=(5,[50,59],(staff-A,o,read,+,manager,yes)) by step 4.a. As authorization A_1 is with the grant option the interval [60,200] is added to T by step 4.b. Then, procedure *casc_revoke*(staff-A,o,read,{[60,200]}) is executed. Step 2 of the procedure considers all the subjects which had received an authorization from staff-A to read o for a time instant in [60,200]. During the first iteration of step 2, staff-B is considered. Authorization A_2 is considered by step 2.a and all the authorizations supporting it in the interval [60,180]=[55,180]$\cap$[60,200] are detected by step 2.a.2. The only authorization detected by step 2.a.2 is authorization A_4, which supports A_2 for the interval [80,150]. Thus, after the execution of step 2.a.2, I = {[60,79]}. Therefore A_2 is replaced by the two authorizations: (55,[55,59],(staff-B,o,read,+,staff-A,yes)) and (55,[80,150],(staff-B,o,read,+, staff-A,yes)) by step 2.a.3. Since A_2 is the only authorization in TAB granted by staff-A to staff-B, step 2.a terminates and *casc_revoke*(staff-B,o,read,{[60,79]}) is executed. The only authorization which satisfies the conditions of step 2.a is authorization A_3. No supporting authorization for A_3 is detected by step 2.a.2 for the interval [60,70] = [60,70]$\cap$[60,79], thus authorization A_3 is removed from TAB by step 2.a.3. As A_3 is without the grant option $T'=\emptyset$, thus procedure '*casc_revoke*()' is not called by step 2.b. No other iteration of step 2 is executed, as there is no other subject receiving an authorization from staff-A. Hence the algorithm terminates. The resulting TAB is:

{(5,[50,59],(staff-A,o,read,+,manager,yes)),

(55,[55,59],(staff-B,o,read,+,staff-A,yes)),
(55,[80,150],(staff-B,o,read,+,staff-A,yes)),
(50,[80,150],(staff-A,o,read,+,staff-D,yes))}.

□

The correctness of the Revoke Algorithm is stated by the following theorem.

Theorem 5.1 *(i) Algorithm 5.1 terminates. (ii) Algorithm 5.1 computes function rvk().*

The formal proof is reported in Appendix.

6 Expressiveness of the model

Our model supports the derivation of authorizations on the basis of four different temporal operators. The only operator of our model for which a correspondence exists in current authorization models is the WHENEVER operator, corresponding to a simple implication relationship between authorizations. By contrast, no correspondence exists for the other three operators. However, these operators, together with time specifications and temporal operators, make our model able to represent different protection requirements that traditional authorization models cannot support or can support only partially. In the following, we illustrate some examples of how those operators can be used to represent different protection requirements.

The WHENEVERNOT operator derives authorizations on the basis of the absence of other authorizations. As an example where this operator can turn useful, suppose that two different subjects must be authorized for an access over complementary time intervals. This requirement can be expressed by specifying the authorizations for the first subject and then derive the authorizations for the second subject for every instant in which the other subject does not have (WHENEVER) the authorization. Consider for example rule R_3 in Figure 1, which derives the authorization for `staff-A` to write `staff-document` for every instant, starting from time `30`, in which `staff-B` is not authorized.

Like the WHENEVER operator, the ASLONGAS operator derives an authorization on the basis of the existence of another authorization. It differs from the WHENEVERNOT operator in that it does not allow any derivation at and after the first instant in which the authorization on the right of the operator does not hold, regardless of whether it will start holding again. Analogously, the UNLESS operator, similar to the WHENEVERNOT operator, differs from the latter in that it derives an authorization only up to the instant in which the authorization on the right of the operator starts to hold. The ASLONGAS and UNLESS operators can be used to express constraints where the fact that an authorization starts or stops to hold has implication on authorizations to be derived.

To illustrate an example of use of ASLONGAS suppose that `temporary-staff` is to be authorized for all the authorizations on `bulletin` *currently* held by `staff`, until these authorizations hold or, at most, until time `300`. This requirement cannot be expressed by a WHENEVER rule, which would cause the derivation of authorizations for `temporary-staff` also for authorizations that have not been holding continuously since the time at which the rule was specified. By contrast, the use of the ASLONGAS operator allows to limit the derivation only up to the time where the authorizations on the right of the operator satisfies the condition to have been continuously holding. Rule R_2 in Figure 1 expresses the requirement above. According to this rule the authorization for `temporary-staff` to read `bulletin` is derived in time interval `[10,40]`. A WHENEVER rule would have derived the authorization also in the interval `[50,100]`.

To illustrate the UNLESS operator, suppose that `new-staff` will be hired and is to be authorized to write `worksheet`. Until that time, `staff` should be authorized for it. Since the time from which the `new-staff` will be authorized may not be known apriori and the authorization for `new-staff` may be even specified by a different person, no specific endpoint for the authorization for `staff` can be specified. This requirement is instead easily expressed by the UNLESS rule R_4 in Figure 1. Authorization A_5, specified later on, for `new-staff` will limit the derivation of authorizations by rule R_4 only up to time `119`. The use of WHENEVERNOT would have not been appropriate since it would have allowed to derive again the authorization for `staff` upon revocation of the authorization of `new-staff`.

Our model provides also flexibility in the administration of authorizations since it allows users to retain complete control over the objects created or delegate other users the privilege of administering accesses on the object. Delegation can be enforced by either giving a user the privilege to administer the object, or selectively by granting authorizations with the grant option. The grant option allows to delegate administration only with reference to specific privileges and to specific time intervals. The combination of administer privilege and grant-option thus provides a flexible framework for expressing delegation. For instance, user `Tom` can create object `document` and grant `manager` the administer privilege on it. Moreover, `Tom` can grant `secretary` the authorization to `read document` in time interval `[10,100]` with the grant option. As a result, `manager` will be able to grant all privileges on `document`, whereas `secretary` will be allowed only to grant the `read` privilege and only for time instants in `[10,100]`.

Note that, although delegation of administrative privileges necessarily implies some loss of control from the owners of the objects, our model still allows the owner to retain some control on the accesses that others will allow to execute on his objects, by specifying negative authorizations and derivation rules. For instance, with reference to the example just mentioned, `Tom` can forbid user `consultant` to read the document simply by specifying a negative authorization for it. As a consequence, `consultant` will not be allowed to read `document` even if an authorization is specified for it by `manager` or `secretary`. Again, as another example, `Tom` can forbid `staff-A` and `staff-B` to simultaneously hold a privilege on `document`, by specifying a WHENEVER rule that derives a negative authorization for any privilege on `document` for `staff-A` whenever `staff-B` is authorized for the privilege.

These are only some examples of protection and administrative requirements that can be expressed in our model. We believe that many other requirements arise in real-world applications which cannot be expressed in traditional authorization models and can instead be expressed in our model by properly combining authorizations, administrative capabilities, and derivation rules.

As a final remark, note that an important benefit of the WHENEVERNOT operator is to support both the open and closed policies within the same system with reference to specific accesses (for instance, depending on the object or the access mode). Like most discretionary models, our model is based on a closed world policy, i.e., only accesses explicitly authorized are allowed. The coexistence of positive and negative authorizations is regulated by the denials-take-precedence principle, i.e., negative authorizations override positive authorizations. As a result, a subject is allowed for an access if and only if he has a positive authorization for it and he has no negative authorizations for it. An alternative policy is the open policy, where only negative authorizations can be specified and subjects are allowed for an access only if they are not explicitly denied for it. The combination of negative and positive authorizations together with WHENEVERNOT rules allows us to easily enforce the open policy, as follows. A WHENEVERNOT rule can be specified stating that a positive authorization can be derived whenevernot a corresponding negative authorization holds. For instance, rule `([1,∞], ((*,public-document,*,+,Tom,no)` WHENEVERNOT `(*,public-document,*,-,*,*))` states that a positive authorization on `public-document` can be

derived for each subject and each access mode for which no negative authorization is specified. As a consequence, all accesses not explicitly denied will be authorized. The advantage of our approach is that users are not constrained to the use of a single policy. Rather they can choose the policy that best suits their needs. For instance, the closed policy can be used by default and the open policy be used on public documents to which access should be given to everybody apart of few exceptions. Closed and open policies can then be both used within the same system with specific reference to the object, subject or access mode to which the policy applies. For instance, by substituting the metacharacter "`*`" with access mode `read` in the rule above, the closed policy will be enforced only with reference to the read access mode. This flexibility is a main advantage of our model. As a matter of fact, several authors [11, 4, 14] have recognized the need for building models and mechanisms able to support multiple policies in a flexible way.

7 Concluding remarks and future work

In this report we have presented a decentralized administrative policy for a temporal authorization model previously proposed by us [3]. We have described the basic concepts of the model and illustrated the operations for granting and revoking authorizations and administrative privileges. The semantics of the revoke operation is to delete from the authorization state all the authorizations which would have not existed had the revoked authorization never been granted. We have extended this semantics, first proposed for the System R database system, to the consideration of temporal authorizations. In our model, every time a user is revoked an authorization, the authorizations the user has consequently granted may need to be revoked or their time interval be modified. As a consequence of a revocation, an authorization can also be split into several authorizations with disjoint time intervals. We have given the formal semantics of the revoke operation in our model together with an algorithm implementing it. We have also illustrated how our model, with its temporal and administrative capabilities, can be used to represent different protection requirements that arise in real-life applications.

The work presented in this report can be extended in several directions. An important issue that we are investigating concerns the consideration of different temporal operators for the derivation of authorizations and periodic authorizations/rules. A further issue concerns implementation strategies for efficiently enforcing derived authorizations. Another important direction concerns the development of authorization administration tools. Administration tools are particularly crucial when dealing with sophisticated authorization models. The area of administration tools has not, however, been so far widely investigated. We plan to invest a major effort in this direction.

References

[1] M. Abadi, M. Burrows, B.W. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] M. Baudinet, M. Niézette, and P. Wolper. On the representation of infinite temporal data and queries (extended abstract). In *Proc. ACM Symp. on Principles of Database Systems*, pages 280–290, Denver, CO, May 1991.

[3] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. A temporal access control mechanism for database systems. *IEEE Trans. on Knowledge and Data Engineering*, to appear 1996.

[4] E. Bertino, S. Jajodia, and P. Samarati. Supporting multiple access control policies in database systems. In *Proc. of the 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1996.

[5] E. Bertino, P. Samarati, and S. Jajodia. Authorizations in relational database management systems. In *Proc. First ACM Conference on Computer and Communications Security*, Fairfax, Virginia, November 1993.

[6] S. Bobrowski. Safeguarding. *DBMS*, pages 44–52, September 1993.

[7] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database security.* Addison Wesley, 1995.

[8] Oracle Corporation. *SQL Language-Reference Manual. Version 7.0.* 1992.

[9] R. Fagin. On an authorization mechanism. *ACM Trans. on Database Systems*, 3(6):310–319, Nov 1976.

[10] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Trans. on Database Systems*, 1(3):242–255, September 1976.

[11] D. Jonscher and K. Dittrich. Argos - a configurable access control system for interoperable environments. In *Proc.of the IFIP WG11.3 Working Conference on Database Security*, pages 39–66, Rensselaerville, NY, USA, 1994.

[12] J. V. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1974.

[13] J. Melton. Ansi x3h2-90-309,. Technical report, (ISO/ANSI working draft) Database Language SQL2., August 1990.

[14] E.A. Schneider. A security framework for policy-neutral object managers. In *???*, 1995.

[15] Informix Software. *Informix-OnLine/Secure Security Features User's Guide.* Inc., Menlo Park, CA, 1993.

[16] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Conference Proceedings*, pages 191–202, Dallas, TX, Winter 1988. USENIX.

[17] R.K. Thomas and R.S. Sandhu. Discretionary access control in object-oriented databases: Issues and research directions. In *Proc. 16th National Computer Security Conference*, pages 63–74, Baltimore, MD, Sept. 1993.

[18] T.Y.C. Woo and S.S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2 & 3):107–136, 1993.

A Correctness of the Revoke Algorithm

Proof of Theorem 5.1
(i) Termination. To prove the termination of the Revoke algorithm is sufficient to prove the termination of procedure '*casc_revoke*()' (step 5), since step 4 of the algorithm is a finite iteration bounded by the number of authorizations in TAB. Consider procedure '*casc_revoke*()'. Suppose

that procedure '*casc_revoke*()' never terminates. This means that it indefinitively calls itself. Procedure '*casc_revoke*()' continues to call itself if at each iteration the set T' is not empty. T' is a set of time intervals representing the set of time instants of the authorizations of subject $\mathtt{s}_i$ for $\mathtt{m}$ on $\mathtt{o}$, received as arguments by function '*Delete*() till the current point in the computation. It is easy to verify that procedure '*casc_revoke*()' does not add any authorization to TAB nor increases the time interval of the existing authorizations. Moreover, by the semantics of function '*Delete*()', the next time $\mathtt{s}_i$ is examined by procedure '*casc_revoke*()' the time intervals of the remaining authorizations of $\mathtt{s}_i$ for $\mathtt{m}$ on $\mathtt{o}$ do not contain any instant in the set T' computed in the previous call. Then, since the time intervals of the authorizations for $\mathtt{s}_i$ are subsets of the finite interval $[\mathtt{1},\mathtt{t}_{max}]$, procedure'*casc_revoke*()' terminates.

(ii). Let TAB$'$ be the TAB resulting from the execution of the Revoke Algorithm. By definition 5.4 proving the thesis is equivalent to prove: TAB$'$=*Delete*(TAB,(*REV*(TAB)$\cup$RREV(TAB))). Since our algorithm considers only instants less than constant $\mathtt{t}_{max}$, it is first necessary to prove the following lemma.

Lemma A.1 *Given a TAB, let TAB$_{\mathtt{t}_{max}}$ be the TAB obtained by substituting each occurrence of symbol '∞' appearing in a temporal authorization in TAB with $\mathtt{t}_{max}$. Steps 1, ..., 5 of algorithm 5.1 compute Delete(TAB$_{\mathtt{t}_{max}}$,(REV(TAB$_{\mathtt{t}_{max}}$)$\cup$RREV(TAB$_{\mathtt{t}_{max}}$))).*

Proof Note that, the Revoke algorithm neither adds any authorization to TAB$_{\mathtt{t}_{max}}$ nor increases the time interval of the existing authorizations; updates to TAB$_{\mathtt{t}_{max}}$ are only performed by calling function '*Delete*()'. Therefore, to prove the thesis is sufficient to prove that the overall set of authorizations received as argument by function '*Delete*()' is equal to $\{REV(\text{TAB}_{\mathtt{t}_{max}}) \cup RREV(\text{TAB}_{\mathtt{t}_{max}})\}$. For brevity in the following we use *REV* (*RREV*) for $REV(\text{TAB}_{\mathtt{t}_{max}})$ ($RREV(\text{TAB}_{\mathtt{t}_{max}})$) The only steps in which function '*Delete*()' is called are step 4a of the Revoke algorithm and step 2.a.3 of procedure '*casc_revoke*()'. Let DEL_1 be the overall set of authorizations received as argument by function '*Delete*()'during the execution of step 4a of the algorithm. It is trivial to prove that DEL_1 is equal to *REV*. Thus, we have only to prove that the set of authorizations received as argument by function '*Delete*()' during the execution of procedure '*casc_revoke*()' is equal to *RREV*. Let DEL_2 be the overall set of authorizations received as argument by function '*Delete*()' during the execution of procedure '*casc_revoke*()'. We have to prove $\mathtt{A} \in DEL_2 \Leftrightarrow \mathtt{A} \in RREV$. Let us first prove $\mathtt{A} \in DEL_2 \Rightarrow \mathtt{A} \in RREV$. Suppose that $\mathtt{A} \in DEL_2$ and $\mathtt{A} \notin RREV$. Let $\mathtt{A}$ be one of the authorizations in DEL_2 not belonging to *RREV* with the minimum timestamp. There could be more than one authorization in DEL_2 with the same timestamp, but the choice of the one considered is not relevant for the proof. Let $\mathtt{A} = (\mathtt{ts},[\mathtt{t}_i,\mathtt{t}_j],\mathtt{auth})$. By hypothesis, $\mathtt{A} \notin RREV$, then it means that one of the following conditions hold: *(i)* $\nexists\ \mathtt{A}' = (\mathtt{ts},[\mathtt{t}'_i,\mathtt{t}'_j],\mathtt{auth}) \in$ TAB such that $[\mathtt{t}_i(\mathtt{A}),\mathtt{t}_j(\mathtt{A})] \subseteq [\mathtt{t}'_i,\mathtt{t}'_j]$, or *(ii)* $\exists t \in [\mathtt{t}_i(\mathtt{A}),\mathtt{t}_j(\mathtt{A})]$ and $\exists \langle \mathtt{A}_1,\ldots,\mathtt{A}_n,\mathtt{A}\rangle^t$, such that $\mathtt{A}_1,\ldots,\mathtt{A}_n,\mathtt{A} \in Delete(\text{TAB}_{\mathtt{t}_{max}},REV)$. Suppose that *(i)* holds. It means that there does not exist in TAB an authorization $\mathtt{A}'$ having the same timestamp, subject,object,access mode,sign,grantor and grant option as $\mathtt{A}$ and such that $[\mathtt{t}_i(\mathtt{A}),\mathtt{t}_j(\mathtt{A})] \subseteq [\mathtt{t}_i(\mathtt{A}'),\mathtt{t}_j(\mathtt{A}')]$. Therefore, from how the set I is computed by steps 2.a.1 and 2.a.2 of procedure '*casc_revoke*()', $\mathtt{A} \notin DEL_2$, which contradicts the assumption. Let us now suppose condition *(ii)* holds, that is, $\exists t \in [\mathtt{t}_i(\mathtt{A}),\mathtt{t}_j(\mathtt{A})]$ and $\exists \langle \mathtt{A}_1,\ldots,\mathtt{A}_n,\mathtt{A}\rangle^t$, such that $\mathtt{A}_1,\ldots,\mathtt{A}_n,\mathtt{A} \in Delete(\text{TAB}_{\mathtt{t}_{max}},REV)$, where $\mathtt{A}_n$ is a generic authorization supporting $\mathtt{A}$ at time t. Since $Delete(\text{TAB}_{\mathtt{t}_{max}},REV)$ is the TAB resulting from the execution of step 4 of the algorithm, when procedure '*casc_revoke*()' is called by step 5, there exists in TAB$_{\mathtt{t}_{max}}$ a supporting authorization for $\mathtt{A}$ at time t. Since there exists a supporting chain $\langle \mathtt{A}_1,\ldots,\mathtt{A}_n,\ \mathtt{A}\rangle^t$ for $\mathtt{A}$, at time t such that $\mathtt{A}_1,\ldots,\mathtt{A}_n,\mathtt{A} \in Delete(\text{TAB}_{\mathtt{t}_{max}},REV)$, then there exists a supporting chain $\langle \mathtt{A}_1,\ldots,\mathtt{A}_n\rangle^t$

for $\mathtt{A}_n$, at time t such that $\mathtt{A}_1, \ldots, \mathtt{A}_n \in Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$. Therefore there does not exists in $RREV$ an authorization with the same timestamp, subject,object,access mode,sign,grantor and grant option as $\mathtt{A}_n$ and such that its time interval contains instant t. Such an authorization does not belong also to DEL_2, since A is the element of DEL_2 not belonging to $RREV$ with the minimum timestamp, and by the definition of supporting chain $\mathtt{ts(A}_n\mathtt{)<ts(A)}$. DEL_2 represents the overall set of authorizations received as argument by function '*Delete*()' during the execution of procedure '*casc_revoke*()'. Thus, by the semantics of function '*Delete*()', there exists in TAB′ an authorization with the same timestamp, subject,object,access mode,sign,grantor and grant option as $\mathtt{A}_n$ containing t in its time interval, that is, there exists in TAB′ an authorization supporting A at time t. We have already proved that, since $\mathtt{A} \in DEL_2$, there exists in TAB an authorization A′ having the same timestamp, subject,object,access mode,sign,grantor and grant option as A and such that $[\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}] \subseteq [\mathtt{t}_i\mathtt{(A')},\mathtt{t}_j\mathtt{(A')}]$. Authorizations A and A′ have the same supporting authorizations in the interval $[\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}]$. Thus, when A′ is considered by step 2.a, $t \notin I$ at the end of step 2.a.2, as there exits in TAB′ an authorization supporting A at time t. Therefore, there does not exists in DEL_2 an authorization with the same timestamp, subject, object, access mode sign, grantor and grant option as A′ and such that t belongs to its time interval Then, $\mathtt{A} \notin DEL_2$, as by hypothesis A is an authorization which differs from A′ only for its time interval and such that $t \in [\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}]$, which contradicts the assumption.

We now prove that $\mathtt{A} \in RREV \Rightarrow \mathtt{A} \in DEL_2$. Suppose $\mathtt{A} \in RREV$ and $\mathtt{A} \notin DEL_2$. Let A be one of the authorizations in $RREV$ not belonging to DEL_2 with the minimum timestamp. As in the previous case, there could be more than one authorization in $RREV$ with the minimum timestamp, but the choice of the one to be considered is not relevant. If $\mathtt{A} \in RREV$, it means that, $\forall t \in [\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}]$ $\nexists\langle \mathtt{A}_1, \ldots, \mathtt{A}_n, \mathtt{A}\rangle^t$, such that $\mathtt{A}_1, \ldots, \mathtt{A}_n, \mathtt{A} \in Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$, that is, there does not exists in $Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$ a supporting chain for A for each instant of its time interval. Let $\overline{t} \in [\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}]$. By definition of supporting chain, it means that either there does not exists in $Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$ an authorization supporting A at time $\overline{t}$ or, for each authorization $\mathtt{A}' \in Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$, supporting A at time $\overline{t}$, $\nexists\langle \mathtt{A}_1, \ldots, \mathtt{A}_m, \mathtt{A}'\rangle^{\overline{t}}$, with $\mathtt{A}_1, \ldots, \mathtt{A}_m \in Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$. If there does not exists in $Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$ an authorization supporting A at time $\overline{t}$, then, when procedure '*casc_revoke*()' is executed, there does not exist in TAB an authorization supporting A at time $\overline{t}$. If for each authorization $\mathtt{A}' \in Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$ supporting A at time $\overline{t}$, $\nexists\langle \mathtt{A}_1, \ldots, \mathtt{A}_m, \mathtt{A}'\rangle^t$, with $\mathtt{A}_1, \ldots, \mathtt{A}_m \in Delete(\mathrm{TAB}_{\mathtt{t}_{max}}, REV)$, then for each authorization in TAB supporting A at $\overline{t}$, there exists in $RREV$ an authorization with the same timestamp, subject, object, access mode, sign, grantor and grant option and with $\overline{t}$ belonging to its time interval. The authorizations supporting A that are in $RREV$ are also in DEL_2 as, by hypothesis, A is the authorization in $RREV$ not belonging to DEL_2 with the minimum timestamp and each authorization supporting A has the timestamp lesser than the timestamp of A. Thus, the authorizations supporting A which are in DEL_2 are received as argument by function '*Delete*()' in step 2.a.3 of procedure '*casc_revoke*()'. Therefore, there does not exist in TAB′ an authorization supporting A at time $\overline{t}$. The same considerations hold $\forall t \in [\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}]$. Therefore each authorization supporting A in TAB does not belong to TAB′. Since $\mathtt{A} \in RREV$, it means that existed in TAB an authorization A* with the same timestamp,subject,object,access mode, sign,grantor and grant option of A, such that $[\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}] \subseteq [\mathtt{t}_i\mathtt{(A^*)},\mathtt{t}_j\mathtt{(A^*)}]$. As, by hypothesis, TAB contains only legal authorizations, there exists in TAB a supporting authorization for A* for each instant of its time interval. $\forall t \in [\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}]$, A and A* have the same supporting authorizations. Since A does not have any supporting authorization in TAB′, when A* is considered by step 2.a of procedure '*casc_revoke*()', $[\mathtt{t}_i\mathtt{(A)},\mathtt{t}_j\mathtt{(A)}] \subseteq T$, as T represents the set time instants of the authorizations of $\mathtt{s(A^*)}$ for $\mathtt{m(A^*)}$ on $\mathtt{o(A^*)}$ deleted till the current point of the computation. Then,

$[t_i(A),t_j(A)] \subseteq I$ at the end step 2.a.3, as A does not have any supporting authorization. Thus, A is added to DEL_2, which contradicts the assumption. □

We are now ready to prove the theorem. We are now ready to prove the theorem. We have to prove that $A \in TAB' \Leftrightarrow$
$A \in Delete(TAB,(REV(TAB) \cup RREV(TAB)))$.
Let us first prove that $A \in TAB' \Rightarrow A \in Delete(TAB,(REV \cup RREV))$. Let $A = (ts,[t_i,t_j],auth)$. Suppose that $t_{max} \notin [t_i,t_j]$. Then, if $A \in TAB'$, by lemma A.1 $A \in Delete(TAB_{t_{max}},(REV \cup RREV))$. By the semantics of function '*Delete*()' this implies that $\nexists\ A'=(ts,[t'_i,t'_j],auth) \in (REV \cup RREV)$ such that $[t'_i,t'_j] \cap [t_i,t_j] \neq \emptyset$. It is easy to prove that this implies that $A' \notin (REV(TAB) \cup RREV(TAB))$, as $TAB_{t_{max}}$ is obtained by replacing each symbol '∞' occurring in the time interval of authorizations in TAB with t_{max}. Indeed if $A' \notin REV$ it means that one of the following conditions hold: *(i)* auth is not a positive authorization whose revocation is explicitly required, and then A' does not belong also to $REV(TAB)$, or *(ii)* $\forall (ts,[t_u,t_v],auth) \in TAB_{t_{max}}$ $[t_u,t_v] \cap [t1,t2] = \emptyset$, where [t1,t2] is the time interval for which the revocation of auth has been requested. If *(ii)* is true, then the same condition holds in TAB. Therefore, $A' \notin REV(TAB)$. If $A' \notin RREV$, it means that one of the following conditions hold: *(i)* $\nexists\ (ts,[t_u,t_v],auth) \in TAB_{t_{max}}$ such that $[t'_i,t'_j] \subseteq [t_u,t_v]$, or *(ii)* $\exists t \in [t'_i,t'_j]$ and $\exists \langle A_1,\ldots,A_n,A' \rangle^t$, with $A_1,\ldots,A_n,A' \in Delete(TAB_{t_{max}},REV)$. If *(i)* is true, then such an authorization does not exist also in TAB. Then $A' \notin RREV(TAB)$. If *(i)* is false, than *(ii)* holds. This implies that authorizations $A_1,\ldots,A_n,A' \in Delete(TAB,REV(TAB))$, as , by the previous step, $REV = REV(TAB)$. Thus $A' \notin RREV(TAB)$. Then $A \in Delete(TAB,REV(TAB) \cup RREV(TAB))$. Now suppose that $t_{max} \in [t_i,t_j]$. By step 6 of the algorithm this means that $t_j = \infty$. By lemma A.1, this implies that there exists $A' \in Delete(TAB_{t_{max}},(REV \cup RREV))$ such that $A' = (ts,[t_i,t_{max}],auth)$. By the semantics of function '*Delete*()' this implies that $\nexists\ (ts,[t'_i,t'_j],auth) \in (REV \cup RREV)$ such that $[t'_i,t'_j] \cap [t_i,t_{max}] \neq \emptyset$. This means that this authorization neither belongs to REV nor to $RREV$. Thus, $\nexists$ in REV an authorization $(ts,[t'_i,t'_j],auth)$ such that $[t'_i,t'_j] \cap [t_i,t_{max}] \neq \emptyset$. It is easy to prove, analogously to the previous case, that this authorization does not exist also in $REV(TAB)$. By the definition of t_{max}, this implies that $\nexists$ in $REV(TAB)$ an authorization $(ts,[t'_i,t'_j],auth)$ such that $[t'_i,t'_j] \cap [t_i,\infty] \neq \emptyset$. Similarly, it can be proved that since $\nexists$ in $RREV$ an authorization $(ts,[t'_i,t'_j],auth)$ such that $[t'_i,t'_j] \cap [t_i,t_{max}] \neq \emptyset$, then this authorization does not exist also in $RREV(TAB)$. By the definition of t_{max}, this implies that there does not exist in $RREV(TAB)$ an authorization $(ts,[t'_i,t'_j],auth)$ such that $[t'_i,t'_j] \cap [t_i,\infty] \neq \emptyset$. Therefore $A \in Delete(TAB,REV(TAB) \cup RREV(TAB))$.
We have finally to prove $A \in Delete(TAB,(REV(TAB) \cup RREV(TAB))) \Rightarrow A \in TAB'$. Suppose that $t_{max} \notin [t_i(A),t_j(A)]$. If $A \in Delete(TAB,(REV(TAB) \cup RREV(TAB)))$, it is easy to prove, using considerations analogous to the ones used in the previous step, that $A \in Delete(TAB_{t_{max}},(REV \cup RREV))$. Then, by lemma A.1 $A \in TAB'$, since $t_j(A) < t_{max}$. Suppose now that $t_{max} \in [t_i(A),t_j(A)]$. By definition of t_{max}, this implies $t_j(A) = \infty$. Let $A = (ts,[t_i,\infty],auth)$.
$A \in Delete(TAB,(REV(TAB) \cup RREV(TAB)))$ implies that $\nexists\ (ts,[t'_i,t'_j],auth) \in (REV(TAB) \cup RREV(TAB))$ such that $[t'_i,t'_j] \cap [t_i,\infty] \neq \emptyset$. This implies that $\nexists\ (ts,[t'_i,t'_j],auth) \in (REV(TAB) \cup RREV(TAB))$ such that $[t'_i,t'_j] \cap [t_i,t_{max}] \neq \emptyset$. It is trivial to prove that this implies that such an authorization does not exist also in $(REV \cup RREV)$. Therefore $(ts,[t_i,t_{max}],auth) \in Delete(TAB_{t_{max}},(REV \cup RREV))$. Then, by lemma A.1 $(ts,[t_i,t_{max}],auth)$ belongs to the TAB resulting from the execution of steps 1,...,5 of the algorithm. Therefore step 6 substitutes the interval $[t_i,t_{max}]$ with $[t_i,\infty]$. Then $A = (ts,[t_i,\infty],auth) \in TAB'$. □