A Temporal Access Control Mechanism for Database Systems

Elisa Bertino, *Member, IEEE*, Claudio Bettini, Elena Ferrari, and Pierangela Samarati

Abstract—This paper presents a discretionary access control model in which authorizations contain temporal intervals of validity. An authorization is automatically revoked when the associated temporal interval expires. The proposed model provides rules for the automatic derivation of new authorizations from those explicitly specified. Both positive and negative authorizations are supported. A formal definition of those concepts is presented in the paper, together with the semantic interpretation of authorizations and derivation rules as clauses of a general logic program. Issues deriving from the presence of negative authorizations are discussed. We also allow negation in rules: it is possible to derive new authorizations on the basis of the absence of other authorizations. The presence of this type of rules may lead to the generation of different sets of authorizations, depending on the evaluation order. An approach is presented, based on establishing an ordering among authorizations and derivation rules, which guarantees a unique set of valid authorizations. Moreover, we give an algorithm detecting whether such an ordering can be established for a given set of authorizations and rules. Administrative operations for adding, removing, or modifying authorizations and derivation rules are presented and efficiency issues related to these operations are also tackled in the paper. A materialization approach is proposed, allowing to efficiently perform access control.

Index Terms—Database security, temporal authorization, database management, temporal reasoning, general logic programs, access control.

1 Introduction

In many real-world situations, permissions have a temporal dimension, in that they are usually limited in time or may hold only for specific periods of time. In general, however, access control mechanisms provided as part of commercial data management systems do not have temporal capabilities. In a typical commercial Relational DBMS (RDBMS), for example, 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 week. If such a need arises, authorization management and access control must be implemented at application program level. This approach makes authorization management very difficult, if at all possible. Thus the need of adding temporal capabilities to access control model appears very strong, as pointed out also by Thomas and Sandhu in [11].

In this paper, we present an authorization model that extends conventional authorization models, like those provided by commercial RDBMSs, with temporal capabilities. Our temporal authorization model is based on two main concepts. The first concept is the temporal interval for authorizations. Each authorization has a time interval associated with it, representing the set of time instants for which the authorization is granted. An authorization expires after the associated time interval has elapsed. The second concept is the temporal dependency among authorizations. A temporal dependency can be seen as a rule allowing an authorization to be derived from the presence (or absence) of another authorization. A temporal dependency can be used, for example, to specify that a user has an authorization as long as another user has the same or a different authorization. Four different temporal dependency operators are provided in our model. Temporal dependencies are expressed in form of derivation rules. Such rules may be parametric, in that a single rule may denote a set of dependencies. For example, a single derivation rule may specify that a user can read *all* the files that another user can read, relatively to an interval of time.

Besides these temporal capabilities, the model supports both positive and negative authorizations. The capability of supporting explicit denials, provided by negative authorizations, can be used for specifying exceptions and for supporting a stricter control in the case of decentralized authorization administration [5]. The combination of positive/negative authorizations with temporal authorizations results in a powerful yet flexible authorization model.

A critical issue in our model is represented by the presence of derivation rules that allow to derive new authorizations on the basis of the absence of other authorizations. From one point of view these rules provide more expressiveness for the representation of temporal dependencies. From another point of view they introduce the problem of generating a unique set of authorizations. Indeed, a given set of authorizations and derivation rules may generate different sets of authorizations, depending on the evaluation order. To avoid this problem we impose a syntactical restriction on the set of derivation rules and we show how this condition guarantees the uniqueness of the set of derived authorizations. In the

• *The authors are with the Dipartimento di Scienze dell'Informazione, Università di Milano, via Comelico 39, 20135 Milano, Italy.*
E-mail: {ebertino, bettini, ferrarie, samarati}@dsi.unimi.it.

Manuscript received Feb. 1, 1995; revised Oct. 4, 1995.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number K96007.

paper, we show also how this problem is related to the problem of negation in logic programming.

Another issue discussed in the paper is the efficiency of the access control. Whenever an access must be enforced, the system must check whether the appropriate authorization is present in the authorization catalogs or whether it can be inferred from the authorizations in the catalogs through the derivation rules. The activity of inferring an authorization can be rather expensive, like performing a query on a deductive database. Thus, a materialization approach has been adopted. This approach is very similar to the view materialization approach used in deductive and relational databases [6], [8]. Under such an approach, the results of a view are calculated and stored when the view is defined, rather than being recomputed each time the view is queried. We use a similar approach: each time a new authorization is added, all authorizations that can be inferred from it are calculated and stored into the authorization catalogs. Thus, access control is very efficient, since there is no difference in costs between explicit authorizations and derived authorizations. Note that administrative operations become more expensive, but they are much less frequent than access control. Moreover, we use proper maintenance algorithms to update the materialized authorizations without need of recomputing them all upon execution of administrative requests.

Time issues in access control and derivation rules for authorizations have come to the attention of the researchers only recently. The Kerberos system [10], based on the client-server architecture, provides the notion of *ticket*, needed for requiring a service to the server, with an associated validity time. The validity time is used to save the client from the need to acquire a ticket for each interaction with the server. The ticket mechanism is not used to grant accesses to the resources managed by the system. Rather, it is only used to denote that a client has been authenticated by the authentication server. Thus, the scope of the temporal ticket mechanism is very different from our access control model.

Woo and Lam in [13] have proposed a very general formalism for expressing authorization rules. Their language to specify rules has almost the same expressive power of first order logic. A major issue in their formalism is the tradeoff between expressiveness and efficiency which seems to be strongly unbalanced in their approach. We think that it is important to devise more restricted languages focusing only on relevant properties. The temporal authorization model we propose in this paper is a step in this direction.

A logic language for stating security specifications, 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.

A preliminary version of the authorization model presented in this paper was presented by Bertino, Bettini, and Samarati in [4]. The model presented in this paper has a number of major differences with respect to the previous model. The current model supports both positive and negative authorizations, and it provides substantial extensions to derivation rules. In particular, in the current model, derivation rules also have temporal interval of validity. This extension coupled with negative and positive authorizations leads to several interesting questions concerning both theory and implementation, that we investigate in the current paper. We investigate also efficiency issues, by proposing a materialization strategy for computing the set of valid authorizations and by giving algorithms for the maintenance of such materialization.

In this paper, we only deal with discretionary access control and not with mandatory access control. Note, however, that the majority of DBMS only provide discretionary access control. Therefore, since the focus of our research is how to extend the authorization facilities provided by a conventional DBMS, we only address discretionary access control. Recent multilevel DBMS (like Trusted Oracle [9]) provide mandatory access control coupled with discretionary access control. The new features provided by our model could be orthogonally incorporated into such systems as well.

The remainder of this paper is organized as follows: Section 2 describes the authorization model giving the basic definitions and examples. In Section 3, we present the formal semantics for authorizations and derivation rules and explain the problems due to the presence of negations in rules. A sufficient condition to guarantee the presence of a unique set of derived authorizations, and an algorithm for checking this condition are given. In Section 4, we show how all the valid authorizations can be computed. Administrative operations that allow the users to add, remove, or modify temporal authorizations and rules are described in Section 5. Efficiency issues concerning the need of updating the set of valid authorizations upon administrative operations are considered in Section 6. For lack of space we refer the reader interested in proofs to [3].

2 The Authorization Model

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

In the following, U denotes the set of users, O the set of objects, and M the set of access modes executable on the objects.

Our model allows the specification of explicit *authorizations*, stating the permission or denial for users to exercise access modes on objects, and of *derivation rules* stating the permission or denial for users to exercise access modes on objects conditioned on the presence or the absence of other permissions or denials. Each authorization and derivation rule has a time interval associated with it indicating the time at which the authorization/rule is applicable.

We assume time to be discrete. In particular, we take as our model of time the natural numbers IN with the total order relation $<$.

We are now ready to introduce temporal authorizations and derivation rules.

2.1 Temporal Authorizations

In our model, both positive and negative authorizations can be specified. Positive authorizations indicate permissions whereas negative authorizations indicate denials for access.

Authorizations are formally defined as follows.

DEFINITION 2.1 (Authorization): *An authorization is a 5-tuple* (s,o,m,pn,g) *where:*

- s $\in$ *U is 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.*

Tuple (s,o,m,+,g) states that user s has been granted access mode m on object o by user g. Tuple (s,o,m,-,g) states that user s has been forbidden access mode m on object o by user g.

We consider a temporal constraint to be associated with each authorization. We refer to an authorization together with a temporal constraint as a *temporal authorization*. Temporal authorizations are defined as follows.

DEFINITION 2.2 (Temporal authorization): *A temporal authorization is a pair* (time,auth), *where* time *is a time interval* $[t_b, t_e]$, *with* $t_b \in$ IN, $t_e \in$ IN $\cup \infty$, $t_b \leq t_e$, *and* auth = (s, o, m, pn, g) *is an authorization.*

Temporal authorization ($[t_1, t_2]$,(s, o, m, pn, g)) states that user g has granted user s an authorization (positive if pn = '+' or negative if pn = '-') for access mode m on object o that holds between times t_1 and t_2. For example, authorization ([10, 50], (John, o_1, read, +, Bob)) states that John can read object o_1 between time instants 10 and 50 and that this authorization was granted by Bob.

Note that an authorization without any temporal constraint can be represented as a temporal authorization whose validity spans from the time at which the authorization is granted to infinity.

In the following, given a temporal authorization A = ($[t_b, t_e]$, (s,o,m,pn,g)) we denote with s(A), o(A), m(A), pn(A), g(A), t_b(A), and t_e(A), respectively the subject, the object, the privilege, the sign of the authorization (positive or negative), the grantor in A, and the starting and ending time of A.

2.2 Derivation Rules

Additional authorizations can be derived from the authorizations explicitly specified. The derivation 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 both positive as well as negative authorizations. Like authorizations, derivation rules have a time interval associated with them. The time interval associated with a derivation rule indicates the set of instants in which the rule is applied.

Derivation rules can also contain variables in their specification. We refer to derivation rules where all the terms in the authorizations are explicitly specified as *ground derivation rules* and to derivation rules containing variables as *parametric derivation rules*.

2.2.1 Ground Derivation Rules

Ground derivation rules are defined as follows.

DEFINITION 2.3 (Ground derivation rule): *A ground derivation rule is defined as* ($[t_b, t_e]$, $A_1\langle$OP$\rangle A_2$), *where* $[t_b, t_e]$ *is the time interval associated with the rule,* $t_b \in$ IN, $t_e \in$ IN $\cup \infty$, $t_b \leq t_e$, A_1, *and* A_2 *are authorizations,* g(A_1) *is the user who specified the rule, and* ⟨OP⟩ *is one of the following operators:* WHENEVER, ASLONGAS, WHENEVERNOT, UNLESS.

Rule ($[t_b, t_e]$, (s_1,o_1,m_1,pn_1,g_1) ⟨OP⟩ (s_2,o_2,m_2,pn_2,g_2)) states that user s_1 is authorized (if pn_1 = '+') or denied (if pn_1 = '−') for access mode m_1 on object o_1 according to the presence or absence (depending on the operator) of the authorization (s_2, o_2, m_2, pn_2, g_2).

The formal semantics of the temporal operators used in the derivation rules will be given in Section 3. Their intuitive semantics is as follows:

- ($[t_b, t_e]$, A_1 WHENEVER A_2).
  We can derive A_1 for each instant in $[t_b, t_e]$ for which A_2 is given or derived. For example, rule R_1 in Fig. 1, specified by Sam, states that every time, in [7,35], Ann can read object o_1 thanks to an authorization granted by Sam, also Chris can read object o_1.
- ($[t_b, t_e]$,A_1 ASLONGAS A_2).
  We can derive A_1 for each instant t in $[t_b, t_e]$ such that A_2 is either given or derived for each instant from t_b to t. Note that, unlike the WHENEVER operator, the ASLONGAS operator does not allow to derive A_1 at an instant t in $[t_b, t_e]$ if there exists an instant t', with $t_b \leq t' \leq t$, such that A_2 is not given and cannot be derived at t'.
- ($[t_b, t_e]$,A_1 WHENEVERNOT A_2).
  We can derive A_1 for each instant in $[t_b, t_e]$ for which A_2 is neither given nor derived.
- ($[t_b, t_e]$,A_1 UNLESS A_2).
  We can derive A_1 for each instant t in $[t_b, t_e]$ such that A_2 is neither given nor can be derived for each instant from t_b to t. Note that, unlike the WHENEVERNOT, the UNLESS operator does not allow to derive A_1 at an instant t in $[t_b, t_e]$ if there exists an instant t', with $t_b \leq t' \leq t$, such that A_2 is given or derived at t'.

(A1) ([10, 20], (Ann, o_1, read, +, Sam))
(A2) ([30, 40], (Ann, o_1, read, +, Sam))
(R1) ([7, 35], (Chris, o_1, read, +, Sam) WHENEVER (Ann, o_1, read, +, Sam))
(R2) ([10, 35], (Matt, o_1, read, +, Sam) ASLONGAS (Ann, o_1, read, +, Sam))
(R3) ([5, ∞](John, o_1, read, +, Sam) WHENEVERNOT (Ann, o_1, read, +, Sam))
(R4) ([5, 15], (Bob, o_1, read, +, Sam) UNLESS (Ann, o_1, read, +, Sam))
(R5) ([1, 80], (Jim, o_1, read, +, Sam) WHENEVER (Bob, o_1, read, +, Sam))

Fig. 1. An example of authorizations and derivation rules.

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

- ([10,20], (Chris,o_1,read,+,Sam)) and ([30, 35], (Chris, o_1, read, +, Sam)) from authorizations A_1 and A_2 and rule R_1.
- ([10,20], (Matt,o_1,read,+,Sam)) from authorization A_1 and rule R_2.
- ([5,9], (John,o_1,read,+,Sam)), ([21,29], (John,o_1,read,+,Sam)), and ([41,∞], (John,o_1,read,+,Sam)) from rule R_3.
- ([5,9], (Bob,o_1,read,+,Sam)) from rule R_4.
- ([5,9], (Jim,o_1,read,+,Sam)) from rules R_4 and R_5.

2.2.2 Authorizations and Derivation Rules Specification

Before proceeding to illustrate the semantics of derivation rules and authorizations we need to make a remark on authorizations and rules. In our model, only users explicitly authorized can specify authorizations and derivation rules. Administrative privileges give users the authority of granting accesses on objects to users either directly (explicit authorizations) or indirectly (through derivation rules). Three different administrative privileges are considered: refer, administer, and own. The semantics of these privileges is as follows.

- refer: If a user has the refer privilege on an object, he can refer to the object in a derivation rule, i.e., the object can appear at the right of the temporal operator in a derivation rule specified by the user.
- administer: If a user has the administer privilege on an object, he can grant to and revoke from other users authorizations to access the object (either explicitly or through rules).
- own: It indicates possession of an object. When a user creates an object he receives the own privilege on it. The own privilege allows the user to grant and revoke access authorizations as well as to grant and revoke administrative privileges (but own) on his object.

Administrative authorizations, i.e., authorizations for administrative privileges are not constrained to a specific time interval but hold from the time at which they are specified until the time they are revoked by the object's owner. However, for sake of simplicity and uniformity with respect to other authorizations, we associate time intervals also to administrative authorizations. The time interval associated with an administrative authorization spans from the time at which the authorization is specified to ∞. Administrative authorizations are formally defined as follows.

DEFINITION 2.4 (Administrative authorization): *An administrative authorization is defined as* ([t_b,∞], (s,o,p)) *where* [t_b,∞] *is the time interval associated with the authorization,* s ∈ *U is the user to whom the authorization is granted,* o ∈ *O is the object on which the authorization is specified, and* p *is the administrative privilege granted to* s.

For instance, administrative authorization ([20,∞], (John, o_1,administer)) states that John has the administer privilege on object o_1, and therefore can grant other users access authorizations on o_1, starting from time 20.

For each authorization ([t_b,t_e], A_1), we require g(A_1) to have the own or administer privilege on o(A_1). Moreover, for each derivation rule R = ([t_b,t_e],A_1 ⟨op⟩ A_2) both the following conditions must be satisfied:

- g(A_1) has either the own or administer privilege on o(A_1),
- g(A_1) has either the own, administer, or refer privilege on o(A_2).

These conditions are checked at the time an authorization/rule is specified and the insertion of the authorization/rule is accepted only if the conditions are satisfied.[1]

2.2.3 Parametric Derivation Rules

Derivation rules can also use variables in their specification. We refer to these rules as parametric derivation rules. To introduce parametric derivation rules, we first give the definition of authorization pattern.

DEFINITION 2.5 (Authorization pattern): *An authorization pattern* AP *is a tuple* (s,o,m,pn,g) *where* s, g ∈ *U* ∪ {*}, o ∈ *O* ∪ {*}, m ∈ *M* ∪ {*}, *and* pn ∈ {+,−}.

Symbol '*' is a special character denoting any user, object, or access mode, depending on its position in the authorization pattern.

DEFINITION 2.6 (Matching authorization): *An authorization* A *matches a pattern* AP *if each element of* A *is equal to the corresponding element of* AP, *if different from '*'.*

Parametric derivation rules are defined as follows.

DEFINITION 2.7 (Parametric derivation rule): *A parametric derivation rule is defined as* ([t_b,t_e],AP_1 ⟨op⟩ AP_2), *where* AP_1 *and* AP_2 *are authorization patterns, and all the other elements are as in Definition* 2.3. *Authorization patterns in the rule must verify the following conditions:*

- g(AP_1) *and at least one element among* s(AP_1), o(AP_1), m(AP_1) *are different from '*'*
- *if symbol '*' is used for* s(AP_1), o(AP_1), *or* m(AP_1) *it is also used for the corresponding element* s(AP_2), o(AP_2), m(AP_2) *in* AP_2.

A parametric derivation rule can be seen as a shorthand for specifying several ground derivation rules operating on different subjects, objects, or access modes. Given a parametric derivation rule, we refer to the ground rules to which it corresponds as instances of the parametric rule. This is expressed by the following definition.

DEFINITION 2.8 (Parametric rule instances): *Let* R = ([t_b,t_e], AP_1 ⟨op⟩ AP_2) *be a parametric derivation rule. The set of instances of* R *is the set composed of all possible ground derivation rules* ([t_b,t_e],A_m ⟨op⟩ A_n) *such that* A_m *matches* AP_1, A_n *matches* AP_2, *and such that the following conditions are satisfied:*

- *if* s(AP_1) = '*' *then* s(A_m) = s(A_n)
- *if* o(AP_1) = '*' *then* o(A_m) = o(A_n)
- *if* m(AP_1) = '*' *then* m(A_m) = m(A_n)

Note that instances derived from parametric rules must also satisfy the constraints on administrative privileges illustrated in the previous section for rules.

1. We will elaborate on this in Section 5.

The following example illustrates the use of parametric derivation rules.

EXAMPLE 2.2. Sam wishes to grant the authorization to exercise a certain number of access modes on certain objects to a group of friends, Chris, Matt, and Jim. Instead of specifying one authorization for every access mode and every object for each of his friends, Sam can proceed as follows. A new user sam-friends, playing the role of the group is defined. For each user that Sam wishes to include in the group, a WHENEVER rule parametric over the object and the access mode is defined where the authorization at the left of the operator has as subject the user identifier and the authorization at the right has as subject sam-friends (see Fig. 2). The time interval associated with the rule can be interpreted as the time interval at which the user appearing on the left is considered as a member of group sam-friends. For example, rules R_1, R_2, and R_3 in Fig. 2 allow given a positive authorization specified for sam-friends, to derive the same authorization for for Chris, Matt, and Jim, respectively. Rule R_2 expires at time 100 (intuitively, after that time Matt will not be considered anymore a member of the group); hence, the time interval associated with the authorizations derived for Matt will have ending time equal to 100.

(A1) ([10, ∞], (sam-friends, o_1, read, +, Sam))
(A2) ([20, 200], (sam-friends, o_1, write, +, Sam))
(A3) ([20, ∞], (sam-friends, o_2, read, +, Sam))
(A4) ([10, ∞], (sam-friends, o_2, write, +, Sam))
(A5) ([50, ∞], (Jim, o_2, write, -, John))
(R1) ([1, ∞], (Chris, *, *, +, Sam) WHENEVER
(sam-friends, *, *, +, Sam))
(R2) ([1, 100], (Matt, *, *, +, Sam) WHENEVER
(sam-friends, *, *, +, Sam))
(R3) ([1, ∞], (Jim, *, *, +, Sam) WHENEVER
(sam-friends, *, *, +, Sam))

Fig. 2. An example of parametric derivation rules.

In the example above, Sam appears as grantor of the authorization on the right of the operators in rules R_1-R_3. Hence, authorizations for Chris, Matt, and Jim will be derived only from authorizations granted to sam-friends by Sam. Sam can require the rules to fire regardless of the grantor of the authorizations to sam-friends by putting '*' as grantor in the right side of rules R_1-R_3.

3 Formal Semantics

In this section, we formalize the semantics of temporal authorizations and derivation rules. First of all, it is necessary to point out that the possibility to express negative authorizations introduces potential conflicts among authorizations. Suppose that a negative authorization for a privilege on an object is granted to a user who has previously obtained the same privilege on that object. We then have, for a given time interval, the presence of both negative and positive authorizations. This is not to be intended as an inconsistency, since we consider negative authorizations as prevailing with respect to positive authorizations.

Considering the set of authorizations and rules in Fig. 2, from rule R_3 and authorization A_4 we can derive ([10, ∞], (Jim, o_2, write, +, Sam)). By authorization A_5 we have ([50, ∞], (Jim, o_2, write, -, John)). This is not an inconsistency, since we apply the denials-take-precedence principle. Hence, the negative authorization prevails, and Jim will have the authorization to write object o_2 only in the interval [10, 49]. The formal semantics obeys to the denials-take-precedence principle. We start the description of the formal semantics by introducing the concept of a TAB.

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

In the rest of the paper, we denote with INST-TAB a TAB where each parametric rule has been substituted by its set of instances according to Definition 2.8. Obviously, TAB and INST-TAB are equivalent.

The semantics of a TAB is given as a set of clauses in a *general logic program* corresponding to INST-TAB. We use a logic with two sorts, the natural numbers (IN) as a temporal sort and a generic domain ($\mathcal{D}$) as the other sort. The language includes constant symbols 1, 2, ... for natural numbers, a finite set of constant symbols (e.g., s_1, o_1, m_1, g_1, -, +, s_2, ...) for elements in $\mathcal{D}$, and temporal variable symbols t, t', t''. Predicate symbols include the temporal predicate symbols ≤ and < with the fixed interpretation of the corresponding order relation on natural numbers, the predicate symbol $F()$ with temporal arity 1 and domain arity 5, the predicate symbols $F_N()$ and $F_G()$ with temporal arity 2 and domain arity 5, and the predicate symbol $G()$ with temporal arity 1 and domain arity 3. The resulting language is very similar to the temporal deductive language proposed in [2] with the main difference being the presence of negation in our rules.

For each type of authorization/rule in INST-TAB, Table 1 reports its corresponding clause/set of clauses. Intuitively, the predicate $F()$ is used to represent the authorizations at specific instants. The fact that $F(t, A)$ 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, $G(t, s, o, m)$ 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 that interpretation. $F_N(t'', t, A)$ 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. $F_P(t'', t, A)$ 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.

We denote the logic program corresponding to a TAB with P_{TAB}. We consider *stable model semantics* of logic programs with negation [7] to identify the models[2] of P_{TAB}.

DEFINITION 3.2 (Valid Authorization): *Given a model M of P_{TAB}, an authorization A is said to be valid at time t with respect to M if $F(t, A)$ is contained in M. If P_{TAB} has a unique model M and M contains $F(t, A)$, we simply say that A is valid at time t.*

2. Due to the properties of the resulting program, in this case stable models are identical to *well-founded* models [12].

TABLE 1
SEMANTICS OF TEMPORAL AUTHORIZATIONS AND RULES

$[t_b, t_e], (s, o, m, -, g)$:
$F(t, s, o, m, -, g) \leftarrow t_b \le t \le t_e$

$[t_b, t_e], (s, o, m, +, g)$:
$F(t, s, o, m, +, g) \leftarrow t_b \le t \le t_e, \neg G(t, s, o, m)$

$[t_b, t_e], (s_1, o_1, m_1, -, g_1)$ WHENEVER (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, -, g_1) \leftarrow t_b \le t \le t_e, F(t, s_2, o_2, m_2, pn, g_2)$

$[t_b, t_e], (s_1, o_1, m_1, +, g_1)$ WHENEVER (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, +, g_1) \leftarrow t_b \le t \le t_e, F(t, s_2, o_2, m_2, pn, g_2), \neg G(t, s_1, o_1, m_1)$

$[t_b, t_e], (s_1, o_1, m_1, -, g_1)$ ASLONGAS (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, -, g_1) \leftarrow t_b \le t \le t_e, F(t, s_2, o_2, m_2, pn, g_2), \neg F_N(t_b, t, s_2, o_2, m_2, pn, g_2)$

$[t_b, t_e], (s_1, o_1, m_1, +, g_1)$ ASLONGAS (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, +, g_1) \leftarrow t_b \le t \le t_e, F(t, s_2, o_2, m_2, pn, g_2), \neg F_N(t_b, t, s_2, o_2, m_2, pn, g_2), \neg G(t, s_1, o_1, m_1)$

$[t_b, t_e], (s_1, o_1, m_1, -, g_1)$ WHENEVERNOT (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, -, g_1) \leftarrow t_b \le t \le t_e, \neg F(t, s_2, o_2, m_2, pn, g_2)$

$[t_b, t_e], (s_1, o_1, m_1, +, g_1)$ WHENEVERNOT (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, +, g_1) \leftarrow t_b \le t \le t_e, \neg F(t, s_2, o_2, m_2, pn, g_2), \neg G(t, s_1, o_1, m_1)$

$[t_b, t_e], (s_1, o_1, m_1, -, g_1)$ UNLESS (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, -, g_1) \leftarrow t_b \le t \le t_e, \neg F(t, s_2, o_2, m_2, pn, g_2), \neg F_P(t_b, t, s_2, o_2, m_2, pn, g_2)$

$[t_b, t_e], (s_1, o_1, m_1, +, g_1)$ UNLESS (s_2, o_2, m_2, pn, g_2) :
$F(t, s_1, o_1, m_1, +, g_1) \leftarrow t_b \le t \le t_e, \neg F(t, s_2, o_2, m_2, pn, g_2), \neg F_P(t_b, t, s_2, o_2, m_2, pn, g_2), \neg G(t, s_1, o_1, m_1)$

$F_P(t'', t, s, o, m, pn, g) \leftarrow t'' \le t' < t, F(t', s, o, m, pn, g)$

$F_N(t'', t, s, o, m, pn, g) \leftarrow t'' \le t' < t, \neg F(t', s, o, m, pn, g)$

$G(t, s, o, m) \leftarrow F(t, s, o, m, -, g)$

3.1 Restrictions on Rules

An important property that we require for our set of temporal authorizations and rules is that we must always be able to derive a unique set of valid authorizations. This means, for example, that each set of rules together with a fixed set of explicit authorizations should not derive different authorizations depending on the evaluation order. We give an example illustrating how different authorizations can be derived depending on the evaluation order.

EXAMPLE 3.1. Consider the following rules:

(R_1) ($[t_b, t_e]$, A_1 WHENEVERNOT A_2)
(R_2) ($[t_b, t_e]$, A_2 WHENEVERNOT A_1)

Suppose that there are no explicit authorizations for A_1 or A_2 in the TAB and these are the only rules. If we consider first R_1 we derive authorization A_1 and we cannot derive A_2. If we consider first R_2, we derive A_2 and not A_1. Hence, we have two different sets of derived authorizations. In this case there is no reason to give preference to one set or the other.

From the point of view of the semantics that we have given, 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. Hence, we limit derivation rules so that a unique model can be computed. In the rest of this section we formally define sets of rules that should be avoided in order to guarantee a unique model for P_{TAB}, and we give an algorithm for their detection.

In the following, we use the term *negative operator* (NEGOP) to refer to WHENEVERNOT or UNLESS, and *negative rule* to refer to a rule using a negative operator. Similarly, *positive operator* (POSOP) is used to refer to WHENEVER or ASLONGAS, *present operator* (PRESENTOP) is used to refer to WHENEVER or WHENEVERNOT, and *past operator* (PASTOP) is used to refer to UNLESS or ASLONGAS. Moreover, we use symbols A_i as a shortcut for the 5-tuple $(s_i, o_i, m_i, pn_i, g_i)$, while A_i^+ forces $pn_i = +$ and A_i^- forces $pn_i = -$.

A binary relation $\hookrightarrow$ among the temporal authorizations appearing in INST-TAB is defined as follows:

- If there is a rule ($[t_b, t_e]$, $A_m \langle OP \rangle A_n$) in INST-TAB, where $\langle OP \rangle$ is an arbitrary operator, then $A_n[t] \hookrightarrow A_m[t]$ for each t with $t_b \le t \le t_e$. The $\hookrightarrow$ relation represents a dependency of A_m at instant t from A_n at the same instant. When $\langle OP \rangle$ is a negative operator we say that $\hookrightarrow$ represents a strict dependency.
- If there is a rule ($[t_b, t_e]$, $A_m \langle PASTOP \rangle A_n$) in INST-TAB, then $A_n[t] \hookrightarrow A_m[t']$ for each t, t' with $t_b \le t < t' \le t_e$, where $\hookrightarrow$ represents a strict dependency.

Using this relation we can define the more complex notion of priority among temporal authorizations.

DEFINITION 3.3 (Priority): *An authorization A_n at time t has higher priority than an authorization A_m at time t' (written $A_n[t] > A_m[t']$) if one of the following conditions holds:*

- *a sequence $A_n[t] = A_1[t] \hookrightarrow, \ldots, \hookrightarrow A_{k-1}[t''] \hookrightarrow A_k[t'] = A_m[t']$ exists such that at least one of the $\hookrightarrow$ relationships is a strict dependency,*
- *two sequences $A_n[t] = A_1[t] \hookrightarrow \ldots \hookrightarrow A_l^-[t'']$ and $A_{l+1}^+[t''] \hookrightarrow \ldots \hookrightarrow A_k[t'] = A_m[t']$ exist such that $s(A_l^-) = s(A_{l+1}^+)$, $o(A_l^-) = o(A_{l+1}^+)$, and $m(A_l^-) = m(A_{l+1}^+)$,*
- *an authorization A_1 and an instant t'' exist such that $A_n[t] > A_1[t'']$ and $A_1[t''] > A_m[t']$.*

Note that the second condition in the above definition implies that each negative authorization has higher priority than its positive counterpart at the same instant.

We are now ready to identify critical sets of derivation rules.

DEFINITION 3.4 (Critical set): *A TAB contains a critical set of rules if and only if an authorization A_m in INST-TAB and an instant t exist such that A_m at instant t has priority over itself ($A_m[t] > A_m[t]$).*

EXAMPLE 3.2. Consider the set of rules:

(R_1) ([5,∞], (Ann,o_1,write,-,Jim) WHENEVERNOT (Bob,o_1,write,+,*))
(R_2) ([10,∞], (John,*,write,-,Jim) WHENEVER (Ann,*,write,-,*))
(R_3) ([40,∞], (Bob,o_1,*,+,Jim) UNLESS (John,o_1,*,+,*))

These three rules form a critical set. It is easily checked that Definition 3.4 applies to this set of rules. Indeed, by the first condition in Definition 3.3 and rules R_1 and R_2 we have that (Bob,o_1,write,+,Jim)[41] > (John,o_1,write,-,Jim)[41]. But we have (John,o_1,write,-,Jim)[41] > (John,o_1,write,+,Jim)[41] by the second condition, and, again by the first condition and rule R_3, we obtain (John,o_1,write,+,Jim)[41] > (Bob,o_1,write,+,Jim)[41]. Applying twice the third condition (transitivity) we have

(Bob, o_1, write, +, Jim)[41] > (Bob, o_1, write, +, Jim)[41]. Hence, this set of rules is critical.

The CSD (Critical Set Detection) algorithm, described in the next subsection, can be used to recognize and reject a TAB containing a critical set.

3.2 An Algorithm for Critical Set Detection

We use a set of disjoint[3] 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 ($T_1 \cap T_2$), and difference ($T_1 \setminus T_2$) have the usual semantics of set operations. However, we implement these 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.

Given a INST-TAB, the algorithm for critical set detection returns FALSE if a critical set exists in TAB; otherwise it returns a sequence of sets (levels) $\langle L_1, \ldots, L_k \rangle$ representing a partition of the set of pairs $\langle A, t \rangle$ for each authorization A appearing (either explicitly or in a rule) in INST-TAB and for each instant t between 1 and t_{max}. We define t_{max} to be the first instant greater than the maximum temporal constant appearing in INST-TAB. In the following, we refer to each set L_i as level i. If pair $\langle A, T \rangle$ is in level i, we say that A is in level i for each $t \in T$. Intuitively, authorizations appearing at lower levels for a certain set of instants have higher priority for evaluation than authorizations appearing at higher levels (for the same or different sets of instants). In this and other algorithms in the paper, we use the functions 'Add()' and 'Delete()' to add/delete or modify the pairs $\langle A, T \rangle$. The result of the statement 'Add $\langle A_m, T' \rangle$ to L'' is the addition of that pair to L if there is no pair $\langle A_m, T \rangle$ in L for any T, otherwise it is the replacement of $\langle A_m, T \rangle$ with $\langle A_m, T \cup T' \rangle$. Analogously, the result of 'Delete $\langle A_m, T' \rangle$ from L'' is the deletion of that pair from L if the pair $\langle A_m, T \rangle$ is in L with $T' = T$, otherwise it is the replacement of $\langle A_m, T \rangle$ with $\langle A_m, T \setminus T' \rangle$.

The algorithm is reported in Figs. 3, 4, and 5, and it works as follows. In step 1, t_{max} is substituted for each occurrence of symbol '∞' in time intervals associated with authorizations and rules in INST-TAB. There is no need to consider all time instants up to ∞. For instants greater than t_{max} the authorizations that are valid remain unchanged. If a critical set exists, it will be found at a time lower than or equal to t_{max}. In step 2, max-level is determined as the number of authorizations appearing in INST-TAB multiplied by t_{max}. max-level corresponds to the number of pairs $\langle A, t \rangle$ to be partitioned. Then, the number of levels (top-level) is initialized to 1. Level 1 initially contains all authorizations in INST-TAB for each instant between 1 and t_{max}. Step 3 recursively calls function 'check-levels()' which examines the authorizations at different levels and the dependencies among authorizations. It possibly changes level to pairs $\langle A, T \rangle$ on the basis of the dependency. The loop at step 3 ends when the last call of 'check-levels()' does not change any level or the level number is greater than max-level. In the first case, the levels constructed by the algorithm are returned. In the second case, FALSE is returned.

3. 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 3.1 *Critical Set Detection (CSD) Algorithm*

INPUT: INST-TAB.

OUTPUT: FALSE if a critical set is detected;
otherwise a sequence of sets $\langle L_1, \ldots, L_k \rangle$ representing a partition of the set of pairs $\langle A, t \rangle$ such that A appears in INST-TAB and $1 \leq t \leq t_{max}$.
Each set L_i is called level i and $L_i = \{\langle A_1, T_{1,i} \rangle, \ldots, \langle A_r, T_{r,i} \rangle\}$ where $T_{j,i}$ is a set of time intervals associated with A_j at level i.

METHOD:

1. For each temporal authorization or rule having the time limit $t_e = \infty$ substitute it with t_{max}.
2. max-level := n-auth*t_{max}, where n-auth is the number of authorizations appearing in INST-TAB
   top-level := 1
   $L_1 := \emptyset$
   For each authorization A appearing in INST-TAB Do
   $L_1 := L_1 \cup \{\langle A, \{[1, t_{max}]\} \rangle\}$
   endfor
3. Repeat check-levels ($\langle L_1, \ldots, L_{\text{top-level}} \rangle$)
   Until there are no changes to any level or top-level > max-level.
4. Return FALSE if top-level > max-level, the sequence $\langle L_1, \ldots, L_{\text{top-level}} \rangle$ otherwise.

Fig. 3. An algorithm for critical set detection.

Function check-levels($\langle L_1, \ldots, L_{\text{top-level}} \rangle$)

1. For l :=top-level,...,1 Do
   partition the set of pairs $\langle A_i, T_{i,l} \rangle \in L_l$ such that pn(A) = '–' in $\{S_1, \ldots, S_u\}$,
   where $S_i = \{\langle A_1, T_{1,l} \rangle, \ldots, \langle A_v, T_{v,l} \rangle\}$ such that s(A_1)=...=s(A_v)=s_i,
   o(A_1)=...=o(A_v)=o_i, and m(A_1)=...=m(A_v)=m_i.
   For each S_i Do
   $T_i := \bigcup \{T_{n,l} \mid \langle A_n, T_{n,l} \rangle \in S_i\}$
   For $h := l, \ldots, 1$ Do
   For each $\langle A_m, T_{m,h} \rangle \in L_h$ with $T_{m,h} \cap T_i \neq \{[\,]\}$ and
   such that s(A_m)=s_i, o(A_m)=o_i, m(A_m)=m_i, pn(A_m) = '+' Do
   $T' := T_{m,h} \cap T_i$
   Delete $\langle A_m, T' \rangle$ from L_h
   If l = top-level then top-level := $l + 1$
   Add $\langle A_m, T' \rangle$ to L_{l+1}
   endfor
   endfor
   endfor
2. For each rule $R = ([t_b, t_e], A_m$ (PRESENTOP) $A_n\}$ Do
   $T_R := \{[t_b, t_e]\}$
   l := top-level
   While $T_R \neq \{[\,]\}$ and $l \geq 1$ Do
   If $\langle A_n, T_{n,l} \rangle \in L_l$ with $T_{n,l} \cap T_R \neq \{[\,]\}$ then
   Case (PRESENTOP) of
   WHENEVER: If $l > 1$ then update(l, A_m, $T_{n,l} \cap T_R$)
   WHENEVERNOT: update($l + 1$, A_m, $T_{n,l} \cap T_R$)
   endcase
   $T_R := T_R \setminus T_{n,l}$
   endif
   $l := l - 1$
   endwhile
   endfor
3. For each rule $R = ([t_b, t_e], A_m$ (PASTOP) $A_n)$ Do
   l := top-level
   $t_f := t_e$
   While $t_f \geq t_b$ and $l \geq 1$ Do
   If $\langle A_n, T_{n,l} \rangle \in L_l$ with $\{[t_b, t_f]\} \cap T_{n,l} \neq \{[\,]\}$ then
   $t_l := \min\{t : t \in \{[t_b, t_f]\} \cap T_{n,l}\}$
   case (PASTOP) of
   ASLONGAS: If $t_l < t_f$ then update ($l + 1$, A_m, $[t_l + 1, t_f]$)
   update (l, A_m, $[t_l, t_l]$)
   UNLESS: update ($l + 1$, A_m, $[t_l, t_f]$)
   endcase
   $t_f := t_l - 1$
   endif
   $l := l - 1$
   endwhile
   endfor

Fig. 4. **Function** check-levels.

```
Function update(lev, A_m, T)
If lev > top-level then
  top-level := lev
  L_lev := ∅
endif
h := lev − 1
While T ≠ {[]} and h ≥ 1 Do
    If ⟨A_m, T_m,h⟩ ∈ L_h with T_m,h ∩ T ≠ {[]} then
      T' := T_m,h ∩ T
      Delete ⟨A_m, T'⟩ from L_h
      Add ⟨A_m, T'⟩ to L_lev
      T := T \ T_m,h
    endif
    h := h − 1
endwhile
```

Fig. 5. **Function** update.

Function 'check-levels()' is composed of three steps. In step 1, all levels from top-level to 1 are examined. If a negative authorization A_n is found at a given level l for a certain set of time intervals $T_{n,l}$, the level of all positive authorizations A_m having same subject, object, and access mode as A_n and appearing at a level lower than l is increased to $l + 1$ for all time instants in $T_{n,l}$. In step 2, all the rules $R = ([t_b, t_e], A_m \langle \text{PRESENTOP} \rangle A_n)$ are evaluated. Levels are examined in decreasing order starting from top-level. Every time authorization A_n is found at level l for a time interval $T_{n,l}$ not disjoint from $[t_b, t_e]$, function 'update()' is called to increase the level of A_m for the time instants appearing in both $T_{n,l}$ and $[t_b, t_e]$. The new level is l, if the operator in the rule is WHENEVER, and $l + 1$, if it is WHENEVERNOT. In step 2, all the rules $R = ([t_b, t_e], A_m \langle \text{PASTOP} \rangle A_n)$ are evaluated. Again, levels are examined in decreasing order starting from top-level. Every time authorization A_n is found at level l for a time interval $T_{n,l}$ not disjoint from $[t_b, t_e]$, function 'update()' is called to increase the level of A_m for the time instants $[t_b, t_e]$ greater than or equal to the minimum instant t_l in both $[t_b, t_e]$ and $T_{n,l}$. The new level is l, for instant t_l, and $l + 1$, for instants in $[t_b, t_e]$ greater than it.

Function 'update(),' given a level *lev*, an authorization A_m, and a set of time intervals T, brings authorization A_m at level *lev* for each time instant for which A_m appears at levels lower than *lev*.

EXAMPLE 3.3. Consider a TAB containing the following authorizations and rules:

([10, 200], A_1)
([5, 100], A_2 WHENEVER A_1)
([40, 60], $\bar{A}_{2|John}$ WHENEVERNOT A_3)
([10, 80], A_4 WHENEVERNOT A_5)

where $\bar{A}_{2|John}$ indicates a negative authorization with same subject, object, and access mode as A_2 but with John as grantor. The algorithm for critical set detection returns the following levels:

$L_1 = \{\langle A_1, \{[1, 201]\}\rangle, \langle \bar{A}_{2|John}, \{[1, 39], [61, 201]\}\rangle, \langle A_3, \{[1, 201]\}\rangle, \langle A_4, \{[1, 9], [81, 201]\}\rangle, \langle A_5, \{[1, 201]\}\rangle\}$

$L_2 = \{\langle \bar{A}_{2|John}, \{[40, 60]\}\rangle, \langle A_2, \{[1, 39], [61, 201]\}\rangle, \langle A_4, \{[10, 80]\}\rangle\}$

$L_3 = \{\langle A_2, \{[40, 60]\}\rangle\}$

3.3 Correctness of the CSD Algorithm and Model Uniqueness

The following two theorems state some properties of the levels returned by the CSD algorithm with respect to the dependencies among authorizations.

THEOREM 3.1. *Let A_n and A_m be two authorizations appearing in* INST-TAB *and t, t' be two time instants lower than or equal to t_{max} such that $A_n[t] \hookrightarrow A_m[t']$. Then, either the algorithm returns FALSE or, at the end of the execution, authorization A_m for instant t' appears at a level higher than or equal to that of authorization A_n for instant t. If $\hookrightarrow$ is a strict dependency then A_m for instant t' appears at a level higher than that of A_n for instant t.*

THEOREM 3.2. *Let A_n and A_m be two authorizations appearing in* INST-TAB *with same subject, access mode, and object but with different sign. Then, either the algorithm returns FALSE or, at the end of the execution, the positive authorization appears at a level higher than that of the negative authorization for each time instant between 1 and t_{max}.*

The correctness of the CSD algorithm is stated by the following theorem.

THEOREM 3.3. *Given a TAB, 1) the CSD algorithm terminates and 2) it returns a FALSE value if and only if the TAB contains a critical set.*

As we have observed, for the purpose of determining the authorization state of the system at a certain instant, the uniqueness of the P_{TAB} model at that instant is required. The uniqueness of the model in absence of critical sets is guaranteed by the following theorem.

THEOREM 3.4. *Given a TAB with no critical sets, the corresponding logic program P_{TAB} has a unique model.*

4 Materialization of Authorizations

In our model, the control of whether a request to access an object for a given access mode can be authorized may require the evaluation of several rules. Two different strategies can be used to enforce access control:

Run-time derivation: Every time a user requires an access, the system verifies whether the access request can be authorized on the basis of the authorizations and the derivation rules in TAB and by computing, if necessary, the derived authorizations.

Materialization: The system permanently maintains all the valid authorizations, both explicit and derived. Upon an access request, the system can immediately check whether a valid corresponding positive authorization exists.

Both these approaches have some pros and cons. The first approach has the advantage that no actions are required upon modification of the TAB; however access control becomes cumbersome since each access request may require the computation of derived authorizations. In the second approach, this run-time computation is avoided at the price of explicitly maintaining the derived authorizations that will have to be updated every time the TAB is modified.

Since, generally, access requests are considerably more frequent than administrative requests modifying authorizations and/or rules, we argue that the second approach is preferable. Moreover, the drawback provided by the need of recalculating the explicit authorizations upon modifications to the TAB can be overcome by the application of efficient algorithms that update the materialized authorizations upon modifications without need of reconsidering all rules and recomputing all the materialized authorizations.

For the reasons above, we adopt the materialization approach. In the following we illustrate how to compute, given a TAB, the corresponding valid authorizations. In Section 6, we will provide algorithms for reflecting changes to the TAB in the materialized authorizations without the need of recomputing all authorizations from the beginning.

DEFINITION 4.1 (Temporal Authorization Base Extent): *The Temporal Authorization Base Extent* (TAB_{EXT}) *of TAB is the set of valid authorizations derived from TAB.*

TAB_{EXT} contains all the valid authorizations of TAB computed according to the semantics of explicit authorizations and derivation rules.

Authorizations are maintained in TAB_{EXT} using a compact representation: each A_k is associated with a set T_k of disjoint intervals, representing the instants at which A_k is valid.

At time $t = 0$, TAB_{EXT} does not contain any explicit or derived authorizations. Upon the execution of each administrative operation (such as grant/revoke of authorizations or rules) TAB_{EXT} is updated to reflect the effects of the operation execution.

If the strategy of maintaining both explicit and derived authorizations is not adopted from the beginning, it is necessary to populate TAB_{EXT} from the explicit authorizations and derivation rules already present in TAB. If there is no critical set, the CSD algorithm returns a sequence of levels $\langle L_1, \ldots, L_k \rangle$ such that, for each authorization, the corresponding set of instants $1, \ldots, t_{max}$ is partitioned among the k levels. This sequence is essential to establish an evaluation order that guarantees that the computed TAB_{EXT} contains all and only valid authorizations.

Algorithm 4.1, reported in Fig. 6, computes the TAB_{EXT} of a TAB. The algorithm receives as input the TAB's instantiated version INST-TAB and the sequence $\langle L_1, \ldots, L_k \rangle$ given by the CSD algorithm. The algorithm is based on the technique used to compute the model of (locally) stratified logic programs. Intuitively, rule instances and authorizations are partitioned among a finite number of levels according to a priority relation and inferences at a certain level are performed only when all possible inferences at lower levels have been performed.

The main step of the algorithm (step 2) is an iteration on the k levels returned by the CSD algorithm. For each level i, starting from $i = 1$, the algorithm:

1) Constructs the set $\overline{X}_i$ of authorizations and rules available at level i. More precisely, $\overline{X}_i$ contains pairs $\langle x, T' \rangle$ where x is an element of INST-TAB. x can be an explicit authorization $([t_b, t_e], A_m)$ or a rule $([t_b, t_e], A_m \langle \text{OP} \rangle A_n)$. T' is the set of intervals representing all instants $t \in [t_b, t_e]$ such that A_m is in level i for instant t.
2) Derives new authorizations drawing all possible inferences at level i by using the elements in $\overline{X}_i$ and the authorizations previously derived.

Algorithm 4.1

INPUT: The output $\langle L_1, \ldots, L_k \rangle$ of the CSD Algorithm and INST-TAB.

OUTPUT: 1) $TAB_{EXT} = \{\langle A_i, T_i \rangle \mid A_i$ is a valid authorization for each interval in $T_i\}$
2) A sequence $\langle \overline{X}_1, \ldots, \overline{X}_k \rangle$, where $\overline{X}_i = \{\langle z, T' \rangle \mid z \in$ INST-TAB with $z = ([t_b, t_e], A_m)$ or $z = ([t_b, t_e], A_m \langle \text{OP} \rangle A_n)$, and $T' = [t_b, t_e] \cap T_{m,i} \neq \emptyset\}$.

METHOD:

1) TAB_{EXT} and each $\overline{X}_i$ are initialized to be empty
2) For i:=1 to k Do
   a) #Construction of $\overline{X}_i$#
      For each element z in INST-TAB, where $z = ([t_b, t_e], A_m)$ or $z = ([t_b, t_e], A_m \langle \text{OP} \rangle A_n)$ Do
         If $\langle A_m, T_{m,i} \rangle \in L_i$ with $T_{m,i} \cap [t_b, t_e] \neq \emptyset$ then
            $\overline{X}_i := \overline{X}_i \cup \langle z, T_{m,i} \cap [t_b, t_e] \rangle$
      endfor
   b) #Construction of TAB_{EXT}#
      Repeat
         For each element $\langle z, T' \rangle \in \overline{X}_i$, where $z = ([t_b, t_e], A_m)$ or $z = ([t_b, t_e], A_m \langle \text{OP} \rangle A_n)$ Do
            T := Derive-auth($\langle z, T' \rangle$, TAB_{EXT})
            If $T \neq \emptyset$ then Add $\langle A_m, T \rangle$ to TAB_{EXT}
         endfor
      Until no new authorization can be derived
   endfor
3) For each $\langle A, T \rangle$ in TAB_{EXT} with $t_{max} \in T$, substitute t_{max} with ∞.

Function Derive-auth($\langle z, T' \rangle$, TAB_{EXT})
   $T := T'$
   If $z = ([t_b, t_e], A_m \langle \text{OP} \rangle A_n)$ then
      Case OP of
         WHENEVER: $T := T' \cap T_n$
         WHENEVERNOT: $T := T' \setminus T_n$
         ASLONGAS: If $t_b \in I$ for some $I \in T_n$ then $T := T' \cap \{I\}$
                   else $T := \emptyset$
         UNLESS: If $T_n \cap [t_b, t_e] \neq \emptyset$ then
                   $t_k := \min(\{t \mid t \in T_n$ and $t_b \leq t \leq t_e\})$
                   If $t_k > t_b$ then $T := [t_b, t_k - 1] \cap T'$
                   else $T := \emptyset$
                   endif
      endcase
   If $pn(A_m) =$ '+' then
      $T := T \setminus \bigcup \{T_k \mid pn(A_k) =$ '−', $s(A_k) = s(A_m)$, $o(A_k) = o(A_m)$, $u(A_k) = u(A_m)\}$
   return T

Fig. 6. An algorithm for TAB_{EXT} generation.

The last step of the algorithm (step 3) extends the intervals of derived authorizations on the basis of the following observation:

> If we have derived an authorization for the instant t_{max}, we are guaranteed that the authorization can be derived for any instant greater than t_{max}.

This fact is due to the particular form of our rules and it is formally proved as part of the proof of Theorem 4.1.

The following example illustrates an application of the algorithm for TAB_{EXT} generation.

EXAMPLE 4.1. Consider the TAB illustrated in Example 3.3. The levels computed by the CSD algorithm are illustrated in the same example. We now apply the algorithm for TAB_{EXT} generation. Let $TAB_{EXT}^{(i)}$ be the TAB_{EXT} resulting from the ith iteration.

- For $i = 1$ we obtain:
  2.a) $\overline{X}_1 = \{\langle ([10, 200], A_1), \{[10, 200]\} \rangle\}$
  2.b) $TAB_{EXT}^{(1)} = \{\langle A_1, \{[10,200]\} \rangle\}$ since $\langle ([10,200], A_1), \{[10,200]\} \rangle$ is the only element of $\overline{X}_1$ and there are no authorizations in $TAB_{EXT}^{(1)}$ blocking A_1.

- For $i = 2$ we obtain:
  2.a)
  $\overline{X}_2 = \{\langle([40, 60], \mathtt{A}_{2|John}^{-} \text{ WHENEVERNOT } \mathtt{A}_3), \{[40, 60]\}\rangle,$
  $\langle([5, 100], \mathtt{A}_2 \text{ WHENEVER } \mathtt{A}_1), \{[5, 39], [61, 100]\}\rangle,$
  $\langle([10, 80], \mathtt{A}_4 \text{ WHENEVERNOT } \mathtt{A}_5), \{[10, 80]\}\rangle\}.$
  2.b) From $\langle([40, 60], \mathtt{A}_{2|John}^{-} \text{ WHENEVERNOT } \mathtt{A}_3), \{[40, 60]\}\rangle$ and authorizations in $TAB_{EXT}^{(1)}$ we obtain: $\langle \mathtt{A}_{2|John}^{-}, \{[40, 60]\}\rangle$. From $\langle$([5, 100], $\mathtt{A}_2$ WHENEVER $\mathtt{A}_1$), {[5, 39], [61, 100]}$\rangle$ we obtain: $\langle A_2$, {[10, 39], [61, 100]}$\rangle$. From $\langle$([10, 80], $\mathtt{A}_4$ WHENEVERNOT $\mathtt{A}_5$),{[10, 80]}$\rangle$ we obtain: $\langle \mathtt{A}_4$, {[10, 80]} $\rangle$. Hence,
  $TAB_{EXT}^{(2)} = \{\langle \mathtt{A}_1, \{[10, 200]\}\rangle, \langle \mathtt{A}_2, \{[10, 39], [61, 100]\}\rangle$
  $\langle \mathtt{A}_{2|John}^{-}, \{[40, 60]\}\rangle, \langle \mathtt{A}_4, \{[10, 80]\}\rangle\}.$
- For $i = 3$ we obtain:
  2.a) $\overline{X}_3 = \{\langle([5, 100], \mathtt{A}_2 \text{ WHENEVER } \mathtt{A}_1), \{[40, 60]\}\rangle\}$.
  2.b) Function
  Derive - auth($\langle([5, 100], \mathtt{A}_2$ WHENEVER $\mathtt{A}_1), \{[40, 60]\}\rangle$, $TAB_{EXT}^{(2)}$)
  returns $T = \emptyset$, since authorization $([40, 60], \mathtt{A}_{2|John}^{-})$ blocks $\mathtt{A}_2$ for the time interval [40, 60].
  $TAB_{EXT}^{(3)} = \{\langle \mathtt{A}_1, \{[10, 200]\}\rangle, \langle \mathtt{A}_2, \{[10, 39], [61, 100]\}\rangle,$
  $\langle \mathtt{A}_{2|John}^{-}, \{[40, 60]\}\rangle, \langle \mathtt{A}_4, \{[10, 80]\}\rangle\}$
- There are no more levels, $t_{max} = 201$ in this example and it does not appear in $TAB_{EXT}^{(3)}$. Hence, the algorithm terminates returning $TAB_{EXT}^{(3)}$.

The correctness of the algorithm is stated by the following theorem:

THEOREM 4.1. *Given* TAB_{EXT} *as returned by Algorithm 4.1, an authorization* A *is valid at time* $\bar{t}$ *if and only if there exists* $\langle$A, $T\rangle$ *in* TAB_{EXT} *with* $\bar{t} \in T$.

Once we have an updated TAB_{EXT}, each access request can be checked against TAB_{EXT}. An access request from user s_1 to exercise access mode m_1 on object o_1 at time t will be allowed only if a pair $\langle$A, $T\rangle$ exists in TAB_{EXT} such that s(A) = s_1, o(A) = o_1, m(A) = m_1, pn(A) = '+', and $t \in T$.

5 TAB Administration

Administrative operations allow the users to add, remove, or modify temporal authorizations and derivation rules and to give or revoke other users the right to administer their objects or to refer to them in derivation rules. 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 considered in our model. The syntax of the operations in BNF form is given in Table 2. With reference to the figure, nonterminal symbols ⟨subject⟩, ⟨object⟩, ⟨access-mode⟩, ⟨auth-t⟩, and ⟨nat-number⟩ represent elements of the domains U, O, M , {+, -}, and IN, respectively. Nonterminal 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.

TABLE 2
SYNTAX OF ADMINISTRATIVE OPERATIONS

⟨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⟩ 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⟩ | *
⟨start-time⟩	::=	# | ⟨nat-number⟩
⟨end-time⟩	::=	∞ | ⟨nat-number⟩ | +⟨nat-number⟩

Administrative requests can affect access authorizations, derivation rules, or administrative authorizations, as follows.

- **Requests affecting the authorizations on an object**
  These are requests for granting or revoking authorizations on an object. The user requesting them must have either the own or the administer privilege on the object.

  GRANT. To grant an access mode on an object to a subject for a specified time interval. The grant operation results in the addition of a new temporal authorization. The starting time of the authorization must be greater than or equal to the time at which the authorization is inserted (it is not possible to specify retroactive authorizations).

  DENY. To deny an access mode on an object to a subject for a given time interval. The deny operation results in the addition of a new temporal negative authorization.

  REVOKE. To revoke an access mode on an object from a subject. The revoke operation can be required with

reference to a single authorization by specifying its label (i.e., the deletion of a specific authorization is requested) or with reference to an access mode on an object with respect to a given time interval. The revoke operation results in the deletion or modification of all the temporal authorizations of the revokee for the access mode on the object granted by the user who revokes the privilege. If the time interval for which the revocation is requested spans from the time of the request to ∞ all authorizations for the access mode on the object granted by the revokee to the revoker will be deleted. If the revocation is required for a specific time interval, all the authorizations for the access mode on the object granted to the revokee by the revoker will be deleted or modified to exclude the interval (and possibly split in more authorizations). Note that a user can revoke only the authorizations he granted and then the revoke request by a user affects only the authorizations granted by that specific user.

REVOKE NEGATION. To revoke the negation for an access mode on an object from a subject. It is analogous to the Revoke operation with the only exception that it applies to negative authorizations.

- **Requests affecting rules**
  These are requests for specifying or deleting rules. The user requesting them must have either the `own` or the `administer` privilege on the object appearing at the left of the operator and either the `own`, `administer`, or `refer` privilege on the object appearing at the right of the operator.

  ADDRULE. To add a new derivation rule. The grantor of the authorization appearing at the left of the temporal operator 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.

  DROPRULE. To drop a derivation rule previously specified. The operation requires, as argument, the label of the rule to be deleted. Like for the revocation of authorizations, a user can drop only the rules that he has specified.

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

  GRANTADM. To grant the `administer` privilege on an object to a subject. It results in a new administrative authorization spanning from the time of the request to ∞.

  REVOKEADM. To revoke the `administer` privilege on an object to a subject. It results in: 1) the deletion of the authorization for the `administer` privilege on the object previously granted to the revokee, and 2) the deletion of the authorizations on the object and of the derivation rules where the object appears in the authorization at the left of the operator specified by the revokee. If the revokee does not have the reference privilege on the object, also the derivation rules where the object appears in the authorization at the right of the operator are deleted.

  GRANTREF. To grant the `refer` privilege on an object to a subject.

  REVOKEREF. To revoke the `refer` privilege on an object to a subject. It results in the deletion of the authorization for the `refer` privilege on the object previously granted to the subject and in the deletion of all the rules granted by the revokee where the object appears in the authorization at the right of the operator.

6 TAB_{EXT} Maintenance

Execution of administrative operations illustrated in the previous section can change the set of valid authorizations. The TAB_{EXT} has to be modified accordingly. For instance, the insertion of an explicit authorization can cause the deletion of authorizations from TAB_{EXT}. This happens if the authorization appears in the right side of a negative rule, or if it is a negative authorization. A similar problem arises for authorization deletion.

We have devised a set of algorithms that update TAB_{EXT} upon each administrative request, without the need of recomputing all the materialized authorizations. These algorithms use methods similar to those employed for the maintenance of materialized recursive views with negation [8].

The maintenance algorithms make use of sequences $\langle L_1 \ldots, L_k \rangle$ and $\langle \overline{X}_1, \ldots, \overline{X}_k \rangle$, defined in Section 4, that are permanently stored and updated by them to reflect the changes in TAB. The approach exploits the fact that, authorizations in $TAB_{EXT}^{(i)}$ are derived using only authorizations in $TAB_{EXT}^{(i-1)}$ and rules in $\overline{X}_i$. Thus, a change for an authorization/rule of level i does not affect authorization in $TAB_{EXT}^{(j)}$ with $j < i$. Only authorizations in $TAB_{EXT}^{(j)}$ with $j \geq i$ need to be reconsidered.

In the following, we illustrate an algorithm for updating TAB_{EXT} upon insertion of new positive authorizations, based on the Dred algorithm [8]. The methods to maintain TAB_{EXT} after the insertion/deletion of a negative authorization and the deletion of a positive one are very similar to that for positive authorizations insertion. We refer the reader to [3] for the description of these algorithms and for the ones for insertion/deletion of derivation rules.

6.1 Insertion of Explicit Positive Authorizations

The algorithm in Fig. 7 implements the maintenance of TAB_{EXT} for the insertion of an explicit positive authorization. It receives as input TAB_{EXT}, INST-TAB, its corresponding sequences $\langle L_1, \ldots, L_k \rangle$ and $\langle \overline{X}_1, \ldots, \overline{X}_k \rangle$ and a positive authorization and returns TAB_{EXT}^{u}, the set of valid authorizations resulting from the insertion of the positive authorization, and the updated sequences $\langle L'_1, \ldots, L'_{\bar{k}} \rangle$ and $\langle \overline{X}'_1, \ldots, \overline{X}'_{\bar{k}} \rangle$. The algorithm works as follows: Suppose that a positive authorization $([t_b, t_e], A_k)$ has been inserted. If the

inserted authorization does not appear in INST-TAB or its time interval exceeds t_{max}, it is necessary to recompute the sequence $\langle L_1 \ldots, L_k \rangle$ into which authorizations have been partitioned by the CSD algorithm, because the partition of the authorizations among the levels changes and the number of levels could increase (step 1). In step 2 the positive authorization is inserted in INST-TAB. Step 3 iteratively considers all the elements $\langle A, T \rangle$ in TAB_{EXT} and replaces each symbol '∞' in T with t_{max}. Step 4 initializes S_{INS}, S_{DEL}, and TAB^u_{EXT}. S_{INS} and S_{DEL} are two data structures containing the authorizations inserted and deleted from TAB^u_{EXT} till the current point of the computation. The authorizations are kept in S_{INS} and S_{DEL} using the same representation as for TAB_{EXT}. Then (step 5), the algorithm computes l_{min}, the least level in which authorization A_k appears in an instant t of the time interval $[t_b, t_e]$. All the operations for TAB_{EXT} maintenance will be executed starting from level l_{min}. Step 6 computes the sets $\overline{X}'_i$, for $i < l_{min}$. In this case $\overline{X}'_i = \overline{X}_i$, since the insertion of the new authorization does not change the levels of the authorizations in INST-TAB. Step 7 is an iteration on the levels returned by the CSD algorithm, starting from level l_{min}. For each level i, the algorithm performs the following operations:

- compute the set $\overline{X}'_i$, for $i > l_{min}$, by adding to $\overline{X}_i$ the element $\langle A_k, T_{k,i} \cap [t_b, t_e] \rangle$, where $T_{k,i} \cap [t_b, t_e]$ is the set of time intervals representing all the time instants $t \in [t_b, t_e]$ in which the inserted authorization is in level i;
- compute T, the set of time intervals representing all the time instants $t \in [t_b, t_e]$ in which the inserted authorization is in level i and it is not blocked by a negative authorization;
- insert the element $\langle A_k, T \rangle$ in S_{INS} and in TAB^u_{EXT};
- call function 'Dred-Ext()' that computes all the authorizations of level i which have to be inserted or removed from TAB^u_{EXT} because of the insertion of $([t_b, t_e], A_k)$.

Finally, the last step of the algorithm iteratively considers all the elements $\langle A, T \rangle$ in the updated TAB_{EXT} and substitutes each value t_{max} in T with symbol '∞'.

Function 'Dred-Ext(),' reported in Fig. 8, given a level l and the authorizations inserted and deleted from TAB^u_{EXT} till the current point in the computation, updates the TAB^u_{EXT} according to the rules that can be fired in level l. The function consists of three main steps: step a) adds to S_{DEL} and removes from TAB^u_{EXT} an overestimate of the authorizations that need to be deleted because of the insertion of $([t_b, t_e], A_k)$. An authorization is added to S_{DEL} by step a) if the insertion of $([t_b, t_e], A_k)$ invalidates any derivation of the authorization from the elements of $\overline{X}'_l$. Step b) reinserts in TAB^u_{EXT} the authorizations deleted in the previous step that have an alternative derivation. The reinserted authorizations are obviously removed from S_{DEL}. Finally step c) adds to S_{INS} and to TAB^u_{EXT} all the new authorizations that can be derived from the derivation rules in $\overline{X}'_l$ because of the insertion of $([t_b, t_e], A_k)$.

Algorithm 6.1 *Insertion of an explicit positive authorization*

INPUT: 1) A positive authorization $([t_b,t_e],A_k)$.
2) INST-TAB, TAB_{EXT} (each element denoted with $\langle A_i, T_i \rangle$).
3) The sequence $\langle L_1, \ldots, L_k \rangle$.
4) The sequence $\langle \overline{X}_1, \ldots, \overline{X}_k \rangle$ (see Algorithm 4.1).

OUTPUT: 1) TAB^u_{EXT} (the updated TAB_{EXT}).
2) The sequence $\langle L'_1, \ldots, L'_{k'} \rangle$.
3) The sequence $\langle \overline{X}'_1, \ldots, \overline{X}'_{\overline{k}} \rangle$.

METHOD:

1) If $A_k \notin$ INST-TAB or $t_e > t_{max}$ then $\langle L'_1, \ldots, L'_{k'} \rangle$:= CSD(INST-TAB $\cup$ $([t_b,t_e],A_k)$)
else $\overline{k} := k$, $L'_j := L_j$ $\forall j = 1, \ldots, k$
2) Insert $([t_b,t_e],A_k)$ in INST-TAB
3) For each $\langle A, T \rangle$ in TAB_{EXT} substitute each symbol '∞' in T with t_{max}
4) S_{INS} and S_{DEL} are initialized to be empty, $TAB^u_{EXT} := TAB_{EXT}$
5) $l_{min} := \min\{i \mid \langle A_k, T_{k,i} \rangle \in L'_i$ and $T_{k,i} \cap [t_b,t_e] \neq \emptyset\}$
6) $\overline{X}'_i := \overline{X}_i$, $\forall i = 1, \ldots, l_{min} - 1$
7) For i:=l_{min} to $\overline{k}$ Do
(a) If $\exists \overline{X}_i$ then $\overline{X}'_i := \overline{X}_i \cup \langle A_k, T_{k,i} \cap [t_b,t_e] \rangle$
else $\overline{X}'_i := \langle A_k, T_{k,i} \cap [t_b,t_e] \rangle$
(b) $T := \{T_{k,i} \cap [t_b,t_e]\} \setminus \bigcup \{T^n_m \mid pn(A_m) = '-', s(A_m) = s(A_k), o(A_m) = o(A_k), n(A_m) = n(A_k)\}$
(c) Add $\langle A_k, T \rangle$ to S_{INS} and TAB^u_{EXT}
(d) $\{T_I, T_D\}$:= Dred-Ext(S_{INS},S_{DEL},i)
(e) For each $\langle A, T \rangle \in T_I$ Add $\langle A, T \rangle$ to S_{INS}
(f) For each $\langle A, T \rangle \in T_D$ Add $\langle A, T \rangle$ to S_{DEL}
endfor
8) For each $\langle A, T \rangle \in TAB^u_{EXT}$, substitute each value $t_{max} \in T$, with symbol '∞'

Fig. 7. An algorithm for positive authorization insertion.

```
Function Dred-Ext(S_I,S_D,l)
(a) Repeat
    For each ⟨x,T'⟩ ∈ X̄'_l, where x = ([t_b,t_e],A_m⟨OP⟩A_n) or x = ([t_b,t_e],A_m) Do
      If pn(A_m) = '+' then S_1 := S_I
      else S_1 := ∅
      If x = ([t_b,t_e],A_m⟨OP⟩A_n) then
        Case OP of
            NEGOP:  S_2 := S_I
            POSOP:  S_2 := S_D
        endcase
      else S_2 := ∅
      If ⟨A_m,T''_k⟩ ∈ S_2 where T''_k ∩ T' ≠ ∅ or ⟨A_k,T'_k⟩ ∈ S_1, pn(A_k) = '−', s(A_k) = s(A_m), o(A_k) = o(A_m),
          n(A_k) = n(A_m), T'_k ∩ T' ≠ ∅ then
        T := Derive({R,T'},S_1,S_2,TAB_EXT)
        Add ⟨A_m,T⟩ to S_D
      endif
    endfor
  Until S_D does not change
  For each ⟨A,T⟩ ∈ S_D Delete ⟨A,T⟩ from TAB^u_EXT
(b) Repeat
    For each ⟨x,T'⟩ ∈ X̄'_l, where x = ([t_b,t_e],A_m⟨OP⟩A_n), or x = ([t_b,t_e],A_m), ⟨A_m,T''_m⟩ ∈ S_D with T''_m ∩ T' ≠ ∅ Do
      T := Derive-auth({x,T'},TAB^u_EXT)
      Delete ⟨A_m,T⟩ from S_D
      Add ⟨A_m,T⟩ to TAB^u_EXT
    endfor
  Until S_D does not change
(c) Repeat
    For each ⟨x,T'⟩ ∈ X̄'_l, where x = ([t_b,t_e],A_m⟨OP⟩A_n), or x = ([t_b,t_e],A_m) Do
      If pn(A_m) = '+' then S_1 := S_D
      else S_1 := ∅
      If x = ([t_b,t_e],A_m⟨OP⟩A_n) then
        Case OP of
            NEGOP:  S_2 := S_D
            POSOP:  S_2 := S_I
        endcase
      else S_2 := ∅
      If ⟨A_n,T''_n⟩ ∈ S_2 where T''_n ∩ T' ≠ ∅ or ⟨A_k,T'_k⟩ ∈ S_1, pn(A_k) = '−', s(A_k) = s(A_m), o(A_k) = o(A_m),
          n(A_k) = n(A_m), T'_k ∩ T' ≠ ∅ then
        T := Derive({R,T'},S_1,S_2,TAB^u_EXT)
        Add ⟨A_m,T⟩ to S_I and TAB^u_EXT
      endif
    endfor
  Until S_I does not change
  For each ⟨A,T⟩ ∈ TAB^u_EXT Delete ⟨A,T⟩ from S_D
  For each ⟨A,T⟩ ∈ TAB_EXT Delete ⟨A,T⟩ from S_I
  return ⟨S_I,S_D⟩
```

Fig. 8. **Function** Dred-Ext.

```
Function Derive(⟨x, T'⟩,S1,S2,From.TAB_EXT)
# x = ([t_b,t_e],A_m⟨OP⟩A_n) or x = ([t_b,t_e],A_m)
  T := Derive-auth(⟨x, T'⟩,From.TAB_EXT)
  If S2 ≠ ∅ then
    Case OF of
      WHENEVER,WHENEVERNOT: T̄' := T ∩ T̄^2_n
      ASLONGAS,UNLESS:      If T ≠ ∅ and T ∩ T^2_n ≠ ∅ then
                              t_k := min({t_i | t_i ∈(T^2_n ∩ T)})
                              t_e := max({t|t∈T})
                              T̄' := {[t_k,t_e]}
                            endif
    endcase
  else T̄' := ∅
  If S1 ≠ ∅ then T := T ∩ (⋃{T̄^1_k | pn(A_k) = '−', s(A_k) =s(A_m),o(A_k) =o(A_m)})
  T := T ∪ T̄'
  return T
```

Fig. 9. Function Derive.

THEOREM 6.1. *Given a* TAB *and a positive authorization* ($[t_b, t_e]$, A_k), 1) *Algorithm 6.1 terminates. Moreover* 2) *the sequence* $\langle \overline{X}'_1, \ldots, \overline{X}'_k \rangle$, *computed by the algorithm is correct. Finally,* 3) *the* TAB^u_{EXT}, *computed by the algorithm contains all and only the valid authorizations wrt* TAB $\cup$ ($[t_b, t_e]$, A_k)}.

The following example illustrates how Algorithm 6.1 works.

EXAMPLE 6.1. Consider the TAB illustrated in Example 3.3. The set of materialized authorizations and sets $\overline{X}_i$ for this TAB are illustrated in Example 4.1. Suppose that at time $t = 7$ authorization ([40, 50], A_3) is inserted in the TAB. It is not necessary to run the CSD algorithm since the upper bound of the time interval of the inserted authorization does not exceed $t_{max} = 201$, and A_3 already appears in INST-TAB. Since $l_{min} = 1$, then all the $\overline{X}'_i$, with $1 \le i \le 3$ will be considered.

After the first iteration of step (7) of Algorithm 6.1, $\overline{X}'_1 = \overline{X}_1 \cup \langle A_3, \{[40,50]\} \rangle$, $S_{INS} = \{\langle A_3, \{[40, 50]\}\rangle\}$, and $S_{DEL} = \emptyset$. Thus, function `Dred-Ext()' inserts $\langle [40, 50], A_3 \rangle$ in TAB^u_{EXT}.

In the second iteration, each element of $\overline{X}'_2 = \overline{X}_2$ is considered. Step (a) of function `Dred-Ext()' searches for elements $\langle R, T' \rangle$ in $\overline{X}'_2$ such that $R = ([t_b, t_e], A_m \langle \text{NEGOP} \rangle A_n)$, $\langle A_n, T^{INS}_n \rangle$ is in S_{INS} with $T^{INS}_n \cap T' \neq \emptyset$. The only element that satisfies the condition is $\langle A_3, \{[40, 50]\}\rangle$. Then $\langle \bar{A}_{2|John}, \{[40,50]\} \rangle$ is added to S_{DEL} and removed from TAB^u_{EXT}. In the third iteration, step (c) of function 'Dred-Ext()' is executed and $\{\langle A_2, \{[40, 50]\}\rangle\}$ is added to S_{INS}. No other changes will be made by this iteration. Hence, the algorithm terminates. The resulting TAB^u_{EXT} is:

$$TAB^u_{EXT} = \{\langle A_1, \{[10,200]\} \rangle, \langle A_2, \{[10,50],[61,100]\} \rangle, \langle \bar{A}_{2|John}, \{[51,60]\} \rangle, \langle A_3, \{[40,50]\} \rangle, \langle A_4, \{[10,80]\} \rangle\}.$$

7 Conclusions and Future Work

In this paper, we have presented an authorization model with temporal capabilities. The model introduces the concept of temporal authorization which is an authorization together with a start and an expiration time. Both negative as well as positive authorizations can be specified. Derivation rules can be expressed which allow new temporal authorizations to be derived on the basis of the presence or the absence of other temporal authorizations. Four different temporal operators can be used in the derivation rules. Administrative authorizations regulate the insertion and removal of authorizations and rules by users.

We have given the formal semantics of temporal authorizations and derivation rules in terms of a general logic program. The problem of ensuring the uniqueness of the derived authorizations corresponds to the theoretical issue of the existence of a unique model for the logic program. We have presented an approach to solve this problem based on the stratification of authorizations and derivation rules. We have provided an algorithm that determines whether an authorization base has a stratification and proved that, if the authorization base is stratified, a unique set of derived authorizations is always computed.

Performance issues have been addressed and a materialization approach in which derived authorizations are explicitly stored has been proposed. Algorithms for building the materialized set of derived authorizations and for maintaining them upon execution of administrative operations have been proposed.

The proposed model is currently under implementation to investigate the system's performance for various characteristics of the authorization base.

We are currently extending this work in several directions. First, decentralized authorization administration facilities are being added to the model. Second, the model is being extended with periodic authorizations. Such capability allows to specify, for example, that a given subject may access a data item every Thursday. Also, access control based on past access histories will be included into the model. Finally, we plan to investigate different temporal logic formalisms and constraint logic programming as possible foundations for temporal authorization models.

Acknowledgment

The authors wish to thank Prof. Michael Gelfond for useful discussions on problems related to the semantics of negation.

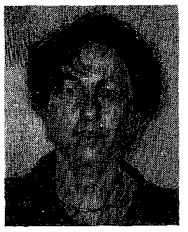
References

[1] M. Abadi, M. Burrows, B.W. Lampson, and G. Plotkin, "A calculus for access control in distributed systems," *ACM Trans. Programming Languages and Systems*, vol. 15, no. 4, pp. 706–734, Sept. 1993.

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

[3] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "A temporal access control model for database systems," Technical Report 137-95, DSI–Univ. of Milano, 1995.

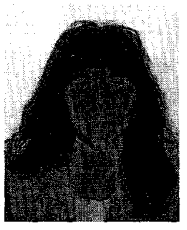
[4] E. Bertino, C. Bettini, and P. Samarati, "A temporal authorization model," *Proc. Second ACM Conf. Computer and Communications Security*, pp. 126–135, Fairfax,Va., Nov. 1994.
[5] E. Bertino, P. Samarati, and S. Jajodia, "Authorizations in relational database management systems," *Proc. First ACM Conf. Computer and Comm. Security*, Fairfax, Va., Nov. 1993.
[6] S. Ceri and J. Widom, "Deriving production rules for incremental view maintenance," *Proc. 17th VLDB*, pp. 735–749, Barcelona, 1991.
[7] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," *Proc. Fifth Int'l Conf. Logic Programming*, R. Kowalski and K. Bowen, eds., pp. 1,070–1,080, Cambridge, Mass.: MIT Press, 1988.
[8] I.S. Gupta, A. Mumick, and V.S. Subrahmanian, "Maintaining views incrementally," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 157–166, Portland, Ore., May 1993.
[9] W.T. Maimone and I.B. Greenberg, "Single-level multiversion schedulers for multilevel secure database systems," *Proc. Sixth Ann. Computer Security Applications Conf.*, pp. 137–147, Tucson, Ariz., Dec. 1990.
[10] J.G. Steiner, C. Neuman, and J.I. Schiller, "Kerberos: An authentication service for open network systems," *USENIX Conf. Proc.*, pp. 191–202, Dallas, Winter 1988.
[11] R.K. Thomas and R.S. Sandhu, "Discretionary access control in object-oriented databases: Issues and research directions," *Proc. 16th Nat'l Computer Security Conf.*, pp. 63–74, Baltimore, Sept. 1993.
[12] A. van Gelder, K. Ross, and J.S. Schlipf, "The well-founded semantics for general logic programs," *J. ACM*, vol. 38, no. 3, pp. 620–650, July 1991.
[13] T.Y.C. Woo and S.S. Lam, "Authorizations in distributed systems: A new approach," *J. Computer Security*, vol. 2, nos. 2-3, pp. 107–136, 1993.



Elisa Bertino is a professor of computer science at the Department of Computer Science of the University of Milan, Italy. She has also been a professor in the Department of Computer and Information Science of the University of Genova, Italy. Until 1990, she was a researcher for the Italian National Research Council in Pisa, Italy, where she headed the Object-Oriented Systems Group. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, California, at the Microelectronics and Computer Technology Corporation in Austin, Texas, and at George Mason University in Fairfax, Virginia.

Her main research interests include object-oriented databases, deductive databases, multimedia databases, interoperability of heterogeneous systems, integration of artificial intelligence and database techniques, and database security. Prof. Bertino is a coauthor of the book *Object-Oriented Database Systems—Concepts and Architectures* (Addison-Wesley International, 1993), and a coauthor of the forthcoming books *Principles of Database Security* (Benjamin/Cummings) and *Intelligent Database Systems* (Addison-Wesley International). She has participated in several research projects sponsored by the Italian National Research Council and the European Economic Communities. She is currently serving as program chair of the 1996 European Symposium on Research in Computer Security (ESORICS'96). She is a member of the editorial boards of *IEEE Transactions on Knowledge and Data Engineering*, the *International Journal of Theory and Practice of Object Systems*, and the *Journal of Computer Security*. She is member of ACM and the IEEE.

Claudio Bettini received an MS degree in information sciences in 1987 and a PhD in computer science in 1993, from the University of Milan, Italy. He has been an assistant professor in the Computer Science Department of the University of Milan since 1993. His main research interests include temporal logics, description logics, temporal reasoning in knowledge and data bases, and temporal aspects of database security. On these topics he has published several papers. He has been a visiting researcher at IBM Kingston, New York, and at George Mason University, Virginia.



Elena Ferrari received an MS degree in computer science from the University of Milan, Italy, in 1992. Since December 1993, she has been a PhD candidate in the Department of Computer Science of the University of Milan. Her research interests include authorization models and temporal databases. Her current research activity concerns extensions of authorization models with temporal capabilities. She is also investigating formal temporal object-oriented data models able to represent temporal evolution of objects.

Pierangela Samarati is an assistant professor of computer science at the University of Milan. Her main research interests are information systems security, database security, authorization models, and databases. On these topics, she has published several papers. She has been a visiting researcher at Stanford University, California, and at George Mason University, Virginia. She is currently serving as the the Italian representative on the IFIP TC-11 (Technical Committee 11 on Security and Protection in Information Processing Systems). She is a coauthor of the book *Database Security* (Addison-Wesley, 1995).