An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning

Elisa Bertino, Claudio Bettini, Elena Ferrari and Pierangela Samarati
University of Milano

Access control models, such as the ones supported by commercial DBMSs, are not yet able to fully meet many application needs. An important requirement derives from the temporal dimension that permissions have in many real-world situations. Permissions are often limited in time or may hold only for specific periods of time. In this paper, we present an access control model in which periodic temporal intervals are associated with authorizations. An authorization is automatically granted in the specified intervals and revoked when such intervals expire. Deductive temporal rules with periodicity and order constraints are provided to derive new authorizations based on the presence or absence of other authorizations in specific periods of time. We provide a solution to the problem of ensuring the uniqueness of the global set of valid authorizations derivable at each instant, and we propose an algorithm to compute this set. Moreover, we address issues related to the efficiency of access control by adopting a materialization approach. The resulting model provides a high degree of flexibility and supports the specification of several protection requirements that cannot be expressed in traditional access control models.

Categories and Subject Descriptors: H.2.7 [**Information Systems**]: Database Administration—*Security, integrity, and protection*

General Terms: Database Security, Time Management

Additional Key Words and Phrases: Access control, temporal constraints, periodic authorization

1. INTRODUCTION

Data protection from unauthorized accesses is becoming more and more crucial as an increasing number of organizations entrust their data to database systems. Moreover, as organizations and users are becoming more aware of data security issues, more articulate access control policies are being devised. An access control policy establishes for each user (or group of users, or functional role within the organization) the actions the user can perform on each object (or set of objects)

This paper extends the previous work by the authors appeared on *IEEE Transactions on Knowledge and Data Engineering*, 8(1), pp. 67-80, ©1996 The Institute of Electrical and Electronics Engineers, Inc.
Address: Dipartimento di Scienze dell'Informazione, University of Milano, via Comelico 39, I-20135 Milano, Italy. {bertino,bettini,ferrarie,samarati}@dsi.unimi.it.

within the system under which circumstances. An example of a policy is that "all programmers can modify the project files every working day except Friday afternoons". Once the organization security policies are devised, they are implemented in terms of the *access control model* of the DBMS at hand.

The implementation of the security policy in terms of access control models, provided as part of a typical current DBMS, is however quite a difficult task. Current commercial DBMSs are still rather poor with respect to expressiveness of security requirements and therefore unable to directly support many relevant application policies [Jajodia et al. 1997]. Even the simple above requirement cannot be directly supported by any current commercial DBMS. As a matter of fact, in most cases the access control policies must be implemented as code in application programs. Such an approach makes it very difficult to verify and modify the access control policies and to provide any assurance that these policies are actually enforced.

An important requirement, common to many application security policies, is related to the temporal dimension of access permissions. In many real-world situations, permissions may hold only for specific time intervals. A further requirement concerns periodic authorizations. In many organizations, authorizations given to users must be tailored to the pattern of their activities within the organization. Therefore, users must be given access authorizations to data only for the time periods in which they are expected to need the data. The requirement above concerning programmers and project files is an example of specification requiring an authorization mechanism able to support periodic authorizations. As another example, consider part-time staff that should be authorized for accesses only on working days between 9 a.m. and 1 p.m.

Periodic authorizations are also crucial for optimizing resource usage. In particular, authorizations for application programs, whose execution is very resource-expensive, could be assigned only for specific time periods in which other programs are not likely to be executed. Periodic authorizations are, however, even more difficult to handle than simple, non-periodic, temporal authorizations. Therefore, also for periodic authorizations, the solution of implementing them as part of application programs is not viable.

The development of a temporal authorization model entails several issues, including the definition of a formal semantics for the authorization model, the development of strategies for efficient access control, and the development of tools for authorization administration. In this paper we address some of these issues by proposing an access control model characterized by temporal authorizations. In the proposed model, a temporal expression is associated with each authorization, identifying the instants in which the authorization applies. The temporal expression is formed by a periodic expression (e.g., `9 a.m.-to-1 p.m. on Working-days`, identifying the periods from 9a.m. to 1p.m. in all days excluding weekends and vacations), and a temporal interval bounding the scope of the periodic expression (e.g., `[2/1997,8/1997]`, restricting the above periods to those between February and August 1997).

Another relevant feature provided by our model is the possibility of specifying derivation rules expressing temporal dependencies among authorizations. These rules allow the derivation of new authorizations based on the presence or absence of other authorizations in specific periods of time. By using derivation rules, many protection requirements can be concisely and clearly specified. For example, it is possible to specify that two users, working on the same project, must receive the same authorizations on certain types of objects, or that a user should receive the authorization to access an object in certain periods, only if nobody else was ever authorized to access the same object in any instant within those periods. Derivation rules are specified by constraining the rule application with a temporal expression, by providing the authorization to be derived, by specifying one of the three temporal dependency operators that the model provides, and, finally, by giving the body of the rule in the form of a boolean expression of authorizations. The three temporal operators correspond to the three main temporal relations between authorizations that we have identified in common protection requirements.

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 to positive authorizations and for supporting a stricter control in the case of decentralized authorization administration [Bertino et al. 1993]. The combination of positive/negative authorizations with temporal authorizations results in a powerful yet flexible authorization model.

A formal semantics has been defined for temporal authorizations and derivation rules, based on the semantics of Datalog programs with negation and periodicity and order constraints. A critical issue is represented by the presence of negation in our derivation rules. Negation, by allowing the derivation of new authorizations based on the absence of other authorizations, augments the expressive power of the model. However, it does not always ensure the derivation of a unique set of authorizations, since the set of authorizations derivable from a given set of authorizations and rules may depend 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.

Finally, we address the problem of efficient access control by proposing a strategy, based on view materialization approaches. Our approach, which is based on a combination of the Dred [Gupta et al. 1993] and Stdel [Lu et al. 1995] approaches, avoids the high costs arising from the evaluation of the deductive rules when performing access control.

Previous related work

We first presented a proposal for an authorization model supporting authorizations with temporal intervals and a restricted set of derivation rules in [Bertino et al. 1996a]. The current paper extends our previous work introducing, among other features, the support for periodic access authorizations and periodic rules. This is a major extension, both for the practical relevance of periodic expressions in specifying authorizations and for the related theoretical and performance issues. The language for expressing derivation rules has also two major extensions with respect to the one supported by our previous model. The first is the introduction of a new temporal operator (UPON) and the second is the possibility of using an arbitrary boolean expression of authorizations as the right hand side of a rule. These extensions significantly augment the expressiveness of the model in terms of protection requirements that can be supported. A preliminary approach towards the introduction of periodic constraints has been presented by us in [Bertino et al. 1996b]. However, boolean operators where not considered in [Bertino et al. 1996b], nor access control strategies were devised. All the relevant theoretical foundations are only reported in this paper. To the best of our knowledge, the authorization model we are presenting is the first one proposing features such as periodic authorizations and derivation rules.

In the context of database systems, Ingres [Date 1995] supports some temporal features. In Ingres, when granting an authorization, a user can specify that the authorization is valid only in specific hours and days of the week. For instance, it is possible to give a user the authorization to read a table only between 8:00am and 5:00pm between Tuesday and Thursday. Temporal specifications in Ingres are conditions which can be evaluated, upon access control, by checking the time/day of the request against that of the system's clock. Although representing a first step towards the inclusion of temporal features in authorization specification, Ingres provides only a rudimentary treatment of time-based authorizations. For instance, no start or expiration date can be associated with authorizations and no calendar management, time reasoning, or authorization derivation features are supported.

Other relevant related work can be found in the framework of authentication systems. Kerberos [Steiner et al. 1988], based on a client-server architecture, provides the notion of *ticket*, with an associated validity time. The span of the validity time is chosen by the administrator based on a trade-off between security and convenience (a typical choice is a span of 8-24 hours for all tickets). The ticket is used by a client to require a service from a particular server. Associating a validity time to the ticket saves the client from the need to acquire a ticket for each interaction with the same server. Thus, the scope of the temporal ticket mechanism is very different from our access control model. In Kerberos the ticket is only used to denote that a client has been authenticated by the authentication server, and cannot be used to grant access to specific documents or resources managed by the server.

From the side of logical formalisms for security specifications, Woo and Lam in [Woo and Lam 1993] propose a very general formalism for expressing authorization rules. Their language does not have explicit constraints to deal with temporal information. However, the generality of their language, which has almost the same expressive power of first order logic, makes it possible to model temporal constraints. The main drawback is that the trade-off between expressiveness and efficiency seems to be strongly unbalanced in their approach. Finally, Abadi et al. [Abadi et al. 1993] proposed a formal language based on modal logic. However, their language, which is mainly devoted to the modeling of concepts such as roles and delegation of authorities, is not able to support temporal constraints.

Organization of the paper

The remainder of this paper is organized as follows. Section 2 describes the formalisms we use to represent periodic time. Section 3 introduces periodic authorizations and derivation rules and gives the formal semantics of our model. Section 4 deals with the problems due to the presence of negation. A sufficient condition to guarantee the uniqueness of the set of derived authorizations, and an algorithm for checking this condition are given. In Section 5 an algorithm for deriving the set of implicit and explicit authorizations is presented. Administrative operations are discussed in Section 6. In this section we also discuss implementation issues. Section 7 concludes the paper and outline future work. Appendix A illustrates the Datalog extension we use to represent the semantics of our rules. Finally, formal proofs are reported in Appendix B.

2. PRELIMINARIES

To represent periodic authorizations we need a formalism to denote periodic time. Our choice is to provide a symbolic (user friendly) formalism for the user that has to specify authorizations and an equivalent "mathematical" formalism for internal representation. The symbolic formalism consists of a pair $\langle$`[begin, end]`, `P`$\rangle$ where `P` is a *periodic expression* denoting an infinite set of time intervals, and `[begin, end]` denotes the lower and upper bounds that are imposed on time intervals in `P`. The formalism for internal representation is based on sets of periodicity and gap-order constraints over integer numbers and it is inspired by the work in [Toman et al. 1994]. A mapping from symbolic periodic expressions into these types of constraints is needed to describe the semantics of periodic authorizations and rules, to prove formal properties of our model, and to perform deductive reasoning.

In the following we take as our model of time the integers Z with the total order relation $<$.

2.1 Symbolic expressions

The formalism for symbolic periodic expressions is essentially the one proposed in [Niezette and Stevenne 1992], based on the notion of *calendars*. A calendar is defined as a countable set of consecutive intervals. Each interval of a calendar is numbered by an integer number, called *index* of the interval, in such a way that successive intervals are numbered by successive integers.

Example 1. *Days*, *Months*, *Years* are examples of calendars representing respectively the set of all the days, the months and the years.

New calendars can be dynamically constructed from the existing ones. The starting point in the definition of new calendars is the definition of a basic calendar (the tick of the system), from which other calendars can be dynamically defined. In the following, we assume that *Hours* is the basic calendar with the tick indexed by 1 as the first hour of 1/1/94. A new calendar C_1 can be defined from an existing calendar C_0 by means of function '$generate()$'. $C_1 = generate(sp; C_0; (x_1, \ldots, x_n))$ creates a calendar C_1 whose first tick is the union of the first x_1 ticks of calendar C_0, the second tick is the union of the following x_2 ticks of C_0, etc. cycling through the given values. Parameter *sp* is the synchronization point, that is, it is the index of the interval of C_0 which starts at the same time as the first interval of C_1.

Example 2. Starting from the basic calendar *Hours*, the following calendars can be defined:

—$Days = generate(1;Hours;(24))$;

—$Months = generate(1;Days;\ (31,28,\ldots,31,28,\ldots,31,29,\ldots,31,28,\ldots,31))$;

—$Weeks = generate(2;Days;(7))$;

—$Years = generate(1;Days;(365,365,366,365))$;[1]

Note that, in the definition of *Weeks*, 2 is used as the synchronization point since the first full week of 1994 starts on Sunday, January 2nd. The definition of *Years* says to take the first 365 days as the first tick of the calendar, the following 365 days as the second tick, etc., cycling through the given values.

In our model, we postulate the existence of a set of predefined calendars containing at least the calendars *Hours*, *Days*, *Weeks*, *Months*, and *Years*. Given two calendars C_1 and C_2, C_1 is called a subcalendar of C_2, (written $C_1 \sqsubseteq C_2$), if each interval of C_2 is exactly covered by a finite number of intervals of C_1. It is easy to show that the subcalendar relation defines a partial order on calendars, and that, since we take *Hours* as our basic calendar, $Hours \sqsubseteq C$ for each calendar C defined in our system.

Calendars can be combined to represent more general sets of periodic intervals, not necessarily contiguous, like, for instance, the set of *Mondays* or the set of *The third hour of the first day of each month.* Complex sets of periodic intervals, like the ones above, are represented by means of *periodic expressions*, formally defined as follows.

Definition 1. Given calendars $C_d, C_1, \ldots, C_n$, a *periodic expression* is defined as $\mathtt{P} = \sum_{i=1}^{n} O_i.C_i \rhd r.C_d$, where $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$ and $C_i \sqsubseteq C_{i-1}$ for $i = 2, \ldots, n$, $C_d \sqsubseteq C_n$, and $r \in \mathbb{N}$.

The symbol $\rhd$ separates the first part of the expression, identifying the set of starting points of the intervals it represents, from the specification of the duration of each interval in terms of calendar C_d. For example, $all.Years + \{3, 7\}.Months \rhd 2.Months$ represents the set of intervals starting at the same instant as the third and seventh month of every year, and having a duration of 2 months. In practice, O_i is omitted when its value is *all*, whereas it is represented by its unique element when it is a singleton. $r.C_d$ is omitted when it is equal to $1.C_n$. Table 1 reports a set of periodic expressions with their intuitive meaning.

Periodic expressions are a symbolic representation of infinite sets of periodic intervals. The set of time intervals corresponding to periodic expressions is formalized by function $\Pi()$, defined as follows.

Definition 2. Let $\mathtt{P} = \sum_{i=1}^{n} O_i.C_i \rhd r.C_d$ be a periodic expression, then $\Pi(\mathtt{P})$ is a set of time intervals whose common duration is $r \cdot C_d$, and whose set S of starting points is computed as follows.

—If n=1, S contains all the starting points of the intervals of calendar C_1.

—If $n > 1$, and $O_n = \{n_1, \ldots, n_k\}$, then S contains the starting points of the $n_1^{th}, \ldots, n_k^{th}$ intervals (all intervals if $O_n = all$) of calendar C_n included in each interval of $\Pi(\sum_{i=1}^{n-1} O_i.C_i \rhd 1.C_{n-1})$.

For example, if $\mathtt{P}$ is the last expression in Table 1, then $\Pi(\mathtt{P})$ is the set of time intervals, with a duration of 4 hours, starting with the tenth hour (9a.m. to 10a.m.) of the second, third, fourth, fifth, and sixth day of every week.

[1]The definitions of *Months* and *Years* are not precise, since a period of 400 years should be considered to take into account all the exceptions for leap years.

Table 1. Example of periodic expressions

periodic expression	**meaning**
$Weeks + \{2,6\}.Days$	Mondays and Fridays
$Months + 20.Days$	The twentieth of every month (Pay-days)
$Years + 7.Months \triangleright 3.Months$	Summer-time
$Weeks + \{2,\ldots,6\}.Days$	Working-days
$Weeks + \{2,\ldots,6\}.Days + 10.Hours \triangleright 4.Hours$	Between 9 a.m. and 1 p.m. of working-days

A symbolic formalism is also needed to express the bounds `begin` and `end` which limit the application of the periodic expression. This formalism must guarantee that each expression identifies a single instant in the basic calendar, or, possibly, the special values $+/-\infty$. While the results of this paper do not rely on a particular choice for this formalism, for simplicity we choose a simple date notation.

Definition 3. A *date expression* has the form `mm/dd/yy:hh`, where `mm`$\in \{1, \ldots, 12\}$, `dd`$\in \{1, \ldots, 31\}$, `yy`$\in \{00, \ldots, 99\}$, and `hh`$\in \{01, \ldots, 24\}$.

A date expression denotes a single tick of calendar *Hours*, according to the intuitive semantics.[2]

We require `begin` to be a date expression, and `end` to be ∞ or a date expression. We sometimes abbreviate date expressions. The intuitive semantics is applied; for example, when 1/1/94 is used as `begin`, it denotes the first instant of the first day of January 1994, while, as `end`, it denotes the last instant of 1/1/94.

2.2 Periodicity and gap-order constraints

Symbolic expressions, although convenient for the users, are not easy to manipulate in the deductive process. For this reason, we translate expressions given by the user into expressions in a different formalism. This formalism is based on sets of constraints. Intervals `[begin,end]`, consisting of date expressions, are straightforwardly translated into so-called *gap-order* constraints [Revesz 1995].

Definition 4. Let u, l be integers, c be a non-negative integer, and t, t' be integer variables. A *gap-order constraint* is a formula of the form $l < t$, $t < u$, $t = t'$, or $t + c < t'$.

If `begin` and `end` denote instant $\mathtt{t}_b$ and $\mathtt{t}_e$ respectively, the corresponding constraints are $c_1 < t$ and $t < c_2$, where the constants c_1 and c_2 are respectively $\mathtt{t}_b - 1$ and $\mathtt{t}_e + 1$. For brevity, we often write $\mathtt{t}_b \leq t \leq \mathtt{t}_e$ for the conjunction of the constraints.[3] Gap-order constraints involving two variables will also be used in the reasoning process.

To manipulate periodic expressions we use *periodicity constraints* over integer numbers, as introduced in [Toman et al. 1994]. Periodicity constraints denote infinite periodic sets of integers.

Definition 5. Let K be a finite set of natural numbers, t an integer variable, k an element of K, and $c \in \{0, \ldots, k-1\}$. A *simple periodicity constraint* is a formula of the form: $t \equiv_k c$.

[2]In our examples, we consider `yy` corresponding to year 19`yy` if `yy`≥ 94 and to year 20`yy` if `yy`< 94.
[3]If `end`$=\infty$, no constraint for the upper bound is needed.

Periodicity constraint $t \equiv_k c$ denotes the set of integers of the form $c + nk$, with n ranging from $-\infty$ to $+\infty$ in Z. In the following we use the notation $t \equiv_k (y + c) \forall y = 0, \ldots, u$ as a compact representation for the disjunction of simple constraints: $t \equiv_k c \vee t \equiv_k c + 1 \vee \ldots \vee t \equiv_k c + u$.

In our model, each access authorization is associated with a periodic expression, and with a pair of date expressions defining the lower and upper bounds of its applicability. As it will be shown in the next subsection, a periodic expression corresponds to a disjunction of simple periodicity constraints, and the lower and upper bounds can be represented by a conjunction of gap-order constraints. Hence, it could be natural to associate with each authorization a complex constraint with these two components. However, since the reasoning process requires to manipulate these constraints by the operations of complement, conjunction, and projection, we need an adequate normal form. For this purpose, we introduce a *temporal constraint* Ξ that is represented by the set $\{(PC_1, GC_1), \ldots, (PC_m, GC_m)\}$, where each PC_i is a conjunction of simple periodicity constraints and each GC_i is a conjunction of gap-order constraints. Ξ is associated with a propositional logic formula in disjunctive normal form $\mathcal{F}(\Xi)$=$(PC_1 \wedge GC_1) \vee \ldots \vee (PC_m \wedge GC_m)$, corresponding to the intuitive semantics of Ξ. Ξ denotes all the instants t such that there exists $i \in [1, m]$ with t satisfying both PC_i and GC_i.

Some basic operations have been defined for simple periodicity constraints in [Toman et al. 1994] and for gap-order constraints in [Revesz 1993], using a graph for the representation of PC's and GC's. For example, rules are given to combine two constraints on the same variables, either detecting an inconsistency or deriving a new constraint equivalent to the conjunction of the given ones.

We now define conjunction ($\wedge^*$) and complement ($\neg^*$) on temporal constraints.

Definition 6. Given two temporal constraints Ξ_1 and Ξ_2, $\Xi = \Xi_1 \wedge^* \Xi_2$ is the set of pairs (PC, GC) whose associated formula $\mathcal{F}(\Xi)$ is the disjunctive normal form of $\mathcal{F}(\Xi_1) \wedge \mathcal{F}(\Xi_2)$.

Definition 7. Let Ξ' be a set of pairs (PC, GC). $\Xi = \neg^* \Xi'$ is the set of pairs (PC, GC) whose associated formula $\mathcal{F}(\Xi)$ is the disjunctive normal form of $\neg \mathcal{F}(\Xi)$.

Any propositional logic formula on PC's and GC's in disjunctive normal form, resulting from the above operations, can be represented as a temporal constraint Ξ. Indeed, each pair in Ξ corresponds to a disjunct in the formula. If no PC or no GC appears in a disjunct, the always satisfied constraint {true} is used in its place in the corresponding pair of Ξ.

Example 3. Let $\Xi = \{(PC_1, GC_1),\ (PC_2, GC_2)\}$ and $\Xi' = \{(PC'_1, GC'_1),\ (PC'_2, GC'_2)\}$.

$$\begin{aligned}\Xi \wedge^* \Xi' = \{&(PC_1 \wedge PC'_1, GC_1 \wedge GC'_1),\ (PC_2 \wedge PC'_1, GC_2 \wedge GC'_1),\\ &(PC_1 \wedge PC'_2, GC_1 \wedge GC'_2),\ (PC_2 \wedge PC'_2, GC_2 \wedge GC'_2)\ \};\\ \neg^* \Xi = \{\ &(\neg PC_1 \wedge \neg PC_2, \{\text{true}\}),\ (\{\text{true}\}, \neg GC_1 \wedge \neg GC_2),\\ &(\neg PC_1, \neg GC_2),\ (\neg PC_2, \neg GC_1)\ \}.\end{aligned}$$

Note that conjunction among PC's and among $GC's$ is defined among the operations in [Toman et al. 1994; Revesz 1993] and that negation can be easily eliminated. For example, if $PC = (t \equiv_k c \wedge t' \equiv_{k'} c')$, $\neg PC$ is the disjunction of the constraints (PC's) $t \equiv_k r$ with $r = 0, \ldots, c-1, c+1, \ldots, k-1$, and $t' \equiv_{k'} s$ with $s = 0, \ldots, c'-1, c'+1, \ldots, k'-1$.

Example 4. Consider the periodicity constraints $PC_1 = (t \equiv_7 1 \wedge t' \equiv_{10} 1)$ and $PC_2 = (t \equiv_{14} 1)$. $PC_1 \wedge PC_2 = (t \equiv_1 41 \wedge t' \equiv_{10} 1)$ whereas $\neg PC_1$ is the disjunction of the following PC's: $t \equiv_7 0$, $t \equiv_7 2$, ..., $t \equiv_7 6$, $t' \equiv_{10} 0$, $t' \equiv_{10} 2$, ..., $t' \equiv_{10} 9$.

2.3 From symbolic expressions to constraints

The following proposition states the correspondence between periodic expressions and sets of periodicity constraints.

PROPOSITION 1. *Any symbolic periodic expression can be translated into an equivalent set of simple periodicity constraints.*

In [Niezette and Stevenne 1992], Niezette and Stevenne show how any calendar can be translated into a union of *linear repeating intervals*. It is easily seen that the translation can be extended to periodic expressions.[4] A linear repeating interval is a mathematical expression of the form $kn + (b, e)$ denoting the set of intervals including the interval (b, e) and all intervals obtained by shifting (b, e) by multiples of k. Any instant t in the intervals defined by $kn + (b, e)$ satisfies one of the constraints $t \equiv_k (b + y)\ \forall y = 0, \ldots, e - b$ and vice versa. Hence, for each periodic expression P, there exists a disjunction of simple periodicity constraints such that the set of its solutions is the set of instants contained in the intervals of Π(P).

When the intervals in Π(P) have the same length, the simple periodicity constraints corresponding to P can be represented in a compact way as follows:

$t \equiv_{Periodicity(\mathtt{P})} (y + z - 2)\quad \forall y = 1, \ldots, Granularity(\mathtt{P})$, and $\forall z \in Displacement(\mathtt{P})$.

The values for the parameters in this formula depend on the calendar used to express the constraints, and on the definition of the calendars appearing in P. For example, if we express the constraints in terms of *Hours*, then the expression *Weeks* + $\{3\}$.*Days*, denoting *Tuesdays*, is translated into $t \equiv_{168} (y + 73 - 2)\ \forall y = 1, \ldots, 24$, where: 168 is the number of hours in a week (the periodicity of Tuesdays), $\{73\}$ is the only hour corresponding to the beginning of a Tuesday in each period,[5] and 24 is the number of hours within each Tuesday (the granularity of Tuesdays).

In general, given a symbolic expression P and the basic calendar, the values for *Periodicity*(P), *Displacement*(P), and *Granularity*(P) can be derived as follows:

—*Periodicity*(P) is the number n of units of the basic calendar that identifies the periodicity with which the time intervals in Π(P) repeat themselves. For example, with reference to Table 1, assuming *Hours* as the basic calendar, the periodicity of the first, fourth, and fifth expressions is 168 (a week expressed in hours), and

[4] It can be shown that any periodic expression $\sum_{i=1}^{n} O_i.C_i \rhd r.C_d$ can be reduced to the form $all.C \rhd r.C_d$ where C is a calendar. Then, if C is defined by $generate(\mathrm{sp}; C_d; (l_1, \ldots, l_s))$, the corresponding disjunction of *linear repeating intervals* is given by the formula: $\bigcup_{i=1}^{s} period * n + (sp + \sum_{j=1}^{i-1} l_j, sp + \sum_{j=1}^{i-1} l_j + r - 1)$, where $period$ is the periodicity of C in terms of C_d.

[5] Note that, since we fixed hour 1 as the first hour of 1/1/1994, the period always starts on a Saturday.

the periodicity of the second is $3,506,328$ (400 years in terms of hours; indeed, this is the period considering leap years).

—*Displacement*(P) is a set of numbers, each one representing the position within a period where a segment of the span of time defined by P begins. We assume that the first period always starts from tick 1 of the basic calendar. For example, in Table 1, the displacement of the first expression is $\{2*24+1, 6*24+1\}$, while the displacement of the second is $\{19*24+1, 50*24+1, 78*24+1, \ldots\}$ (each position of the beginning of the 20th day of each month within the period must be specified).

—*Granularity*(P) is the length of each segment of time within the period defined by P. The granularity is expressed using the basic calendar, and it can be easily derived from the part of P following the $\triangleright$ symbol (note that this part can be implicit). For example, in Table 1, the granularity of the first, second, and fourth expression is 24 (1 day in hours), while the granularity of the fifth is 4 (4 hours).

When the intervals in Π(P) have different length, there is no unique value for *Granularity*(P). In this case a simple solution is to partition Π(P) in sets of equal-length intervals, and representing each of them as shown above.

3. THE AUTHORIZATION MODEL

In this section we illustrate the basic components of our authorization model. We do not make any assumption on the underlying data model and on the access modes users can exercise on the data objects. This generality makes our authorization model applicable to the protection of information represented with different data models. Therefore, 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 roles (e.g., staff or manager).

3.1 Periodic authorizations and rules

Our model supports the specification of periodic authorizations, that is, authorizations that hold in specific periodic intervals specified by a periodic expression. A time interval is also associated with each authorization, imposing lower and upper bounds to the potentially infinite set of instants denoted by the periodic expression. Periodic authorizations can be positive or negative. Positive authorizations represent permission, whereas negative authorizations represent explicit denials to exercise a privilege on a given object.

We start by introducing the definition of authorization.

Definition 8. An *authorization* is a 5-tuple (s,o,m,pn,g), with s, g$\in U$, o$\in O$, m$\in M$, pn $\in \{+,-\}$.

For instance, by authorization (Ann,o_1,read,+,Tom) Tom grants Ann the read access mode on object o_1. By contrast, by authorization (John,o_1,write,-,Tom) Tom prevents John from writing object o_1.

As already stated, our model supports periodic authorizations, that is, authorizations together with a time interval and a periodic expression. Periodic authorizations are formally defined as follows.

Definition 9. A *periodic authorization* is a triple `([begin,end], P, auth)`, where `begin` is a date expression, `end` is a date expression denoting an instant greater than or equal to the one denoted by `begin`, or `end` is the constant ∞, `P` is a periodic expression, and `auth` is an authorization.

The periodic authorization `([begin,end],P,(s,o,m,pn,g))`, states that authorization `(s,o,m,pn,g)` is granted for *each* instant in Π(P) that is greater than or equal to the instant t_b denoted by `begin` and smaller than or equal to the instant t_e denoted by `end` (if end$\neq \infty$).

For example, the periodic authorization A_1 = ([1/1/94,∞], Mondays[6], (Matt, o_1, read, +, Bob)), specified by Bob, states that Matt has the authorization to read o_1 each Monday starting from 1/1/94.

A non periodic temporal authorization, that is, an authorization that holds continuously for a specific set of time instants, is expressed by a periodic authorization using the basic calendar as the `period` component. In general, the symbol $\perp$ is used to denote this calendar. Note that the corresponding periodicity constraint is $t \equiv_1 0$, that is equivalent to *true*, since it is always satisfied.

In what follows, given a periodic authorization A = ([begin,end],P,(s,o,m,pn, g)) we use s(A),o(A),m(A),pn(A),g(A) to denote respectively the subject, the object, the privilege, the sign of the authorization (positive or negative), and the grantor in A.

Note that the possibility of expressing negative authorizations introduces potential conflicts. Suppose that authorization A_2 = ([1/1/95,∞], Working-days, (Matt,o_1,read,-,Tom)) is specified in addition to authorization A_1. We then have for each Monday in the interval [1/1/95,∞] both a positive and a negative authorization with the same subject, object, and access mode. We solve this conflict according to the denials-take-precedence principle [Jajodia et al. 1997]. As a consequence, Matt will be allowed to read o_1 only for the Mondays in [1/1/94, 12/31/94]. We say that a positive authorization A is *valid* at a given instant t, if *i)* a temporal authorization ([begin,end],P,A), with $t \in \Pi(P) \cap \{[t_b,t_e]\}$,[7] is specified or it can be derived through the derivation rules, and *ii)* a negative authorization ([begin$'$,end$'$],P$'$,A$'$), with the same subject, object, access mode as A such that $t \in \Pi(P') \cap \{[t'_b,t'_e]\}$ is neither specified nor can be derived through the derivation rules, where begin$'$ and end$'$ denote respectively instants t'_b and t'_e. We say that a negative authorization A is valid at time t if a temporal authorization ([begin,end],P,A)), with $t \in \Pi(P) \cap \{[t_b,t_e]\}$ is specified or can be derived through the derivation rules. The notion of validity will be formalized in Section 3.3.

The second basic component of our model is an inference mechanism based on *derivation rules*, which express temporal dependencies among authorizations. Derivation rules allow the system to derive new periodic authorizations on the basis of the validity or non validity of other periodic authorizations. Like authorizations,

[6]Here and in the following we use intuitive names for periodic expressions, assuming that they are defined with the syntax shown in Section 2.

[7]We use a set of disjoint 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 operation of intersection ($T_1 \cap T_2$) has the usual semantics of set intersection.

```
(A1) ([95,5/20/95], ⊥,(manager,guidelines,write,+,Sam))
(A2) ([10/1/95,∞], Working-days,(technical-staff,guidelines,read,+,Sam))
(A3) ([95,97],Working-days,(staff,document,read,+,Sam))
(A4) ([95,∞],Pay-days,(Tom,pay-checks,write,+,Sam))
(A5) ([96,97],Summer-time,(technical-staff,document,read,+,Sam))
(R1) ([96,98],Working-days,(temporary-staff,document,read,+,Sam) ASLONGAS
      ¬(summer-staff,document,read,+,Sam))
(R2) ([95,∞],Mondays and Fridays,(technical-staff,report,write,+,Sam) UPON
      ¬((manager,guidelines,write,+,Sam) ∨ (staff,guidelines,write,+,Sam)))
(R3) ([95,∞],⊥,(technical-staff,report,write,-,Sam) WHENEVER
      ¬(technical-staff,guidelines,read,+,Sam))
(R4) ([95,∞],Summer-time,(summer-staff,document,read,+,Sam) WHENEVER
      (staff,document,read,+,Sam) ∧ (technical-staff,document,read,+,Sam))
(R5) ([95,96],Working-days,(Ann,pay-checks,read,+,Sam) UPON
      (Tom,pay-checks,write,+,Sam))
```

Fig. 1. An example of authorizations and derivation rules

each derivation rule has a bounding time interval and a periodicity, representing the instants at which it can be applied.

Definition 10. A *derivation rule* is a triple `([begin,end], P, A` $\langle$OP$\rangle$ $\mathcal{A}$`)`, where `begin` is a date expression, `end` is a date expression denoting an instant greater than or equal to the one denoted by `begin`, or `end` is the constant ∞, `P` is a periodic expression, `A` is an authorization, $\mathcal{A}$ is a boolean expression of authorizations, and $\langle$OP$\rangle$ is one of the following operators: WHENEVER, ASLONGAS, UPON.

The notion of validity, previously introduced for authorizations, naturally extends to a boolean expression of authorizations. Given a boolean expression of authorizations $\mathcal{A}$ and an instant t, we say that $\mathcal{A}$ is valid at time t if $\mathcal{A}$ evaluates "true" when each authorization in $\mathcal{A}$ is substituted by the boolean value "true" in case the authorization is valid at time t, by "false" otherwise.

Authorizations derived from derivation rules have as grantor the user who specified the rule.

We now give the intuitive semantics of the different kinds of derivation rules allowed by our model. The formal semantics will be given in Subsection 3.3. In the following we assume all authorizations are granted by the same user and we therefore do not consider the grantor of authorizations in the discussion. Clarifying examples will be given with respect to the authorizations and derivation rules illustrated in Figure 1.

—`([begin,end], P, A` WHENEVER $\mathcal{A}$`)`.
We can derive `A` for each instant in Π(`P`)$\cap$ $\{$`[`t_b`,`t_e`]`$\}$ for which $\mathcal{A}$ is valid.
For instance, rule R_4 in Figure 1 states that `summer-staff` can `read document` for every instant in `Summer-time`, from `1/1/1995`, in which both `staff` and `technical-staff` can `read document`.

—`([begin,end],P, A` ASLONGAS $\mathcal{A}$`)`.
Authorization `A` can be derived for each instant t in Π(`P`)$\cap$ $\{$`[`t_b`,`t_e`]`$\}$ such that $\mathcal{A}$

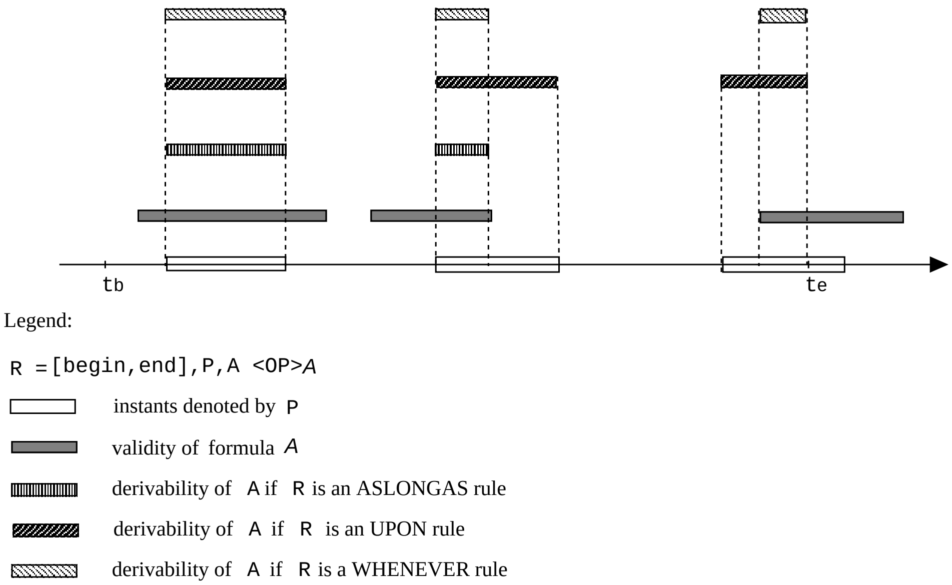

Fig. 2. Semantics of the different temporal operators

is valid for each time instant in $\Pi(\texttt{P})$ that is greater than or equal to $\texttt{t}_b$ and lesser than or equal to t. This implies that the ASLONGAS rule cannot be used anymore to derive authorization `A`, starting from the first instant $t \in \Pi(\texttt{P}) \cap \{[\texttt{t}_b,\texttt{t}_e]\}$ in which $\mathcal{A}$ is not valid.
For instance, rule $\texttt{R}_1$ in Figure 1 states that `temporary-staff` can `read document` each working day in `[1/1/96,12/31/98]` until the first working day in which `summer-staff` will be allowed for that.

—`([begin,end],P, A` UPON $\mathcal{A}$`)`.
We can derive `A` for each instant t in $\Pi(\texttt{P}) \cap \{[\texttt{t}_b,\texttt{t}_e]\}$ if there exists an instant $t' \in \Pi(\texttt{P})$ that is greater than and equal to $\texttt{t}_b$ or lesser than or equal to t such that $\mathcal{A}$ is valid at time t'.
For instance, rule $\texttt{R}_5$ in Figure 1 states that `Ann` can `read pay-checks` each working day starting from the first in `[1/1/95,12/31/96]` in which `Tom` can `write pay-checks`.

A graphical representation of the semantics of the different temporal operators which can appear in a derivation rule is given in Figure 2. Note that, according to the semantics of our temporal operators, the only times of interest in evaluating `R` are those in $\Pi(\texttt{P}) \cap \{[\texttt{t}_b,\texttt{t}_e]\}$.

Example 5. From the authorizations and rules in Figure 1 we can derive the following authorizations:

—`(summer-staff,document,read,+,Sam)` for each working day of the summers 1996 and 1997, from rule $\texttt{R}_4$ and authorizations $\texttt{A}_3$ and $\texttt{A}_5$.

—`(temporary-staff,document,read,+,Sam)` for each working day in `[1/1/96, 6/30/96]`, from rule $\texttt{R}_1$ using rule $\texttt{R}_4$ and authorizations $\texttt{A}_3$ and $\texttt{A}_5$. Note that the ending time of the derived authorization is determined by the starting time of the authorization derived for `summer-staff`, from rule R_4.

—(technical-staff,report,write,+,Sam) for each Monday and Friday from 5/22/95 to ∞, from rule R_2, using authorization A_1. Note that 5/22/95 is the first Monday in [1/1/95,∞] in which neither `manager` nor `staff` can `write` `guidelines`.

—(technical-staff,report,write,-,Sam) for each day in [1/1/95,9/30/95] and for each Saturday and Sunday from 10/1/95 to ∞, from rule R_3 using authorization A_2.

—(Ann,pay-checks,read,+,Sam) for each working day from 1/20/95 to 12/31/96 from rule R_5 and authorization A_4. 1/20/95 is the first pay-day in [1/1/95, 12/31/95].

A simple extension to the syntax of derivation rules allows the use of the special symbol '$*$' instead of a user, object, or access mode in the authorizations appearing in rules, with the meaning that any value in the corresponding domain can be used. We refer to rules containing the symbol '$*$' as *parametric derivation rules*. The wild card character can be used in the authorization on the left of the operator, i.e., in the authorization to be derived as well as in the authorizations appearing in the formula on the right hand side of the operator. Each parametric rule is exploded by the system into different rules, one for each value (or combination of values) to which the wild card character can be instantiated. In this derivation, the value substituted for an authorization element (i.e., user, object, or access mode) to the wild card character must be the same for all the authorizations on the left and right hand side of the same rule in which the wild card character is used instead of the element. Note that instantiation of occurrences of '$*$' in the authorization on the left hand side of the temporal operator is constrained by the authorizations that the grantor of the rule can specify. In particular, since we assume a user can specify only authorizations on the objects he owns, occurrences of the '$*$' symbol for the object in the authorization A on the left hand side of the operator will be instantiated to any object *owned by* g(A).

Example 6. Consider the rule:

(R) ([96,98], Working-days, (part-time-staff,*,read,+,Sam) WHENEVER
(staff,*,*,+,Sam) $\vee$ (temporary-staff,*,read,+,Sam))

Rule R states that `part-time-staff` can read, in an instant of a working day in the period 96-98, any object that either `temporary-staff` can read in that instant or on which `staff` has any privilege for the same instant.

Suppose there are three objects on which `Sam` can grant authorizations: `document`, `guidelines`, and `report`; and two access modes: `read` and `write`. The parametric rule is expanded by the system into the following rules:

(R_1) ([96,98], Working-days, (part-time-staff,document,read,+,Sam)
WHENEVER (staff,document,read,+,Sam) $\vee$
(temporary-staff,document,read,+,Sam))

(R_2) ([96,98], Working-days, (part-time-staff,document,read,+,Sam)
WHENEVER (staff,document,write,+,Sam) $\vee$

```
                 (temporary-staff,document,read,+,Sam))
(R3) ([96,98], Working-days, (part-time-staff,guidelines,read,+,Sam)
     WHENEVER (staff,guidelines,read,+,Sam) ∨
                 (temporary-staff,guidelines,read,+,Sam))
(R4) ([96,98], Working-days, (part-time-staff,guidelines,read,+,Sam)
     WHENEVER (staff,guidelines,write,+,Sam) ∨
                 (temporary-staff,guidelines,read,+,Sam))
(R5) ([96,98], Working-days, (part-time-staff,report,read,+,Sam)
     WHENEVER (staff,report,read,+,Sam) ∨
                 (temporary-staff,report,read,+,Sam))
(R6) ([96,98], Working-days, (part-time-staff,report,read,+,Sam)
     WHENEVER (staff,report,write,+,Sam) ∨
                 (temporary-staff,report,read,+,Sam))
```

3.2 Expressiveness and minimality of the temporal operators

Temporal operators which can appear in our derivation rules have been chosen to express three intuitive temporal relations among different authorizations. The derivation of new authorizations is obtained with WHENEVER by considering authorizations valid in the same time instants, and with ASLONGAS by considering the validity of authorizations in a whole span of time, while UPON allows the expression of *triggering* conditions. The relevance of these relations with respect to others that could be identified in a general temporal context, is motivated by the particular domain of access control.

Consider, for example, the derivation rules reported in Figure 1. In Rule R_4 the WHENEVER operator is used to give `summer-staff` the authorization to `read` a certain `document` at a certain time if, at the same time, both `staff` and `technical-staff` have the authorization to `read` that `document`. The set of instants at which this rule can be applied is restricted by the constraints associated with the rule to those included in each summer since the one in 1995. The same operator can be used to derive authorizations based on the absence of other authorizations at the same instants. This form of derivation can be useful in practice when two subjects must be authorized for an access over complementary intervals. Another example of use of WHENEVER is given by rule R_3: `technical-staff` is given a negative authorization to `write` a `report` for the instants at which it is not authorized to `read` the `guidelines`. The WHENEVER operator is sufficient to model many practical rules of an access control system.

Other protection requirements need the ability of the system to verify that a certain combination of authorizations has been continuously valid through a whole span of time. The existence of one instant in this span in which the combination of authorizations is not valid, must prevent the derivation of a new authorization. The ASLONGAS operator has been introduced for this purpose. In the example of Figure 1, rule R_1 derives an authorization for `temporary-staff` to `read` a `document` at a certain time t only if in each working day from `1996` up to t there was no authorization for `summer-staff` to `read` the `document`.

Finally, there are protection requirements that are based on some *triggering* conditions (expressed again as a combination of authorizations) to derive authorizations for future instants. In this case, it is sufficient that the condition is verified in a single instant to derive the authorization for all future instants according to the rule's constraints. The UPON operator, illustrated by rules R_2 and R_5 in the example of Figure 1, has been introduced to meet these requirements. By rule R_5 an authorization to user `Tom` for writing `pay-checks` in a working day within `1995` or `1996` is used as a triggering condition to issue an authorization to user `Ann` for reading `pay-checks` in any subsequent working day up to the last one in `1996`.

While the relations expressed by the temporal operators are intuitively very different from each other, a question can arise about the minimality of the set of operators that we have chosen to express these relations. Technically, when the interval associated with each rule is finite, any ASLONGAS or WHENEVER rule can be simulated by a set of UPON rules. For example, the same derivations obtained by `R = ([begin,end],P,A` WHENEVER $\mathcal{A}$) can be obtained by the set:

$([t_b,t_b],P,A$ UPON $\mathcal{A})$
$([t_b+1,t_b+1],P,A$ UPON $\mathcal{A})$
...
$([t_e,t_e],P,A$ UPON $\mathcal{A})$

where t_b and t_e are the instants corresponding to `begin` and `end`, respectively. The simulation is much more complex for ASLONGAS rules; technical details are not appropriate for this discussion, but it suffices to note that this simulation requires the introduction of twice the number of "auxiliary" authorizations as there are instants satisfying the constraints associated with the rule. It is easily seen that these simulations are highly impractical from a representational point of view, and that the algorithms we present in this paper would be extremely inefficient if operating on such a representation. Moreover, the simulation is not possible for rules with unbound intervals. We conclude that there are both expressiveness and efficiency arguments supporting the choice of the three temporal operators.

3.3 Formal semantics

In this section we give the formal semantics of periodic authorizations and derivation rules. We start by introducing the concept of a TAB.

Definition 11. A *Temporal Authorization Base* (TAB) is a set of periodic authorizations and derivation rules.

In the following, we use symbol A_i as a shorthand for the 5-tuple (s_i,o_i,m_i,pn_i,g_i), whereas A_i^- forces pn_i = '$-$', and A_i^+ forces pn_i = '$+$'.

The semantics of a TAB is given as a set of Datalog$^{\mathbf{not},\equiv Z,<Z}$ clauses. Datalog$^{\mathbf{not},\equiv Z,<Z}$ is the extension of Datalog with non-monotonic negation, periodicity, and gap-order constraints on the integers (see Appendix A). Programs corresponding to TABs are a very restricted class of Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs: the only predicate symbols are *valid*(), *valid*^{f}(), *once_valid*^{f}(), *once_not_valid*^{f}(), *denied*(), and CNSTR(). A set of non-temporal constants (A_1^+, A_1^-, A_1, $\mathcal{A}_1$,... s_1, o_1,m_1, ..., $+$, $-$,P_1, ...) is provided to denote positive and negative authorizations, authorizations regardless of their sign, boolean expressions of authorizations, users, objects, access modes, sign of authorizations, and periodic expressions.[8] Periodicity

[8]Note that when $\mathcal{A}$ appears as a predicate argument it denotes a constant that we associate with a boolean expression of authorizations. Similarly, `P` denotes a non-temporal constant that we

Table 2. Semantics of periodic authorizations and rules

$(\texttt{[begin, end]}, \texttt{P}, \texttt{A}^-)$:
$valid(t, \texttt{A}^-) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t)$

$(\texttt{[begin, end]}, \texttt{P}, \texttt{A}^+)$:
$valid(t, \texttt{A}^+) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t),\ \mathbf{not}(denied(t, \texttt{s}(\texttt{A}^+), \texttt{o}(\texttt{A}^+), \texttt{m}(\texttt{A}^+))$

$(\texttt{[begin, end]}, \texttt{P}, \texttt{A}^-\ \textsc{whenever}\ \mathcal{A})$:
$valid(t, \texttt{A}^-) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t),\ valid^f(t, \mathcal{A})$

$(\texttt{[begin, end]}, \texttt{P}, \texttt{A}^+\ \textsc{whenever}\ \mathcal{A})$:
$valid(t, \texttt{A}^+) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t),\ valid^f(t, \mathcal{A}),\ \mathbf{not}(denied(t, \texttt{s}(\texttt{A}^+), \texttt{o}(\texttt{A})^+, \texttt{m}(\texttt{A}^+))$

$(\texttt{[begin, end]}, \texttt{P}, \texttt{A}^-\ \textsc{aslongas}\ \mathcal{A})$:
$valid(t, \texttt{A}^-) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t),\ valid^f(t, \mathcal{A}),\ \mathbf{not}(once\_not\_valid^f(\texttt{t}_b, t, \texttt{P}, \mathcal{A}))$

$(\texttt{[begin, end]}, \texttt{P}, \texttt{A}^+\ \textsc{aslongas}\ \mathcal{A})$:
$valid(t, \texttt{A}^+) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t),\ valid^f(t, \mathcal{A}),\ \mathbf{not}(once\_not\_valid^f(\texttt{t}_b, t, \texttt{P}, \mathcal{A})),$
$\mathbf{not}(denied(t, \texttt{s}(\texttt{A}^+), \texttt{o}(\texttt{A}^+), \texttt{m}(\texttt{A}^+))$

$(\texttt{[begin,end]}, \texttt{P}, \texttt{A}^-\ \textsc{upon}\ \mathcal{A})$:
$valid(t, \texttt{A}^-) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t),\ once\_valid^f(\texttt{t}_b, t, \texttt{P}, \mathcal{A})$

$(\texttt{[begin, end]}, \texttt{P}, \texttt{A}^+\ \textsc{upon}\ \mathcal{A})$:
$valid(t, \texttt{A}^+) \leftarrow \texttt{t}_b \leq t \leq \texttt{t}_e,\ \textsc{cnstr}(\texttt{P}, t),\ once\_valid^f(\texttt{t}_b, t, \texttt{P}, \mathcal{A}),\ \mathbf{not}(denied(t, \texttt{s}(\texttt{A}^+), \texttt{o}(\texttt{A}^+), \texttt{m}(\texttt{A}^+))$

Auxiliary clauses:

$denied(t, s, o, m) \leftarrow valid(t, s, o, m, -, g)$

$\{\textsc{cnstr}(\texttt{P}, t) \leftarrow t \equiv_{periodicity(\texttt{P})} y\}$
$\forall y$ such that $t \equiv_{periodicity(\texttt{P})} y \Rightarrow t \in \Pi(\texttt{P})$

$\{once\_valid^f(t'', t, \texttt{P}, \mathcal{A}) \leftarrow t'' \leq t' \leq t,\ \textsc{cnstr}(\texttt{P}, t'),\ valid^f(t', \mathcal{A})\}$
$\forall$ distinct pair $(\texttt{P}, \mathcal{A})$ appearing in an UPON rule

$\{once\_not\_valid^f(t'', t, \texttt{P}, \mathcal{A}) \leftarrow t'' \leq t' < t,\ \textsc{cnstr}(\texttt{P}, t'), \mathbf{not}(valid^f(t', \mathcal{A}))\}$
$\forall$ distinct pair $(\texttt{P}, \mathcal{A})$ appearing in an ASLONGAS rule

$\{valid^f(t, \mathcal{A}) \leftarrow \mathbf{not}(valid(t, \texttt{A}_1)), \ldots, \mathbf{not}(valid(t, \texttt{A}_k)),\ valid(t, \texttt{A}_{k+1}), \ldots, valid(t, \texttt{A}_m)\}$
$\forall$ distinct conjunct $\texttt{C} = \bigwedge_{j=1}^{k} \neg\texttt{A}_j \wedge \bigwedge_{l=k+1}^{m} \texttt{A}_l$ in $\mathcal{A}$, $k \in [0, m]$, $m \in Z^+$,
and $\forall$ distinct $\mathcal{A}$ appearing in a derivation rule

and order constraints only involve temporal variables and do not use the function $+$.

Like [Falaschi et al. 1988; Gottlob et al. 1996] we consider non-ground interpretations of our programs, defined as sets of constrained atoms of the form (B, Ξ) where B is a predicate and Ξ is a temporal constraint. Each constrained interpretation has an equivalent (possibly infinite) Herbrand interpretation containing only ground atoms.

associate with a periodic expression.

Table 2 reports the clause/set of clauses in Datalog$^{\mathbf{not},\equiv Z,<Z}$ corresponding to each type of authorizations and rules allowed by our model. For brevity, we use the form $t'' \leq t' < t$ as a shortcut for the disjunction (using two clauses) of $t' = t''$ and $t'' < t' < t$. In defining the semantics we assume that the boolean formula $\mathcal{A}$ appearing on the right hand side of the temporal operator in a derivation rule is in disjunctive normal form. Moreover, we assume that in each conjunct authorizations preceded by $\neg$ appear before authorizations not preceded by $\neg$. More precisely, we assume $\mathcal{A} = \mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$, where, $\mathtt{C}_i = \bigwedge_{j=1}^{k} \neg\mathtt{A}_{ji} \wedge \bigwedge_{l=k+1}^{m} \mathtt{A}_{li}$, where $\mathtt{A}_{ji}$, $j = 1, \ldots, m$, $i = 1, \ldots, n$, are authorizations, $k \in [0, m]$, $m \in Z^+$.

Intuitively, predicate *valid*() represents the validity of authorizations at specific instants. The fact that $(valid(t, \mathtt{A}), \Xi)$ belongs to an interpretation means that A is valid according to that interpretation at all instants t satisfying Ξ. Predicate $valid^f$ is the analogous for boolean expressions of authorizations. The auxiliary predicates *denied*(), *once_not_valid*^{f}() and *once_valid*^{f}() are introduced to avoid quantification. $denied(t, \mathtt{s}, \mathtt{o}, \mathtt{m})$ is true in an interpretation if there is at least one negative authorization A such that $\mathtt{s(A)} = \mathtt{s}$, $\mathtt{o(A)} = \mathtt{o}$, $\mathtt{m(A)} = \mathtt{m}$, valid at instant t according to that interpretation. $once\_not\_valid^f(t'', t, \mathtt{P}, \mathcal{A})$ ($once\_valid^f(t'', t, \mathtt{P}, \mathcal{A})$, resp.) is true in an interpretation if there is at least one instant t', with $t'' \leq t' < t$ and $t' \in \Pi(\mathtt{P})$, at which $\mathcal{A}$ is not valid (valid, resp.) according to that interpretation.

We denote with P_{TAB} the Datalog$^{\mathbf{not},\equiv Z,<Z}$ program corresponding to a TAB. We consider *stable model semantics* of logic programs with negation [Gelfond and Lifschitz 1988] to identify the models[9] of P_{TAB}. The notion of constrained interpretation presented above naturally extends to constrained (non-ground) stable models.

We are now ready to formally introduce the notion of valid authorization.

Definition 12. Let M be a model of P_{TAB}. An authorization A is *valid at time* $\bar{t}$ *with respect to* M if there exists $(valid(t, \mathtt{A}), \Xi)$ in M with $\bar{t}$ satisfying Ξ. If P_{TAB} has a unique ground model and M is one of its non-ground representations, then we simply say that A is valid at time $\bar{t}$.

4. A UNIQUE SET OF VALID AUTHORIZATIONS

The presence of negation in our derivation rules introduces the problem of generating a unique set of valid authorizations from a given set of periodic authorizations and rules. There are situations in which different sets of authorizations are generated, depending on the rule evaluation order.

Example 7. Consider a TAB consisting of the following rules:

($\mathtt{R}_1$) `([97,98],Working-days,(manager,report,read,+,Sam)`
    WHENEVER $\neg$`(technical-staff,report,write,+,Sam))`
($\mathtt{R}_2$) `([97,98],Working-days,(technical-staff,report,write,+,Sam)`
    WHENEVER $\neg$`(manager,report,read,+,Sam))`

By evaluating first $\mathtt{R}_1$ and then $\mathtt{R}_2$ we derive authorization `(manager, report, read, +, Sam)` for each working day of 1997 and 1998 (from $\mathtt{R}_1$), and we cannot derive any authorization from $\mathtt{R}_2$. If we evaluate first $\mathtt{R}_2$, we derive `(technical-staff,`

[9]Due to the properties of the resulting program, in this case stable models are identical to *well-founded* models [Gelder et al. 1991].

report, write, +, Sam) for each working day of 1997 and 1998, and we cannot derive any authorization from $\mathtt{R}_1$.

From the point of view of the semantics, the property of always having a unique set of valid authorizations is guaranteed only if all the models of the program corresponding to a TAB identify the same set of valid authorizations at any instant (or equivalently, there exists a *unique* ground stable model of P_{TAB}).

In the remainder of this section we formally define restrictions on sets of rules that guarantee a unique ground model for P_{TAB}, and present an algorithm for checking the satisfaction of these restrictions. Intuitively, the restrictions require the existence of a unique evaluation order reflecting the intuitive semantics of derivation rules.

4.1 Restrictions on rules

In the following, we use the term *past operator* (PASTOP) to refer to ASLONGAS and UPON.

Dependencies among authorizations in TAB are identified by the binary relationships *affects* ($\hookrightarrow$) and *strictly affects* ($\overset{+}{\hookrightarrow}$) defined as follows.

—if there is a rule ([begin,end], P, $\mathtt{A}_m\langle\text{OP}\rangle\ \mathcal{A}$) in TAB, where OP is an arbitrary operator, then for each $t \in \{[\mathtt{t}_b,\mathtt{t}_e]\}\cap\Pi(\mathtt{P})$ and for each $\mathtt{A}_k$ appearing in $\mathcal{A}$ we say that $\mathtt{A}_k$ at time t *affects* $\mathtt{A}_m$ at time t (written $\mathtt{A}_k[t] \hookrightarrow \mathtt{A}_m[t]$). If authorization $\mathtt{A}_k$ is preceded by $\neg$ in $\mathcal{A}$ we say that $\mathtt{A}_k$ at time t *strictly affects* $\mathtt{A}_m$ at time t (written $\mathtt{A}_k[t] \overset{+}{\hookrightarrow} \mathtt{A}_m[t]$);

—if there is a rule ([begin,end], P, $\mathtt{A}_m$ $\langle\text{PASTOP}\rangle$ $\mathcal{A}$) in TAB, then for each $t,t' \in \{[\mathtt{t}_b,\mathtt{t}_e]\}\cap\Pi(\mathtt{P})$, $t < t'$, and for each authorization $\mathtt{A}_k$ appearing in $\mathcal{A}$ we say that $\mathtt{A}_k$ at time t *affects* $\mathtt{A}_m$ at time t' (written $\mathtt{A}_k[t] \hookrightarrow \mathtt{A}_m[t']$). If authorization $\mathtt{A}_k$ is preceded by $\neg$ in $\mathcal{A}$ or PASTOP=ASLONGAS, then we say that $\mathtt{A}_k$ at time t *strictly affects* $\mathtt{A}_m$ at time t' (written $\mathtt{A}_k[t] \overset{+}{\hookrightarrow} \mathtt{A}_m[t']$).

Note that $\mathtt{A}_k[t] \overset{+}{\hookrightarrow} \mathtt{A}_m[t']$ implies $\mathtt{A}_k[t] \hookrightarrow \mathtt{A}_m[t']$, for any t,t'. Based on these relationships, we can define the more complex notion of *priority* among periodic authorizations. Intuitively, an authorization $\mathtt{A}_n$ at time t has higher priority than authorization $\mathtt{A}_m$ at time t', if the validity of $\mathtt{A}_m$ at time t' can be evaluated only after evaluating the validity of $\mathtt{A}_n$ at time t.

Definition 13. Authorization $\mathtt{A}_n$ at time t has *higher priority* than authorization $\mathtt{A}_m$ at time t' (written $\mathtt{A}_n[t] > \mathtt{A}_m[t']$) if one of the following conditions holds:

—a sequence $\mathtt{A}_n[t]$=$\mathtt{A}_1[t_1]$, ..., $\mathtt{A}_{k-1}[t_{k-1}]$, $\mathtt{A}_k[t_k]$=$\mathtt{A}_m[t']$ exists such that each element in the sequence affects the successor and there exists one that strictly affects it.

—two sequences $\mathtt{A}_n[t]$=$\mathtt{A}_1[t]$, ..., $\mathtt{A}_l^-[t'']$ and $\mathtt{A}_{l+1}^+[t'']$, ..., $\mathtt{A}_k[t']$=$\mathtt{A}_m[t']$ exist such that each element affects the successor in the sequence, and $\mathtt{s}(\mathtt{A}_l^-)$=$\mathtt{s}(\mathtt{A}_{l+1}^+)$, $\mathtt{o}(\mathtt{A}_l^-)$= =$\mathtt{o}(\mathtt{A}_{l+1}^+)$, and $\mathtt{m}(\mathtt{A}_l^-)$=$\mathtt{m}(\mathtt{A}_{l+1}^+)$;

—an authorization $\mathtt{A}_l$ and an instant t'' exist such that $\mathtt{A}_n[t] > \mathtt{A}_l[t'']$ and $\mathtt{A}_l[t''] > \mathtt{A}_m[t']$.

The second condition in the above definition implies that negative authorizations have higher priority than their positive counterparts at the same instant.

We are now ready to formally characterize *critical sets* of derivation rules, that is, sets of rules which could lead to the derivation of different sets of authorizations depending on the evaluation order.

Definition 14. A TAB contains a *critical set* of rules if and only if there exist an authorization $\mathtt{A}_m$ in TAB and an instant t such that $\mathtt{A}_m$ has priority over itself at time t (i.e., $\mathtt{A}_m[t] > \mathtt{A}_m[t]$).

Example 8. It is easy to verify that the rules of Example 7 form a critical set. Indeed, take t as an arbitrary instant within a working day of 1997. By the first condition in Definition 13 and rule $\mathtt{R}_1$ we have `(technical-staff,report,write,+, Sam)` $[t] >$ `(manager,report,read,+,Sam)`$[t]$. Similarly, by rule $\mathtt{R}_2$ we have `(manager, report, read, +,Sam)`$[t] >$ `(technical-staff, report, write, +, Sam)`$[t]$. Applying the third condition in Definition 13 (transitivity) we have `(technical-staff, report, write, +, Sam)`$[t] >$ `(technical-staff, report, write, +, Sam)`$[t]$. Hence, $\mathtt{R}_1$ and $\mathtt{R}_2$ form a critical set.

The CSD (Critical Set Detection) algorithm, described in the next subsection, recognizes and rejects a TAB containing a critical set.

4.2 The CSD algorithm

Before illustrating the CSD algorithm we need to introduce some notions.

Given a TAB, we introduce its *constraint version*, denoted as TAB_{CNS}, as the set of pairs of the form $\langle x, \Xi \rangle$, where x is either an authorization or rule in TAB and Ξ is the temporal constraint, corresponding to the values of `[begin,end]` and `P` associated with x in TAB. If $\mathtt{A}_m$ is specified more than once in TAB with different temporal constraints, the constraint Ξ associated with $\mathtt{A}_m$ in TAB_{CNS} will be the disjunction of these constraints. Derivation rules are transformed in an analogous way. In the following, given an authorization $\mathtt{A}_m$ in TAB, Ξ_m denotes the temporal constraint associated with $\mathtt{A}_m$ in TAB_{CNS}. Analogously, $\Xi_{\mathtt{R}}$ denotes the temporal constraint associated with rule `R` in TAB_{CNS}.

The algorithm for detecting critical sets is illustrated in Figure 3. It receives as input TAB_{CNS}, i.e., the constraint version of TAB. It returns FALSE if a critical set exists. Otherwise, it returns a sequence of levels $\langle L_1, \ldots, L_k \rangle$ representing a *partition* of the set of pairs $\langle \mathtt{A}, t \rangle$ for each authorization `A` appearing (either explicitly or in a derivation rule) in TAB_{CNS} and $\mathtt{t}_{min} \leq t \leq$ `max-time`, where $\mathtt{t}_{min}$ is the minimum constant appearing in a gap-order constraint in TAB_{CNS}, and `max-time` is an upper bound for the instant after which the validity of any authorization becomes periodic. `max-time` is determined as $\overline{\mathtt{t}}_{max} + \bar{k} \cdot \mathtt{P}_{max}$, where $\overline{\mathtt{t}}_{max}$ is the maximum *finite* constant appearing in a gap-order constraint in TAB_{CNS}, $\mathtt{P}_{max}$ is the least common multiple (lcm) of all the periodicities appearing in TAB_{CNS}, and $\bar{k}$ is the maximum number of ASLONGAS and UPON rules in TAB plus one (see Lemma 3 in Appendix B). The reason for introducing `max-time` in the CSD algorithm is to ensure the finiteness of the instants to be considered in the partition, and hence, the termination of the algorithm. The motivation behind how `max-time` is determined is that the levels produced by the CSD algorithm are used to determine the

ALGORITHM 1. *Critical Set Detection (CSD) Algorithm*

INPUT: TAB_{CNS}.
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 $\langle \mathtt{A}, t\rangle$ such that A appears in TAB and $\mathtt{t}_{min} \leq t \leq \mathtt{max\text{-}time}$. Each set L_i is called level i and $L_i = \{\langle \mathtt{A}_1, \Xi_{1,i}\rangle, \ldots, \langle \mathtt{A}_r, \Xi_{r,i}\rangle\}$ where $\Xi_{j,i}$ is the constraint associated with $\mathtt{A}_j$ at level i
METHOD:
1.Substitute `max-time` to all occurrences of ∞ in the gap-order constraints associated with authorizations or rules in TAB_{CNS}
2.Determine $\mathtt{max\text{-}level} := \mathtt{n\text{-}auth} \cdot \mathtt{max\text{-}time}$, where `n-auth` is the number of authorizations appearing in TAB_{CNS}
3. $\mathtt{top\text{-}level} := 1$
4. Insert $\langle \mathtt{A}_i, \{(\{\text{true}\}, \{\mathtt{t}_{min} \leq t \leq \mathtt{max\text{-}time}\})\}\rangle$ at level L_1 for each $\mathtt{A}_i$ in TAB_{CNS}
5. **Repeat**
 5.1. **For** $l := \mathtt{top\text{-}level}, \ldots, 1$:
   Partition the set of pairs $\langle \mathtt{A}_i, \Xi_{i,l}\rangle \in L_l$ such that $\mathtt{A}_i$ is a negative authorization in $\{S_1, \ldots, S_u\}$, where each S_i contains all authorizations with the same subject s_i, object o_i, and access mode m_i
   For each S_i:
    $\Xi_i := \bigcup\{\Xi_{n,l} \mid \langle \mathtt{A}_n, \Xi_{n,l}\rangle \in S_i\}$
    For $h := l, \ldots, 1$:
     For each $\langle \mathtt{A}_m, \Xi_{m,h}\rangle \in L_h$ such that $\Xi_{m,h} \wedge^* \Xi_i \neq \emptyset$, $\mathtt{s(A}_m\mathtt{)=s}_i$, $\mathtt{o(A}_m\mathtt{)=o}_i$, $\mathtt{m(A}_m\mathtt{)=m}_i$, $\mathtt{pn(A}_m\mathtt{)}$ = '+':
      $\Xi := \Xi_{m,h} \wedge^* \Xi_i$
      delete $\langle \mathtt{A}_m, \Xi\rangle$ from L_h
      If $l = \mathtt{top\text{-}level}$ **then** $\mathtt{top\text{-}level} := l+1$, $L_{\mathtt{top\text{-}level}} := \emptyset$
      add $\langle \mathtt{A}_m, \Xi\rangle$ to L_{l+1}
 5.2. **For** each $\langle \mathtt{A}_m$ WHENEVER $\mathcal{A}, \Xi_{\mathtt{R}}\rangle \in \text{TAB}_{CNS}$:
   For each authorization $\mathtt{A}_n$ appearing in $\mathcal{A}$:
    For $l := \mathtt{top\text{-}level}, \ldots, 1$:
     If $\langle \mathtt{A}_n, \Xi_{n,l}\rangle \in L_l$ with $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l} \neq \emptyset$
      then $\Xi := \Xi_{\mathtt{R}} \wedge^* \Xi_{n,l}$
       If $\mathtt{A}_n$ appears in $\mathcal{A}$ preceded by $\neg$
        then move($\mathtt{A}_m$,Ξ,$l+1$)
       else move($\mathtt{A}_m$,Ξ,l)
 5.3. **For** each $\langle \mathtt{A}_m$ $\langle$PASTOP$\rangle\mathcal{A}, \Xi_{\mathtt{R}}\rangle \in \text{TAB}_{CNS}$:
   Let $\Xi_{\mathtt{R}}$ be $\{(PC_1, GC), \ldots (PC_k, GC)\}$
   Let $GC = \{\mathtt{t}_b \leq t \leq \mathtt{t}_e\}$
   For each authorization $\mathtt{A}_n$ appearing in $\mathcal{A}$:
    For $l := \mathtt{top\text{-}level}, \ldots, 1$:
     If $\langle \mathtt{A}_n, \Xi_{n,l}\rangle \in L_l$ with $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l} \neq \emptyset$
      then Let $\mathtt{t}_l$ be the first instant satisfying $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l}$
       If $\mathtt{A}_n$ appears in $\mathcal{A}$ preceded by $\neg$
        then
         move($\mathtt{A}_m$, $\{(PC_1, \{\mathtt{t}_l \leq t \leq \mathtt{t}_e\}), \ldots (PC_k, \{\mathtt{t}_l \leq t \leq \mathtt{t}_e\})\}$,$l+1$)
       elsif OP = UPON
         then move($\mathtt{A}_m$,$\{(PC_1, \{\mathtt{t}_l \leq t \leq \mathtt{t}_e\}), \ldots, (PC_k, \{\mathtt{t}_l \leq t \leq \mathtt{t}_e\})\}$,$l$)
        elsif OP = ASLONGAS
          then
           move($\mathtt{A}_m$,$\{(PC_1, \{\mathtt{t}_l < t \leq \mathtt{t}_e\}), \ldots, (PC_k, \{\mathtt{t}_l < t \leq \mathtt{t}_e\})\}$, $l+1$),
           move($\mathtt{A}_m$,$\{(\{\text{true}\}, \{\mathtt{t}_l = t\})\}$,$l$)
 Until there are no changes to any level or $\mathtt{top\text{-}level} > \mathtt{max\text{-}level}$
6. **Return** FALSE if $\mathtt{top\text{-}level} > \mathtt{max\text{-}level}$, the sequence $\langle L_1, \ldots, L_{\mathtt{top\text{-}level}}\rangle$, otherwise

Procedure move($\mathtt{A}_h, \Xi, lev$)
1. **If** $lev > \mathtt{top\text{-}level}$ **then** $\mathtt{top\text{-}level} := lev$, $L_{lev} := \emptyset$
2. **For** $i := lev - 1 \ldots 1$:
  If $\langle \mathtt{A}_h, \Xi_{h,i}\rangle \in L_i$ with $\Xi_{h,i} \wedge^* \Xi \neq \emptyset$
   then $\Xi' := \Xi_{h,i} \wedge^* \Xi$, *delete* $\langle \mathtt{A}_h, \Xi'\rangle$ from L_i,
    add $\langle \mathtt{A}_h, \Xi'\rangle$ to L_{lev}

Fig. 3. The CSD algorithm

set of authorizations valid according to the TAB. Intuitively, for instants greater than `max-time` the validity of each authorization can be evaluated considering the corresponding instant in the time period before `max-time`. Hence, derivation of authorizations can be limited to instants smaller than or equal to `max-time` and then extended to ∞.

Each level L_i is a set of pairs $\langle \mathtt{A}_j, \Xi_{j,i}\rangle$, where $\Xi_{j,i}$ denotes the constraint associated with $\mathtt{A}_j$ at level L_i. In the following we say that authorization $\mathtt{A}_j$ appears at level i for instant t if t satisfies $\Xi_{j,i}$. The intuitive semantics of the levels is that 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 for a different set of instants). In other words, let $\mathtt{A}_i$ and $\mathtt{A}_j$ be two authorizations and t_i and t_j be two time instants. If the level of $\mathtt{A}_i$ at instant t_i is lower than the level of $\mathtt{A}_j$ at instant t_j then the validity of $\mathtt{A}_i$ at time t_i must be evaluated before the validity of $\mathtt{A}_j$ at time t_j. (Note that since levels represent a partition of the pairs $\langle \mathtt{A}, t\rangle$, each authorization belongs to exactly one level for each time instant considered). Levels define therefore an order, from the lowest to the highest level, which must be followed when determining the set of temporal authorizations valid according to the given TAB.

The algorithm works as follows. In step 1, `max-time` is substituted to all occurrences of ∞ in the gap-order constraints of authorizations and rules appearing in TAB_{CNS}. In step 2, the maximum number of levels `max-levels` to be generated is determined as the number of authorizations in TAB_{CNS} multiplied by `max-time`. Intuitively, `max-level` is the number of different pairs $\langle \mathtt{A}, t\rangle$ to be partitioned. In step 3, variable `top-level`, representing the number of levels actually generated, is initialized to 1. Then, in step 4, all the authorizations appearing in TAB_{CNS} are put at level 1 for all time instants between $\mathtt{t}_{min}$ and `max-time`. In step 5, the algorithm considers the dependencies caused by negative authorizations (substep 5.1) and by derivation rules (substeps 5.2 and 5.3) and possibly moves authorizations to higher levels. Two operations allow changes to levels: *add* and *delete*. They insert in (delete from) levels, pairs of the form $\langle \mathtt{A}_m, \Xi\rangle$. Deleting pair $\langle \mathtt{A}_m, \Xi\rangle$ from level h means updating $\Xi_{m,h}$ to be $\Xi_{m,h} \wedge^* \neg^* \Xi$. Similarly, adding pair $\langle \mathtt{A}_m, \Xi\rangle$ to level k means updating $\Xi_{m,k}$ to include all the pairs (PC, GC) present in Ξ. If $\mathtt{A}_m$ does not appear in L_k before the execution of the add operation, the result of the operation is the insertion of pair $\langle \mathtt{A}_m, \Xi\rangle$ in L_k. The algorithm uses procedure 'move()' to move authorizations to higher levels. move($\mathtt{A}_h, \Xi, lev$) checks all levels i below *lev* for which $\Xi' = \Xi_{h,i} \wedge^* \Xi \neq \emptyset$. Hence, it deletes $\langle \mathtt{A}_h, \Xi'\rangle$ from L_i and adds it to L_{lev}. Intuitively, move($\mathtt{A}_h, \Xi, lev$) moves authorization $\mathtt{A}_h$ to level *lev* for each instant satisfying Ξ for which $\mathtt{A}_H$ appears in levels lower than *lev*. Step 5 is repeated until no changes to the levels are necessary (i.e., all the priorities are satisfied), or the number of levels becomes greater than `max-level`. In the first case, the levels generated are returned. In the latter case, a FALSE is returned indicating the presence of a critical set.

Example 9. Consider a TAB containing authorizations $\mathtt{A}_1$, $\mathtt{A}_2$, $\mathtt{A}_3$ and rules $\mathtt{R}_1$, $\mathtt{R}_2$, and $\mathtt{R}_3$ in Figure 1. For brevity, here and in the following examples we assume *Days* as our basic calendar and Sunday `1/1/95` as our tick 1. Moreover, we do not use the normal form to represent the constraint Ξ, but the more compact form:

Level 1:
$\langle$(manager, guidelines, write, $+$, Sam), $\{(\{\text{true}\},\{1/1/95 \leq t \leq 1/21/99\})\}\rangle$
$\langle$(technical-staff,guidelines,read,+,Sam),$\{(\{\text{true}\},\{1/1/95 \leq t \leq 1/21/99\})\}\rangle$
$\langle$(staff, document, read, $+$, Sam), $\{(\{\text{true}\},\{1/1/95 \leq t \leq 1/21/99\})\}\rangle$
$\langle$(summer-staff,document,read,+,Sam),$\{(\{\text{true}\},\{1/1/95 \leq t \leq 1/21/99\})\}\rangle$
$\langle$(staff, guidelines, write, $+$, Sam), $\{(\{\text{true}\},\{1/1/95 \leq t \leq 1/21/99\})\}\rangle$
$\langle$(temporary-staff,document,read,+,Sam), $\{(\{\text{true}\},\{1/1/95 \leq t \leq 12/31/95\}),$
$\{(\{t \equiv_7 y\ y = 0, 6\}, \{1/1/96 \leq t \leq 12/31/98\}), (\{\text{true}\},\{1/1/99 \leq t \leq 1/21/99\})\}\rangle$

Level 2:
$\langle$(temporary-staff,document,read,+,Sam), $\{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\},$
$\{1/1/96 \leq t \leq 12/31/98\})\}\rangle$
$\langle$(technical-staff,report,write,-,Sam),$\{(\{\text{true}\}, \{1/1/95 \leq t \leq 1/21/99\})\}\rangle$

Level 3:
$\langle$(technical-staff,report,write,+,Sam),$\{(\{\text{true}\}, \{1/1/95 \leq t \leq 1/21/99\})\}\rangle$

Fig. 4. An example of levels returned by the CSD algorithm

$(\{t \equiv_k (y + c) \forall y = l_1 \ldots l_p\}, c_1 \leq t \leq c_2)$. TAB$_{CNS}$ is as follows:

$\{\langle$(manager,guidelines,write,+,Sam), $\{(\{\text{true}\},\{1/1/95 \leq t \leq 5/20/95\})\}\rangle$, $\langle$(technical-staff,guidelines,read,+,Sam) $\{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\},$ $\{10/1/95 \leq t\})\}\rangle$, $\langle$(staff,document,read,+,Sam) $\{(\{t \equiv_7 (y+1) \forall y = 0, \ldots,$ $4\}, \{1/1/95 \leq t \leq 12/31/97\})\}\rangle$, $\langle$(temporary-staff,document,read,+,Sam) ASLONGAS $\neg$(summer-staff,document,read,+,Sam), $\{(\{t \equiv_7 (y+1)$ $\forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/98\})\}\rangle$, $\langle$(technical-staff,report,write, +,Sam) UPON $\neg$(manager,guidelines,write,+,Sam)$\wedge\neg$(staff,guidelines, write,+,Sam) $\{(\{t \equiv_7 (y+1)\ y = 0, 4\}, \{1/1/95 \leq t\})\}\rangle$, $\langle$(technical-staff, report,write,-,Sam) WHENEVER $\neg$(technical-staff,guidelines,read, +,Sam) $\{(\{\text{true}\}, \{1/1/95 \leq t\})\}\rangle$.

We then have: $\overline{\mathtt{t}}_{max}$ = 12/31/98, $\mathtt{t}_{min}$ = 1/1/95, $\overline{k} = 3$, $\mathtt{P}_{max} = 7$ (i.e, the periodicity of *Weeks*), and max-time = 1/21/99. TAB$_{CNS}$ is modified by substituting 1/21/99 to all occurrences of ∞. All the authorizations appearing in TAB$_{CNS}$ are initially inserted at level 1 with constraints $\{(\{\text{true}\},\{1/1/95 \leq t \leq 1/21/99\})\}$.

The algorithm then cycles moving authorizations to higher levels as follows.

1st iteration:
Since (technical-staff,report,write,-,Sam) is at level 1:
move $\langle$(technical-staff,report,write,+,Sam), $\{(\{\text{true}\},\{1/1/95 \leq t \leq$ $\leq 1/21/99\})\}\rangle$ to level 2.
Considering rule $\mathtt{R}_1$: move $\langle$(temporary-staff,document,read,+,Sam), $\{(\{t \equiv_7$ $(y+1) \forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/98\})\}\rangle$ to level 2.
Considering rule $\mathtt{R}_3$: move $\langle$(technical-staff,report,write,-,Sam), $\{(\{\text{true}\},$ $\{1/1/95 \leq t \leq 1/21/99\})\}\rangle$ to level 2.

2nd iteration:
Since (technical-staff,report,write,-,Sam) is at level 2:
move $\langle$(technical-staff,report,write,+,Sam), $\{(\{\text{true}\}, \{1/1/95 \leq t \leq$ $\leq 1/21/99\})\}\rangle$ to level 3.

3rd iteration:
All dependencies are satisfied. No further changes to the levels are necessary and the algorithm terminates returning the levels illustrated in Figure 4.

Note that several optimizations are possible on the algorithm in Figure 3, which are not reported here for the sake of simplicity. For instance, the 'move()' procedure could keep track of the instants moved and stop if $\mathtt{A}_m$ has been moved for all instants satisfying Ξ, without the need of checking all levels down to the first one.

The number of levels produced by the CSD algorithm is potentially very high. However, note that the maximum number of levels can be reached only if either *i)* the TAB contains a critical set, or *ii)* the TAB does not contain any critical set but the priorities are such that the maximum number of levels needs to be produced. Case *i)*, representing an anomaly in the authorization specifications, should be very unlikely to occur. As for case *ii)*, if the maximum number of levels is reached, each level must contain exactly one authorization with an associated constraint identifying a single instant. In terms of the authorization specifications this can happen only if there exists a cycle involving ASLONGAS rules. More precisely, two ASLONGAS rules spanning to ∞ must exist such that each of them contains, in the formula on the right of the operator, the authorization that appears on the left of the operator in the other rule. For instance, it is easy to see that the pair of rules: $\mathtt{R}_1 = (\mathtt{[1,}\infty\mathtt{]},\ \bot,\ \mathtt{A}_1\ \text{ASLONGAS}\ \mathtt{A}_2)$ and $\mathtt{R}_2 = (\mathtt{[1,}\infty\mathtt{]},\ \bot,\ \mathtt{A}_2\ \text{ASLONGAS}\ \mathtt{A}_1)$ would require the computation of all levels up to `max-level`. A possible solution to this problem is to reject the insertion of any rule causing a cycle in the TAB, regardless of the operators involved in the rules and authorizations causing the cycle. This can be easily accomplished by changing the definition of critical set considering all the *affects* relationships as *strict affects* relationships. In this way the worst case performance of the CSD algorithm would be greatly enhanced since the maximum number of levels that can be produced becomes only dependent on the number of authorizations and rules in TAB. However, we prefer to adopt a less restrictive approach, refusing the insertion of a derivation rule only if its insertion into the existing TAB leads to a corresponding logic program that is neither locally stratified [Gelder et al. 1991], nor equivalent to a locally stratified one, and, hence, with a non clear choice for the rules evaluation order. We accept the insertion of all other rules, even those identified above, which could cause the generation of a high number of levels. We believe that real world specifications will generally require the computation of a very small number of levels.

4.3 Formal properties

The following theorems state that the partition produced by the CSD algorithm obeys the priority relationships existing in TAB. These theorems extend the results reported in [Bertino et al. 1996a] to the consideration of periodic authorizations and derivation rules. Proofs are reported in Appendix B.

THEOREM 1. *Let* $\mathtt{A}_n$ *and* $\mathtt{A}_m$ *be two authorizations appearing in* TAB *and* t, t' *be two time instants between* $\mathtt{t}_{min}$ *and* `max-time`. *If* $\mathtt{A}_n[t] \hookrightarrow \mathtt{A}_m[t']$, *then either the* CSD *algorithm returns* FALSE *or, at the end of the execution, authorization* $\mathtt{A}_m$ *for instant* t' *appears at a level higher than or equal to that of authorization* $\mathtt{A}_n$ *for instant* t. *If* $\mathtt{A}_n[t] \stackrel{+}{\hookrightarrow} \mathtt{A}_m[t']$, *then the level is necessarily higher.*

THEOREM 2. *Let* $\mathtt{A}_n$ *and* $\mathtt{A}_m$ *be two authorizations appearing in* TAB *with same subject, object and access mode but with different sign. Then, either the* CSD *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* $\mathtt{t}_{min}$ *and* `max-time`.

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

THEOREM 3. *Given a* TAB, *i) the* CSD *algorithm terminates and ii) it returns* FALSE *iff the* TAB *contains a critical set.*

The authorization state of the system is unique if and only if P_{TAB} has a unique ground model. The uniqueness of the model in absence of critical sets is guaranteed by the following theorem.

THEOREM 4. *Given a* TAB *with no critical sets, the corresponding logic program* P_{TAB} *has a unique ground model.*

Note that more than one finite constrained non-ground model of P_{TAB} equivalent to the unique ground model can exist, since the same set of instants can be represented by different constraints.

5. ACCESS CONTROL

Determining whether an access request from user `s` to exercise access mode `m` on object `o` at time t can be authorized requires the system to verify whether an authorization `(s,o,m,+,g)` valid at time t exists. Two different strategies can be used to enforce access control: *run-time derivation* and *materialization*. Under the first approach, the system, starting from the access request, verifies whether the access can be authorized by evaluating the derivation rules in TAB according to a top-down strategy. This approach has the advantage that limited actions are required upon modification of the TAB. In particular, addition and removal of authorizations as well as removal of rules can be executed without further actions. Insertion of new rules requires the verification that no critical sets are introduced. However, access control may require the evaluation of several rules (to determine the validity of authorizations and to compute the derived authorizations).

By contrast, under the materialization approach the system permanently maintains all the valid authorizations derivable from a TAB. Access control therefore becomes very efficient: there is no difference in costs between explicit and derived authorizations. As a drawback, the materialization has to be properly updated every time the TAB is modified or whenever an object is created or deleted by a user who specified rules parametric with respect to the object. (Note that the creation/deletion of an object subject to a parametric rule can be seen as the insertion/removal of the rule instantiations corresponding to the object.) In the implementation of the proposed model we have adopted the materialization approach. The main reason behind this choice is that in real systems access requests are generally considerably more frequent than requests modifying authorizations and/or rules. As for creation/deletion of objects subject to parametric rules, we believe that those rules, though possible, will seldom be used in practice. As a matter of fact, while rules parametric with respect to the access mode (allowing the owner to grant any access on an object) or to the subject (allowing the owner to grant an access to everybody or to grant an access if somebody can exercise it) can be imagined to be common in reflecting real world situations, rules parametric with respect to the object, by which a user can grant access to all objects he owns (or will own) certainly have a much stricter applicability.

In the remainder of this section we illustrate how to compute, given a TAB, the corresponding set of valid authorizations. We start with the following definition originally from [Bertino et al. 1996a].

Definition 15. The *Temporal Authorization Base Extent* (TAB_{EXT}) of TAB is the set of valid authorizations derived from TAB.

Authorizations are maintained in TAB_{EXT} using a compact representation similar to that of TAB_{CNS}. Each $\mathtt{A}_k$ is associated with a temporal constraint Ω_k. $\langle \mathtt{A}_k, \Omega_k \rangle$ is in TAB_{EXT} if authorization $\mathtt{A}_k$ is valid at each instant t satisfying Ω_k.

Given two temporal constraint Ξ and Ξ', we say that Ξ is *shift-equivalent* to Ξ' (written $\Xi \stackrel{\rightarrow}{=}^{*} \Xi'$), if the instants satisfying Ξ are a transposition of the instants satisfying Ξ' on the time axis. Formally, $\Xi \stackrel{\rightarrow}{=}^{*} \Xi'$ if $\exists \bar{t} \in \mathbb{N}$ such that $t + \bar{t}$ satisfies Ξ' whenever t satisfies Ξ. For instance, $\Xi = \{(\{\mathtt{1/1/97} \leq t \leq \mathtt{1/31/97}\}, \{\text{true}\})\} \stackrel{\rightarrow}{=} \{(\{\mathtt{1/1/98} \leq t \leq \mathtt{1/31/98}\}, \{\text{true}\})\} = \Xi'$, since for each instant t, $t + 365$ satisfies Ξ' whenever t satisfies Ξ.

Figure 5 presents an algorithm for computing TAB_{EXT}. The algorithm is based on the model computation for (locally) stratified $\text{Datalog}^{\mathbf{not}, \equiv Z, <Z}$ programs given in Appendix A. This computation is represented in Algorithm 2 by an iteration of the inner **repeat-until** cycle. The termination of each iteration is guaranteed by using a finite constant as an upper bound in constraints and computing TAB_{EXT} only up to that value. The periodicity of our rules and their semantics guarantee that the derivation of authorizations can be performed only for instants smaller than or equal to the finite constant `max-time` and then extended to ∞ (step 3 of Algorithm 2), where $\mathtt{max\text{-}time} = \bar{\mathtt{t}}_{max} + \bar{k} \cdot \mathtt{P}_{max}$ is the constant introduced in the presentation of the CSD Algorithm. However, there are many cases in which it is not necessary to compute the TAB_{EXT} up to `max-time`. This is the reason for the outer **repeat-until** cycle. In particular, the algorithm considers two contiguous time intervals after $\bar{\mathtt{t}}_{max}$ (the the maximum *finite* constant appearing in a gap-order constraint in TAB_{CNS}) of length equal to the common periodicity in TAB ($\mathtt{P}_{max}$) and checks whether the constraints associated with the derived authorizations and restricted to these intervals are shift-equivalent (step 2.3). If not, and if the considered intervals do not exceed `max-time`, it proceeds with another iteration of step 2, generating a larger TAB_{EXT} using the constant of the previous iteration incremented by $\mathtt{P}_{max}$ (step 2.1). Otherwise, the **repeat-until** cycle terminates.

The following theorem states the termination and the correctness of the algorithm.

THEOREM 5. *i) Algorithm 2 terminates and ii) an authorization* $\mathtt{A}$ *is valid at time* $\bar{\mathtt{t}}$ *if and only if there exists* $\langle \mathtt{A}, \Omega \rangle$ *in* TAB_{EXT} *such that* $\bar{\mathtt{t}}$ *satisfies* Ω.

In practice, we expect the algorithm to terminate at the first iteration in most cases.

ALGORITHM 2.
INPUT: The output $\langle L_1, \ldots, L_k \rangle$ of the CSD Algorithm and TAB_{CNS}.
OUTPUT: $\text{TAB}_{EXT}=\{\langle \mathtt{A}_i, \Omega_i \rangle \mid \mathtt{A}_i$ is a valid authorization for each instant satisfying $\Omega_i\}$
METHOD:
1. $k := 1$; $success := false$
2. **Repeat**
  2.1. $k := k + 1$; $current\_max := \bar{\mathtt{t}}_{max} + k \cdot \mathtt{P}_{max}$
  2.2. **For** each level L_i:
    Let $X_i \subseteq \text{TAB}_{CNS}$ containing all $\langle \mathtt{A}_m, \Xi_m \rangle$ and $\langle \mathtt{A}_m \ \langle \text{OP} \rangle \ \mathcal{A}, \Xi_m \rangle$ such that $\mathtt{A}_m$ appears in L_i and $\Xi_{m,i} \wedge^* \Xi_m \neq \emptyset$
    Repeat
      For each $\langle x, \Xi \rangle \in X_i$:
        (a) Let Θ be the conjunction of Ξ, $\Xi_{m,i}$ and $\{(\{true\}, \{t \leq current\_max\})\}$
        (b) **If** $x = \mathtt{A}_m \ \langle \text{OP} \rangle \ \mathcal{A}$ **then**
          $\Lambda := \emptyset$
          For each conjunct $\mathtt{C} = \bigwedge_{j=1}^{k} \neg \mathtt{A}_j \wedge \bigwedge_{l=k+1}^{m} \mathtt{A}_l$ appearing in $\mathcal{A}$:
            Let Γ be the conjunction of $\neg^* \Omega_1, \ldots, \neg^* \Omega_k, \Omega_{k+1}, \ldots, \Omega_m$
            Reassign to Λ the disjunction of Λ and Γ
          endfor
          If OP = WHENEVER **then** Reassign to Θ the conjunction of Λ and Θ
          If OP = PASTOP **then**
            Let t be the unique variable appearing in Θ
            Case OP **of**:
              UPON: Reassign to Θ the conjunction of $\{(\{true\}, \{t \geq \bar{t}\})\}$ and Θ, where $\bar{t}$ is the first instant satisfying Ξ and Λ
              ASLONGAS: Reassign to Θ the conjunction of $\{(\{true\}, \{t < \bar{t}\})\}$ and Θ, where $\bar{t}$ is the first instant satisfying Ξ and $\neg^* \Lambda$
        (c) **If** $\mathtt{A}_m$ is a positive authorization **then** Reassign to Θ the conjunction of Θ with the complement of each element in the set $\{\Omega_k \mid \mathtt{pn}(\mathtt{A}_k)=\text{'}-\text{'}, \mathtt{s}(\mathtt{A}_k)=\mathtt{s}(\mathtt{A}_m), \mathtt{o}(\mathtt{A}_k)=\mathtt{o}(\mathtt{A}_m), \mathtt{m}(\mathtt{A}_k)=\mathtt{m}(\mathtt{A}_m)\}$
        (d) Discard the inconsistent pairs from Θ
        (e) Add $\langle \mathtt{A}_m, \Theta \rangle$ to TAB_{EXT}
    until TAB_{EXT} does not change
  2.3. **If** $\forall \langle \mathtt{A}, \Omega \rangle$ in TAB_{EXT}:
    $\Omega \wedge^* \{(\{true\}, \{\bar{\mathtt{t}}_{max} + (k-2) \cdot \mathtt{P}_{max} < t \leq \bar{\mathtt{t}}_{max} + (k-1) \cdot \mathtt{P}_{max}\})\} \stackrel{\rightarrow}{=}^*$
    $\Omega \wedge^* \{(\{true\}, \{\bar{\mathtt{t}}_{max} + (k-1) \cdot \mathtt{P}_{max} < t \leq \bar{\mathtt{t}}_{max} + k \cdot \mathtt{P}_{max}\})\}$ **then** $success := true$
  Until $(success \vee k \geq \bar{k})$
3. **For** each $\langle \mathtt{A}, \Omega \rangle$ in TAB_{EXT}:
  substitute with ∞ each value $\bar{t}$ such that $\bar{\mathtt{t}}_{max} + (k-1) \cdot \mathtt{P}_{max} < \bar{t} \leq \bar{\mathtt{t}}_{max} + k \cdot \mathtt{P}_{max}$

Fig. 5. An algorithm for TAB_{EXT} generation

Example 10. Consider the TAB of Example 9. The levels computed by the CSD algorithm are illustrated in Figure 4. We now apply the algorithm for TAB_{EXT} generation. At the first iteration of the **repeat-until** cycle $k = 2$ and $current\_max$= = 1/14/99 (12/31/98 + 2 · 7), where 7 is the periodicity of *Weeks*. Let $\text{TAB}^{(i)}_{EXT}$ be the TAB_{EXT} resulting from the evaluation of level L_i. We have:

$X_1 = \{\langle$(manager,guidelines,write,+,Sam), $\{(\{\text{true}\}, \{1/1/95 \leq t \leq 5/20/95\})\}\rangle$, $\langle$(technical-staff,guidelines,read,+,Sam) $\{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\}, \{10/1/95 \leq t\})\}\rangle$, $\langle$(staff,document,read,+,Sam) $\{(\{t \equiv_7 (y+1) \forall y =$

$0,\ldots,4\}, \{\texttt{1/1/95} \leq t \leq \texttt{12/31/97}\})\}\rangle\}$.

$\mathrm{TAB}^{(1)}_{EXT} = \{\langle$(manager,guidelines,write,+,Sam), $\{(\{\mathrm{true}\}, \{\texttt{1/1/95} \leq t \leq$
$\leq \texttt{5/20/95}\})\}\rangle$, $\langle$(technical-staff,guidelines,read,+,Sam),
$\{(\{t \equiv_7 (y+1)\ \forall y = 0,\ldots,4\},\ \{\texttt{10/1/95} \leq t \leq \texttt{1/14/99}\})\}\rangle$,
$\langle$(staff,document,read,+,Sam), $\{(\{t \equiv_7 (y+1)\forall y = 0,\ldots,4\},$
$\{\texttt{1/1/95} \leq t \leq \texttt{12/31/97}\})\}\rangle$.

$X_2 = \{\langle$(temporary-staff,document,read,+,Sam) ASLONGAS $\neg$(summer-staff, document,read,+,Sam), $\{(\{t \equiv_7 (y+1)\forall y = 0,\ldots,4\}, \{\texttt{1/1/96} \leq t \leq$ $\leq \texttt{12/31/98}\})\}\rangle$, $\langle$(technical-staff,report,write,-,Sam) WHENEVER $\neg$(technical-staff,guidelines,read, +,Sam) $\{(\{\mathrm{true}\}, \{\texttt{1/1/95} \leq t\})\}\rangle\}$.

From $\langle$(temporary-staff,document,read,+,Sam) ASLONGAS $\neg$(summer-staff, document,read,+,Sam), $\{(\{t \equiv_7 (y{+}1)\forall y = 0,\ldots,4\}, \{\texttt{1/1/96} \leq t \leq \texttt{12/31/98}\})\}\rangle$ and the authorizations in $\mathrm{TAB}^{(1)}_{EXT}$ we derive: $\langle$(temporary-staff,document,read, +,Sam), $\{(\{t \equiv_7 (y+1)\ \forall y = 0,\ldots,4\}, \{\texttt{1/1/96} \leq t \leq \texttt{12/31/98}\})\}\rangle$.
From $\langle$(technical-staff,report,write,-,Sam) WHENEVER $\neg$(technical-staff,guidelines,read, +,Sam), $\{(\{\mathrm{true}\}, \{\texttt{1/1/95} \leq t\})\}\rangle\}$ we derive: $\langle$(technical-staff,report,write,-,Sam), $\{(\{\mathrm{true}\}, \{\texttt{1/1/95} \leq t \leq \texttt{9/30/95}\}),$ $(\{t \equiv_7 y\ y = 0, 6\}, \{\texttt{10/1/95} \leq t \leq \texttt{1/14/99}\})\}\rangle\}$. Thus,

$\mathrm{TAB}^{(2)}_{EXT} = \mathrm{TAB}^{(1)}_{EXT} \cup \{\langle$(temporary-staff,document,read,+,Sam),
$\{(\{t \equiv_7 (y+1)\ \forall y = 0,\ldots,4\}, \{\texttt{1/1/96} \leq t \leq \texttt{12/31/98}\})\}\rangle$,
$\langle$(technical-staff,report,write,-,Sam), $\{(\{\mathrm{true}\},$
$\{\texttt{1/1/95} \leq t \leq \texttt{9/30/95}\}), (\{t \equiv_7 y\ y = 0, 6\}, \{\texttt{10/1/95} \leq t \leq$
$\leq \texttt{1/14/99}\})\}\rangle\}$.

$X_3 = \{\langle$(technical-staff,report,write,+,Sam) UPON $\neg$(manager, guidelines,write,+,Sam)$\wedge\neg$(staff,guidelines,write,+,Sam), $\{(\{t \equiv_7 (y+1)\ y = 0, 4\}, \{\texttt{1/1/95} \leq t\})\}\rangle\}$

From $\langle$(technical-staff,report,write,+,Sam) UPON $\neg$(manager,guidelines, write,+,Sam)$\wedge\neg$(staff,guidelines,write,+,Sam), $\{(\{t \equiv_7 (y+1)\ y = 0, 4\},$ $\{\texttt{1/1/95} \leq t\})\}\rangle\}$ and the authorizations in $\mathrm{TAB}^{(2)}_{EXT}$ we derive: $\langle$(technical-staff,report,write,+,Sam),$\{(\{t \equiv_7 (y+1)\ y = 0, 4\},\ \{\texttt{10/1/95} \leq$ $t \leq \texttt{1/14/99}\})\}\rangle$. Hence,

$\mathrm{TAB}^{(3)}_{EXT} = \mathrm{TAB}^{(2)}_{EXT} \cup \{\langle$(technical-staff,report,write,+,Sam),
$\{(\{t \equiv_7 (y+1)\ y = 0, 4\},\ \{\texttt{10/1/95} \leq t \leq \texttt{1/14/99}\})\}\rangle\}$.

success is set to *true* and the **repeat-until** cycle terminates.
The last step of the algorithm substitutes ∞ to each value $\bar{t}$ such that $\texttt{1/7/99} < \bar{t} \leq \texttt{1/14/99}$. Hence, TAB_{EXT} is equal to:

$\{\langle$(manager, guidelines, write, +, Sam), $\{(\{\mathrm{true}\}, \{\texttt{1/1/95} \leq t \leq \texttt{5/20/95}\})\}\rangle$,
$\langle$(technical-staff,guidelines,read,+,Sam), $\{(\{t \equiv_7 (y+1)\ \forall y = 0,\ldots,4\},$
$\{\texttt{10/1/95} \leq t\})\}\rangle$,
$\langle$(staff, document, read, +, Sam), $\{(\{t \equiv_7 (y+1)\ \forall y = 0,\ldots,4\}, \{\texttt{1/1/95} \leq t \leq$
$\texttt{12/31/97}\})\}\rangle$,
$\langle$(temporary-staff,document,read,+,Sam), $\{(\{t \equiv_7 (y+1)\ \forall y = 0,\ldots,4\},$
$\{\texttt{1/1/96} \leq t \leq \texttt{12/31/98}\})\rangle$,
$\langle$(technical-staff,report,write,-,Sam), $\{(\{\mathrm{true}\}, \{\texttt{1/1/95} \leq t \leq \texttt{9/30/95}\}),$

$(\{t \equiv_7 y\ y = 0, 6\}, \{10/1/95 \leq t\})\}\rangle$,
$\langle$(technical-staff,report,write,+,Sam),$\{(\{t \equiv_7 (y+1)\ y = 0, 4\}, \{10/1/95 \leq$
$\leq t\})\}\rangle\}$.

Once we have generated 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 $\langle A, \Omega\rangle$ exists in TAB_{EXT} such that s(A)=s_1, o(A)=o_1, m(A)=m_1, pn(A)= '+', and t satisfies Ω.

6. AUTHORIZATION UPDATES

Valid authorizations can change due to modifications of the TAB. In this section we discuss administrative operations for changing the TAB and illustrate a strategy to incrementally maintain TAB_{EXT} upon the execution of administrative operations.

6.1 Administrative operations

Administrative operations allow users to specify, delete, or modify authorizations and derivation rules. In our model we assume *ownership* is enforced: the user who creates an object is considered its owner and can, as such, regulate access by others to the object. By regulating access we mean that the user can specify authorizations, either directly or through derivation rules, on the object. In other words the user can specify any authorization on the object and any rule where the object appears on the left hand side of the operator. Authorizations and derivation rules can then be removed or changed only by the user who specified them.

We note that administrative operations can be carried out directly on the TAB, i.e., allowing users direct access and modification to authorizations/rules, or through specifications in a language for administrative operations. This language would interface the user and the TAB by interpreting users' requests to add, remove, or modify authorizations and rules and translating them appropriately on the TAB. For instance, a language such as the one presented in [Bertino et al. 1996a] can be used.

The development of an administrative policy for the specification of temporal authorizations and rules is outside of the scope of this paper. This is the reason behind the choice of the ownership policy, which provides decentralized administration (each user administers his/her own objects) without complications that can be introduced by the delegation of administrative privileges. We note, however, that the administrative policy is orthogonal to the authorization model and, therefore, any administrative policy can be applied to our model. For instance, delegation of administration through either administrative privileges (such as in [Bertino et al. 1996a]) or through grant option (such as in [Bertino et al. 1997]) can be considered. A user would then be allowed only to specify authorizations/rules if s/he is so authorized by the administrative policy.

We note that the administrative policy has some effects on how parametric rules are expanded by the system. Remember that parametric rules can contain the wild card character, meaning any value, in place of subject, object, or access mode in the authorizations. Parametric rules specified by a user (i.e., where the user appears as a grantor in the authorization on the left of the operator) can produce only authorizations that the user can specify. In our case, they produce only authorizations on objects owned by the user who specified the rules. To clarify, consider

Example 6 and suppose there were two additional objects in the system owned by another user, `Jim`. The resulting rules would be exactly the same. Note however that ownership limits only the value that the object element in the authorization on the left of the operator (the authorization to be derived) can assume. It does not limit the value of the object elements in the authorizations on the right of the operator. Then, rules specified by a user can refer to objects that the user does not own.[10] In the case of parametric rules, if the wild card character appears for the object in the authorization on the left of the operator, it will be instantiated only to objects owned by the user who specified the rules. If it appears only in authorizations on the right of the operator it can instead be instantiated to any value.

6.2 Implementation issues

Administrative operations, by changing the TAB, may change the validity of authorizations and, consequently, require changes to the corresponding TAB_{EXT}. For instance, consider the deletion of a derivation rule. The first effect of this operation is the deletion from TAB_{EXT} of the authorizations, if any, derivable from it. However, the rule deletion may have several other effects on the TAB_{EXT} content, such as the deletion of other authorizations, besides the one directly derivable from the rule. This happens if the authorization on the left hand side of the deleted rule appears on the right hand side of other derivation rules, not preceded by the $\neg$ operator. By contrast, if the authorization on the left hand side of the deleted rule appears on the right side of other derivation rules preceded by the $\neg$ operator, then the deletion of the rule may cause the insertion of other authorizations in TAB_{EXT}. The rule deletion may also require to update the TAB stratification. Further effects are possible if the deleted rule allows the derivation of a negative authorization. In this case the deletion of the rule may additionally cause the insertion in TAB_{EXT} of the positive authorizations, if any, that were invalidated by the presence of the negative authorization derivable from the rule. The following example clarifies these concepts.

Example 11. Consider the TAB of Example 9 and the corresponding TAB_{EXT} computed in Example 10. Suppose that rule $\mathtt{R}_3$ is removed. Authorization `(technical-staff,report,write, -,Sam)`, $\{(\{\text{true}\},\{\mathtt{1/1/95} \leq t \leq \mathtt{9/30/95}\}),$ $(\{t \equiv_7 y \;\; y = 0,6\},\{\mathtt{10/1/95} \leq t\})\}$, derived from $\mathtt{R}_3$, must be removed from TAB_{EXT}. Due to this deletion, now rule $\mathtt{R}_2$ allows the derivation of authorization `(technical-staff,report,write,+,Sam)` for each Monday and Friday after `5/20/95`, since in the interval `[5/21/95,9/30/95]` authorization `(technical-staff, report,write,+,Sam)` is no longer invalidated by the removed negative authorization. Therefore, authorization `(technical-staff,report,write,+,Sam)`,$(\{t \equiv_7 (y+1) \;\; y = 0,4\},\{\mathtt{5/21/95} \leq t \leq \mathtt{9/30/95}\})\}$ must be added to TAB_{EXT}. Moreover, the TAB stratification, illustrated in Figure 4, must be modified. $\langle$`(technical-staff,report,write,-,Sam)`, $\{(\{\text{true}\},\{\mathtt{1/1/95} \leq t \leq \mathtt{1/21/99}\})\}\rangle$ must be removed from the stratification since $\mathtt{R}_3$ was the only element in TAB in which `(technical-staff,report,write,-,Sam)` appeared. Consequently,

[10] Administrative privileges can be considered which restrict the objects to which users can refer in rules [Bertino et al. 1996a].

$\langle$(technical-staff,report,write,+,Sam), $\{(\{\text{true}\},\{1/1/95 \leq t \leq 1/21/99\})\}\rangle$ must be moved from level 3 to level 2.

Note that updates similar to those illustrated in Example 11 are required for all the other operations that enforce changes to the TAB. We have devised a set of algorithms for incrementally maintaining both the TAB_{EXT} and the TAB stratification without the need of recomputing them all upon the execution of administrative operations. The algorithms are based on techniques similar to those used for the maintenance of materialized views in constraint and deductive databases [Gupta et al. 1993; Lu et al. 1996; Lu et al. 1995]. These techniques can be successfully employed in our context due to the correspondence between a TAB and a constraint logic program, shown in subsection 3.3. All our algorithms (see [Ferrari 1998] for a detailed description) are based on both the Dred [Gupta et al. 1993] and the Stdel algorithm [Lu et al. 1995] for view maintenance, and extend them to the consideration of negative authorizations. The maintenance algorithms we have developed for the periodic model are an optimization of the ones developed for the non-periodic model [Bertino et al. 1996a], which were based on the Dred algorithm only.

Note that in the worst case the maintenance algorithms must recompute the whole TAB_{EXT} and the TAB stratification. It should be noted, however, that an analysis of the worst case complexity of the maintenance algorithms is not very significant, since this case is very unlikely to be reached in practice. Indeed, the worst case arises only if all of the following three conditions are satisfied:

(1) all the derivation rules in TAB form a unique chain, that is, the set $R = \{\mathtt{R}_1, \ldots, \mathtt{R}_n\}$ of derivation rules in TAB is such that $\forall i \in \{1, \ldots, n-1\}$: $\mathtt{R}_i = \mathtt{A}_i \ \langle\text{OP}\rangle \ \mathcal{A}_i$, $\mathtt{R}_{i+1} = \mathtt{A}_{i+1} \ \langle\text{OP}\rangle \ \mathcal{A}_{i+1}$ with $\mathtt{A}_{i+1}$ appearing in $\mathcal{A}_i$;

(2) no pairs of temporal constraints in the derivation rules in TAB are disjoint;

(3) the administrative operation affects the last rule in the chain.

This is clearly a very unlikely situation. We expect that the size of the chains of derivation rules will be very small in real cases. This expectation is also supported by the experiments we have carried out for the non-periodic model [Bertino et al. 1996a], which have shown that the materialization strategy can be successfully adopted in practice [Bertino et al. 1996c], despite of the worst case. The experiments have been carried out on a HP9000/720 with 64Mb of RAM under the HP-UX 10.01 operating system. The prototype system used to perform the experiments has been implemented on top of the INFORMIX DBMS [Informix Software 1993], using the INFORMIX-ESQL/C language. In particular, the data structures used to implement the authorization model are based on the INFORMIX System Catalogs. The graphical interface has been implemented using the OSF/Motif toolkit [Foundation 1993]. In order to test the feasibility of the materialization strategy, the prototype system we have developed implements both an algorithm enforcing the run-time derivation strategy and all the algorithms to incrementally maintain TAB_{EXT} and the TAB stratification upon the execution of administrative operations. Experiment tests, with TABs of different sizes and characteristics, have shown that the materialization strategy is always preferable to a run-time derivation approach when administrative operations are less than 15%-20% of the number of access requests submitted to the system. This is a reasonable assumption in real situations, since, generally, access requests are considerably more frequent than administrative requests.

Based on the prototype system developed for the non-periodic model, we are currently implementing a prototype system for the model presented in this paper. This has required modifying the non-periodic prototype along two main directions. The first extension is the implementation of the maintenance algorithms we have developed for the periodic model. The second extension is related to the management of periodic time. As we have seen in Section 2, we provide both a symbolic and a constraint formalism to represent periodic time. We are currently implementing a procedure to convert any symbolic expression into an equivalent set of simple periodicity constraints, along the lines skectched in subsection 2.3.

Moreover, we are currently investigating efficient algorithms to perform the operations (i.e., union, complement, and intersection) on temporal constraints associated with authorizations and rules, using a preprocessing technique at the symbolic level. Performing the operations at the symbolic level has several advantages, in terms of user friendliness, saving of storage space and computational cost. However, it seems that in certain "critical cases" executing the operations necessarily require a translation into the constraint formalism. The idea is therefore to operate as much as possible at the symbolic level and shift to periodicity and gap-order constraints only when required.

7. CONCLUSIONS AND FUTURE WORK

Existing authorization models are based on the use of authorizations stating permissions of subjects (users, groups, or roles) to access objects. An authorization is considered valid all the time from when it is granted to the time it is revoked. Mapping real-life protection requirements onto this paradigm becomes often very cumbersome. As a matter of fact, many practical requirements call for more expressiveness and flexibility in specifying authorizations. In this paper we have presented an authorization model where authorizations can be periodic and have a limited time of validity. The model also allows users to specify rules for the automatic derivation of new (periodic) authorizations. The resulting model is therefore very flexible and powerful in terms of the kinds of protection requirements that it can represent. Obviously, the flexibility provided to the users requires a non trivial underlying formal model where temporal constraints, periodicity constraints, and derivation rules can be represented.

We are currently extending our work in several directions. A first direction concerns the investigation of security requirements for new applications, such as workflow systems and computer supported cooperative work, to assess whether our model is adequate to express those requirements. A second direction concerns the investigation and use of different resolution policies to solve conflicts between positive and negative authorizations. In the paper we have assumed the use of the denials-take-precedence policy. This policy has the advantage of always taking the safest solution (i.e., denying the access) in case of conflicts. However, it does not support all forms of exceptions, for instance, it is not possible to specify positive exceptions to negative authorizations. A third direction concerns the enrichment of the authorization specification language. For instance, in the paper we have assumed that parametric authorizations can be specified by using a special symbol (the wild card character) which varies across all the values of a domain. Rules can be extended to the consideration of multiple variables, varying over the same domain, in the authorizations within a rule. A fourth direction concerns the development of tools supporting authorization administration. A final direction concerns the investigation of alternative implementation techniques for our model, such as the use of database triggers.

APPENDIX

A. DATALOG$^{\mathbf{NOT},\equiv Z,<Z}$

In this paper we used $Datalog^{\mathbf{not},\equiv Z,<Z}$ to specify the semantics of a set of periodic authorizations and rules, and the algorithm to generate implicit authorizations mimics a fixpoint computation of the model of a $Datalog^{\mathbf{not},\equiv Z,<Z}$ program. $Datalog^{\mathbf{not},\equiv Z,<Z}$ is a simple extension of $Datalog^{\equiv Z,<Z}$ [Toman et al. 1994] with non-monotonic negation [Gelder et al. 1991], however, to our knowledge, it was never considered in the literature. $Datalog^{\mathbf{not},\equiv Z,<Z}$ programs are defined as follows.

Definition 16. A $Datalog^{\mathbf{not},\equiv Z,<Z}$ program P is a finite set of (function-free) clauses of the form

$$B \leftarrow D_1, \ldots, D_m, \mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}, PC, GC$$

where $B, D_1, \ldots, D_{m+n}$ are atoms, PC is a satisfiable conjunction of simple periodicity constraints, GC is a satisfiable conjunction of gap-order constraints, and **not** represents non monotonic negation.

Bottom-up evaluation of Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs requires the execution of operations on temporal constraints, that we have defined as sets of pairs (PC, GC). In addition to the operations of conjunction ($\wedge^*$) and complement ($\neg^*$), defined in subsection 2.2, we need the operations of subsumption and projection (π) defined in [Toman et al. 1994]. Intuitively, if x, y, and z are the variables in both PC and GC, then $\pi_{xy}(PC, GC)$ returns a set of pairs (PC_i, GC_i) obtained from (PC, GC) by dropping all the constraints involving variable z, after computing and adding all the constraints on the remaining variables implied by the dropped ones. The π operation also discards any resulting pair that is inconsistent. The subsumption operation has its intuitive semantics: a pair (PC_1, GC_1) is subsumed by (PC_2, GC_2) (having the same set of variables) if any assignment satisfying (PC_1, GC_1) satisfies also (PC_2, GC_2). Subsumption and projection operations can be easily extended to operate on temporal constraints, similarly to $\wedge^*$ and $\neg^*$.

Temporal constraints serve as a basis to define a non-ground interpretation for $Datalog^{\mathbf{not},\equiv Z,<Z}$ programs. A $(\equiv, <)$ interpretation is any set of pairs of the form (B, Ξ), where B is a predicate symbol, and Ξ is a temporal constraint.

Given a $Datalog^{\mathbf{not},\equiv Z,<Z}$ program P we can define an operator $\mathrm{TP}^{\mathbf{not},\equiv Z,<Z}$ that maps $(\equiv, <)$ interpretations to $(\equiv, <)$ interpretations. In the following we denote with π^* the projection operation on sets Ξ.

Definition 17. Let P be a $Datalog^{\mathbf{not},\equiv Z,<Z}$ program and I a $(\equiv, <)$ interpretation.

$$\begin{aligned}
&\mathrm{TP}^{\mathbf{not},\equiv Z,<Z}(I) = I \cup \\
&\quad \{\ (B, \Xi) : B \leftarrow D_1, \ldots, D_m, \mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}, PC, GC \in P \\
&\quad\ \ (D_r, \Xi_r) \in I,\ \forall r = 1, \ldots, m, \\
&\quad\ \ \Theta = \Xi_1 \wedge^* \ldots \wedge^* \Xi_m \ \wedge^* \neg^*(\Xi_{m+1}) \wedge^* \ldots \wedge^* \neg^*(\Xi_{m+n}) \wedge^* \{(PC, GC)\}, \\
&\quad\ \ \Xi = \pi^*_{Var(B)}(\Theta),\ (B, \Xi) \text{ is not subsumed by } I\ \}
\end{aligned}$$

where $Var(B)$ denotes the set of free variables in atom B. The variables of the periodicity and gap-order constraints are renamed using the variable names in the associated atoms of the clauses.

We know from the literature that the model of a positive logic program can be computed by the so called fixpoint iteration method; starting from the empty interpretation all the possible inferences from rules in the program are drawn until a fixpoint is reached.

If we restrict our attention to stratified (or locally stratified) [Gelder et al. 1991] Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs the following procedure, based on the fixpoint iteration method, can be used to evaluate programs.

ALGORITHM 3. **(Naive Bottom-up evaluation of stratified Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs)** *Let P be a Datalog$^{\mathbf{not},\equiv Z,<Z}$ program, let $P_1,\ldots,P_n$ be a stratification of P.*[11]

$I := \emptyset$
For $i := 1$ *to* n **do**
  repeat
    $I := TP_i^{\mathbf{not},\equiv Z,<Z}(I)$
  until I *does not change*
endfor
return I

Termination of Algorithm 3 is not guaranteed for any stratified Datalog$^{\mathbf{not},\equiv Z,<Z}$ program, as Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs can express any Turing computable function [Revesz 1993]. There is then the necessity to devise syntactic restrictions of stratified Datalog$^{\mathbf{not},\equiv Z,<Z}$ to guarantee termination. One of these restrictions is Datalog$^{\mathbf{not},\equiv Z,<\mathcal{D}}$, where $\mathcal{D}$ is a finite subset of Z, i.e., the gap-order constraints are on a finite subset of the integers. We prove that for any stratified Datalog$^{\mathbf{not},\equiv Z,<\mathcal{D}}$ program, Algorithm 3 terminates returning a non-ground representation of the unique (ground) model of the program. The ground model can be obtained with the standard bottom-up evaluation, that is, by replacing in Algorithm 3 TP$^{\mathbf{not},\equiv Z,<\mathcal{D}}$ with the standard TP operator. Finally note that Algorithm 3 may return different non-ground models for the same input program. This simply means that there may be several finite representations for the same ground model. The following theorem formalizes these concepts.

THEOREM 6. *Algorithm 3 terminates for any stratified Datalog$^{\mathbf{not},\equiv Z,<\mathcal{D}}$ program P. Let M_P be any output of Algorithm 3 on P. Let G_P be the output of the standard bottom-up evaluation of P, then for any predicate symbol B in P:*

$$B(c_1,\ldots,c_k) \in G_P \iff (B,\Xi) \in M_P \wedge [c_1/x_1,\ldots,c_k/x_k] \models \mathcal{F}(\Xi)$$

where $x_1,\ldots,x_k$ are the free variables in atom B, $[c_1/x_1,\ldots,c_k/x_k]$ denotes the substitution of those variables with constants $c_1,\ldots,c_k$, and $\mathcal{F}(\Xi)$ is the boolean formula associated with Ξ (cfr. subsection 2.2).

PROOF (*Termination*): The algorithm consists of a finite iteration, bound by the number of strata in the input program, containing an inner **repeat-until** cycle. The **repeat-until** cycle never terminates only in case that at each iteration either a new element (B,Ξ) is added to I or a new constraint is generated for a predicate B appearing in I. We show that this is not possible. First of all, the number of predicates in P is finite. Moreover, only a finite number of gap-order constraints can be generated, since we assume that

[11] P_i contains rules of stratum i, $i = 1,\ldots,n$.

gap-order constraints are on a finite domain. Let us consider periodicity constraints. Let $pc_1, \ldots, pc_m$ be the set of all the simple periodicity constraints appearing in P. Let k_i be the modulo factor in pc_i, $i = 1, \ldots, m$. It is easy to show that the operations performed on simple periodicity constraints by the TP operator ensure that the modulo factor of each simple periodicity constraint appearing in I must be less then or equal to $lcm\{k_1, \ldots, k_m\}$. Hence, only a finite number of simple periodicity constraints can be generated. Therefore, we can conclude that the algorithm terminates.
We prove the second part of the theorem by induction on the number of strata in P. Let G_P^i be the output of the i-th iteration of the standard bottom-up evaluation, and let M_P^i be the output of the i-th iteration of Algorithm 3. We prove, by induction on i that, for any predicate symbol B in P, $B(c_1, \ldots, c_k) \in G_P^i$ iff there exist $(B, \Xi) \in M_P^i$ and $c_1, \ldots c_k$ such that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$, where $\{x_1, \ldots, x_k\}$ are the free variables in atom B. In what follows we denote with $G_P^{i,j}$ the output of the j-th rule application in the i-th iteration of the standard bottom-up evaluation. Similarly, $M_P^{i,j}$ denotes the output of the j-th rule application in the i-th iteration of Algorithm 3.

(*Basis: $i = 1$*): Proving the thesis is equivalent to prove, by induction on j, that: *(a)* if $B(c_1, \ldots, c_k) \in G_P^{1,j}$, then $\exists(B, \Xi) \in M_P^1$ such that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$, and *(b)* if there exist $(B, \Xi) \in M_P^{1,j}$ and $c_1, \ldots, c_k$ such that $[c_1/x_1, \ldots c_k/x_k] \models \mathcal{F}(\Xi)$, then $B(c_1, \ldots, c_k) \in G_P^1$.

(*Basis: j=1*): Let us first consider *(a)*. Let $P_1, \ldots, P_n$ be the stratification of P. By definition of stratified program [Gelder et al. 1991] clauses in stratum P_1 do not contain negative literals. Therefore, if $B(c_1, \ldots, c_k) \in G_P^{1,1}$, then the first rule applied is a rule $B \leftarrow PC, GC$. Let $Var = \{x_1, \ldots x_k, y_1, \ldots, y_l\}$ be the variables in the rule. If $B(c_1, \ldots, c_k) \in G_P^{1,1}$, then there exists a substitution $\mathcal{A} = [c_1/x_1, \ldots, c_k/x_k, d_1/y_1, \ldots, d_l/y_l]$ such that $\mathcal{A}$ satisfies PC and GC. Consider the evaluation of $B \leftarrow PC, GC$ by Algorithm 3. (B, Ξ) is added to M_P^1 if it is not subsumed by M_P^1, where $\Xi = \pi^*_{Var(B)}(\Theta)$, $\Theta = \{(PC, GC)\}$. By hypothesis $\mathcal{A} \models \mathcal{F}(\Theta)$ and, by Lemma 4.7 in [Toman et al. 1994], $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$. Thus, if (B, Ξ) is not subsumed by M_P^1, the thesis holds. If (B, Ξ) is subsumed by M_P^1, it means that $\exists(B, \Xi') \in M_P^1$ such that Ξ' subsumes Ξ. Therefore $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi')$. Thus, the claim holds. Let us now prove point *(b)*. If $(B, \Xi) \in M_P^{1,1}$, then the first rule application in the first iteration of Algorithm 3 uses a rule $B \leftarrow PC, GC$ of P_1, such that $\Xi = \pi^*_{Var(B)}(\Theta)$, where $\Theta = \{(PC, GC\}$. Let $Var = \{x_1, \ldots x_k, y_1, \ldots y_l\}$ be the variables in the rule. By Lemma 4.7 in [Toman et al. 1994], $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$ implies that there exists $\mathcal{A} = [c_1/x_1, \ldots, c_k/x_k, d_1/y_1, \ldots, d_l/y_l]$, such that $\mathcal{A} \models \mathcal{F}(\Theta)$. Thus, when the rule $B \leftarrow PC, GC$ is considered by the standard bottom-up evaluation, $B(c_1, \ldots, c_k)$ is added to G_P^1.

(*Induction*): Suppose we have proved the thesis for j, we prove it for $j+1$. Consider point *(a)* first. Suppose that the $(j+1)$-th rule application of the first iteration of the standard bottom-up evaluation uses rule $B \leftarrow D_1, \ldots, D_m, \mathbf{not}D_{m+1}, \ldots, \mathbf{not}D_{m+n}, PC, GC$ of P_1. Let $Var = \{x_1, \ldots x_k, y_1 \ldots y_l\}$ be the variables in the rule. If $B(c_1, \ldots, c_k)$ is added to $G_P^{1,(j+1)}$, then there exists $\mathcal{A} = [c_1/x_1, \ldots c_k/x_k, d_1/y_1, \ldots d_l/y_l]$ such that $D_1(t_1), \ldots, D_m(t_m) \in G_P^{1,j}$, $D_{m+1}(t_{m+1}), \ldots, D_{m+n}(t_{m+n}) \notin G_P^{1,j}$, $\mathcal{A}$ satisfies PC and GC, where t_r is the tuple obtained by applying $\mathcal{A}$ to D_r, $r = 1, \ldots, m+n$. By inductive hypothesis $D_1(t_1), \ldots, D_m(t_m) \in G_P^{1,j}$ implies that there exist $\Xi_1, \ldots, \Xi_m$, such that t_r satisfies Ξ_r and $(D_r, \Xi_r) \in M_P^1$, $r = 1, \ldots, m$. Hence, $\mathcal{A} \models \mathcal{F}(\Xi_r)$, $r = 1, \ldots, m$. By contrast, the inductive hypothesis implies that since $D_r(t_r) \notin G_P^{1,j}$, then there is no (D_r, Ξ_r) in M_P^1 such that t_r satisfies Ξ_r, $r = m+1, \ldots, m+n$. Thus, $\mathcal{A} \models \mathcal{F}(\neg^* \Xi_r)$, $r = m+1, \ldots m+n$. Therefore, consider the evaluation of $B \leftarrow D_1, \ldots, D_m, \mathbf{not}D_{m+1}, \ldots, \mathbf{not}D_{m+n}, PC, GC$

by Algorithm 3. (B, Ξ) is added to M_P^1, if it is not subsumed by M_P^1, where $\Xi = \pi^*_{Var(B)}(\Theta)$ and $\Theta = \Xi_1 \wedge^* \ldots \wedge^* \Xi_m \wedge^* \neg^*\Xi_{m+1} \wedge^* \ldots \wedge^* \neg^*\Xi_{m+n} \wedge^* \{(PC, GC)\}$. By hypothesis $\mathcal{A} \models \mathcal{F}(\Xi_r)$, $r = 1, \ldots, m$ and $\mathcal{A} \models \mathcal{F}(\neg * \Xi_r)$, $r = m+1, \ldots m+n$. Moreover, $\mathcal{A}$ satisfies PC and GC. Thus, $\mathcal{A} \models \mathcal{F}(\Theta)$, and, by Lemma 4.7 in [Toman et al. 1994], $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$. Therefore, if (B, Ξ) is not subsumed by M_P^1, $\exists (B, \Xi) \in M_P^1$ such that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$, which proves the thesis. Suppose that (B, Ξ) is subsumed by M_P^1. Then, $\exists (B, \Xi') \in M_P^1$ such that Ξ' subsumes Ξ. This implies that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi')$, and therefore we can conclude the thesis.
Let us now prove point *(b)*. Suppose that the $(j+1)$-th rule application of the first iteration of Algorithm 3 uses rule $B \leftarrow D_1, \ldots, D_m, \mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}, PC, GC$ of P_1, and let $Var = \{x_1, \ldots x_k, y_1 \ldots y_l\}$ be the variables in the rule. If (B, Ξ) is added to $M_P^{1,j+1}$, it means that $\exists (D_r, \Xi_r) \in M_P^{1,j}$, $r = 1, \ldots, m$ such that $\Xi = \pi^*_{Var(B)}(\Theta)$, where $\Theta = \Xi_1 \wedge^* \ldots \wedge^* \Xi_m \wedge^* \ldots \wedge^* \neg^*\Xi_{m+1} \wedge^* \ldots \wedge^* \neg^*\Xi_{m+n} \wedge^* \{(PC, GC)\}$ and Ξ_r is the constraint associated with D_r in $M_P^{1,j}$ $r = 1, \ldots m+n$. By hypothesis, there exist $c_1, \ldots c_k$ such that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$. This implies, by Lemma 4.7 in [Toman et al. 1994], that there exists $\mathcal{A} = [c_1/x_1, \ldots, c_k/x_k, d_1/y_1, \ldots, d_l/y_l]$ such that $\mathcal{A} \models \mathcal{F}(\Theta)$. Then, $\mathcal{A} \models \mathcal{F}(\Xi_1), \ldots, \mathcal{A} \models \mathcal{F}(\Xi_m)$, $\mathcal{A} \models \mathcal{F}(\neg^*\Xi_{m+1}), \ldots, \mathcal{A} \models \mathcal{F}(\neg^*\Xi_{m+n})$, and $\mathcal{A}$ satisfies PC and GC. Let t_r be the projection of $\mathcal{A}$ onto the variables in Ξ_r, $r = 1, \ldots, m+n$. Thus, t_1 satisfies $\Xi_1, \ldots, t_m$ satisfies Ξ_m, t_{m+1} satisfies $\neg^*\Xi_{m+1}, \ldots, t_{m+n}$ satisfies $\neg^*\Xi_{m+n}$. By inductive hypothesis, $(D_r, \Xi_r) \in M_P^{1,j}$ and t_r satisfies Ξ_r, imply $D_r(t_r) \in G_P^1$, $r = 1, \ldots, m$. Similarly, t_r satisfies $\neg^*\Xi_r$, implies $D_r(t_r) \notin G_P^1$, $r = m+1, \ldots, m+n$. Therefore, when the rule $B \leftarrow D_1, \ldots, D_m, \mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}, PC, GC$ is considered by the standard bottom-up evaluation, $B(c_1, \ldots, c_k)$ is added to G_P^1.

(*Induction*): Suppose we have proved the thesis for i, we prove it for $i+1$, i.e., we prove that $B(c_1, \ldots, c_k) \in G_P^{i+1}$ iff there exist $(B, \Xi) \in M_P^{i+1}$ and $c_1, \ldots, c_k$ such that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$. Analogously to the basis case, proving the thesis is equivalent to prove, by induction on j, that: *(a)* if $B(c_1, \ldots, c_k) \in G_P^{i+1,j}$, then $\exists (B, \Xi) \in M_P^{i+1}$ such that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi)$, and *(b)* if there exist $(B, \Xi) \in M_P^{i+1,j}$ and $c_1, \ldots c_k$ such that $[c_1/x_1, \ldots c_k/x_k] \models \mathcal{F}(\Xi)$, then $B(c_1, \ldots, c_k) \in G_P^{i+1}$.

(*Basis: j=1*): We start by proving point *(a)*. Suppose that the first rule application of the $(i+1)$-th iteration of the standard bottom-up evaluation uses rule $B \leftarrow D_1, \ldots, D_m$, $\mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}$, PC, GC of P_{i+1}. Let $Var = \{x_1, \ldots x_k, y_1 \ldots, y_l\}$ be the variables in the rule. If $B(c_1, \ldots, c_k)$ is added to $G_P^{i+1,1}$, then there exists $\mathcal{A} = [c_1/x_1, \ldots, c_k/x_k$, $d_1/y_1, \ldots, d_l/y_l]$ such that $D_1(t_1), \ldots, D_m(t_m) \in G_P^i$, $D_{m+1}(t_{m+1}), \ldots, D_{m+n}(t_{m+n}) \notin G_P^i$, and $\mathcal{A}$ satisfies PC and GC, where t_r is the tuple obtained by substituting $\mathcal{A}$ into D_r, $r = 1, \ldots, m+n$. By inductive hypothesis, $D_1(t_1), \ldots, D_m(t_m) \in G_P^i$ implies $\exists (D_r, \Xi_r) \in M_p^i$, such that t_r satisfies Ξ_r, $r = 1, \ldots, m$. Hence, $\mathcal{A} \models \mathcal{F}(\Xi_r)$, $r = 1, \ldots, m$. By contrast, the inductive hypothesis implies that since $D_r(t_r) \notin G_P^i$, then there is no (D_r, Ξ_r) in M_P^i such that t_r satisfies Ξ_r, $r = m+1, \ldots, m+n$. Hence, $\mathcal{A} \models \mathcal{F}(\neg^*\Xi_r)$, $r = m+1, \ldots, m+n$. Therefore, consider the evaluation of $B \leftarrow D_1, \ldots, D_m, \mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}, PC, GC$ by Algorithm 3. (B, Ξ) is added to M_P^{i+1}, if it is not subsumed by M_P^{i+1}, where $\Xi = \pi^*_{Var(B)}(\Theta)$ and $\Theta = \Xi_1 \wedge^* \ldots \wedge^* \Xi_m \wedge^* \neg^*\Xi_{m+1} \wedge^* \ldots \wedge^* \neg^*\Xi_{m+n} \wedge^* \{(PC, GC)\}$. Using the same reasonings we have done to prove the corresponding case in the inductive step when $i = 1$, we can conclude that this implies $\exists (B, \Xi') \in M_P^{i+1}$ such that $[c_1/x_1, \ldots, c_k/x_k] \models \mathcal{F}(\Xi')$, which proves the thesis.
We omit the proof of point *(b)* since it is analogous to the proof of the corresponding case in the inductive step when $i = 1$

(*Induction*): We omit the prove of the inductive step as it is analogous to the corresponding case for i = *1*. □

B. PROOFS

In the following we use P instead of P_{TAB} to denote the logic program associated with a TAB. We start by introducing some definitions and lemmas which will be used in the proofs. Let $P_{(t\leq\bar{t})}$ be the program P where for each temporal variable t appearing in a rule, we add the gap-order constraint $t \leq \bar{t}$ to the body of the rule, where $\bar{t}$ is a temporal constant. $G_{P_{(t\leq\bar{t})}}$ is its ground stable model, whereas $M_{P_{(t\leq\bar{t})}}$ denotes any non-ground representation of $G_{P_{(t\leq\bar{t})}}$. The following lemma says that if we are interested in the validity of authorizations at a given instant, we can ignore any ground rule involving instants greater than the given one.

Lemma 1. *For any instant* $\bar{t}$, $valid(\bar{t},\mathtt{A}) \in G_P$ *iff* $valid(\bar{t},\mathtt{A}) \in G_{P_{(t\leq\bar{t})}}$.

Sketch of proof. For each type of rule with $valid(\bar{t},\mathtt{A})$ in the head, no literal in the body contains a temporal variable that can take values greater than $\bar{t}$ for the same rule instance. □

Let P' be the subset of all instantiated (ground) rules in P obtained dropping any rule with a trivially false atom in the body (e.g. $2 \leq 1 \leq 3$ or $3 \equiv_2 4$). Similarly to $P_{(t\leq\bar{t})}$, we can define $P'_{(t\leq\bar{t})}$, obtaining a finite set of rules where each temporal variable has been substituted by a constant less than or equal to $\bar{t}$. Consider now the program $P''_{(t\leq\bar{t})}$ obtained from $P'_{(t\leq\bar{t})}$ by the following substitutions:

a). each literal $\mathbf{not}(once\_not\_valid^f(\mathtt{t}_b,\mathtt{t}_k,\mathtt{P},\mathcal{A}))$ appearing in the body of a rule is replaced by the set of literals $\{valid^f(t',\mathcal{A}) \mid \mathtt{t}_b \leq t' < \mathtt{t}_k \wedge t' \in \Pi(\mathtt{P})\}$;

b). each literal $\mathbf{not}(denied(\mathtt{t}_k,\mathtt{s},\mathtt{o},\mathtt{m}))$ appearing in the body of a rule is replaced by the set of literals
$\{\mathbf{not}(valid(\mathtt{t}_k,\mathtt{A}) \mid \mathtt{A} = (\mathtt{s},\mathtt{o},\mathtt{m},-,\mathtt{g})$ appears in TAB$\}$;

c). each rule r containing the literal $once\_valid^f(\mathtt{t}_b,\mathtt{t}_k,\mathtt{P},\mathcal{A})$ in its body is replaced by the set of rules obtained by substituting $once\_valid^f(\mathtt{t}_b,\mathtt{t}_k,\mathtt{P},\mathcal{A})$ in r with the elements in the set $\{valid^f(t',\mathcal{A}) \mid \mathtt{t}_b \leq t' \leq \mathtt{t}_k \wedge t' \in \Pi(\mathtt{P})\}$;

d). each rule r containing the literal $valid^f(\mathtt{t}_k,\mathcal{A})$, with $\mathcal{A} = \mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$, and $\mathtt{C}_i = \bigwedge_{j=1}^{k} \neg\mathtt{A}_{ji} \wedge \bigwedge_{l=k+1}^{m} \mathtt{A}_{li}$, for $k \in [0,m]$, $m,n \in Z^+$, and $i = 1,\ldots,n$, in its body is replaced by n rules obtained by substituting $valid^f(\mathtt{t}_k,\mathcal{A})$ in r with the conjunction of literals obtained by replacing each $\neg\mathtt{A}_{ji} \in \mathtt{C}_i$ with $\mathbf{not}(valid(\mathtt{t}_k,\mathtt{A}_{ji}))$ and each $\mathtt{A}_{li} \in \mathtt{C}_i$ with $valid(\mathtt{t}_k,\mathtt{A}_{li})$, for each $i = 1,\ldots,n$.

e). each rule r containing the literal $\textsc{cnstr}(\mathtt{P},\mathtt{t}_k)$ in its body is replaced by the set of rules obtained by substituting $\textsc{cnstr}(\mathtt{P},\mathtt{t}_k)$ in r with the elements in the set $\{\mathtt{t}_k \equiv_{periodicity(\mathtt{P})} y \mid (\textsc{cnstr}(\mathtt{P},\mathtt{t}_k) \leftarrow \mathtt{t}_k \equiv_{periodicity(\mathtt{P})} y, t \leq \bar{t}) \in P'_{(t\leq\bar{t})}\}$.

f). any rule having $valid^f()$, $once\_valid^f()$, $once\_not\_valid^f()$, $denied()$ and $\textsc{cnstr}()$ as head is deleted.

Since we have shown that any symbolic periodic expression can be translated into an equivalent finite set of simple periodicity constraints, and we are considering rule instances where t is bound, the resulting set of rules in $P''_{(t\leq\bar{t})}$ is finite. Note that all the rules in $P''_{(t\leq\bar{t})}$ have predicate $valid()$ as head. The following lemma formally states the equivalence between $P''_{(t\leq\bar{t})}$ and $P_{(t\leq\bar{t})}$.

Lemma 2. *For any instant* $\bar{t}$, $valid(\bar{t},\mathtt{A}) \in G_{P_{(t\leq\bar{t})}}$ *iff* $valid(\bar{t},\mathtt{A}) \in G_{P''_{(t\leq\bar{t})}}$.

Sketch of proof. Considering the above transformation it is easy to verify that each atom $valid()$ can be derived using a rule in P' iff it can be derived using the corresponding rule in P''. Then, it is sufficient to remind that $G_{P'_{(t\leq\bar{t})}}$ is equal to $G_{P_{(t\leq\bar{t})}}$. □

In the following, we denote with C_i (Configuration i) the valid authorizations at instants contained in the i-th maximum period after $\overline{\mathtt{t}}_{max}$, where $\overline{\mathtt{t}}_{max}$ is the maximum finite constant in TAB. Formally, $C_i = \{valid(t, \mathtt{A}) \mid valid(t, \mathtt{A}) \in G_{P''_{(t \le \overline{\mathtt{t}}_{max} + i \cdot \mathtt{P}_{max})}}$ and $\overline{\mathtt{t}}_{max} + (i-1) \cdot \mathtt{P}_{max} < t \le \overline{\mathtt{t}}_{max} + i \cdot \mathtt{P}_{max}\}$. It is easy to verify that C_i is a subset of $G_{P''_{(t \le \overline{\mathtt{t}}_{max} + j \cdot \mathtt{P}_{max})}}$, for each $j \ge i$. We define $C_i \stackrel{\rightarrow}{=} C_j$ as true if for each atom in C_i a corresponding one (with the same authorization) can be found in C_j by "shifting" the temporal instant by the value $(j-i) \cdot \mathtt{P}_{max}$, and vice versa.

Lemma 3. *There exists an integer $\bar{k}$ such that $C_k \stackrel{\rightarrow}{=} C_{\bar{k}}$ for each $k > \bar{k}$. The integer $\bar{k}$ is limited by one plus the number of* ASLONGAS *and* UPON *rules in TAB having an unbound associated interval.*

Proof To prove the thesis it is sufficient to consider the derivation of atoms $valid()$ at instants greater than $\overline{\mathtt{t}}_{max}$. Since any derivation of atoms $valid()$ with a constant greater than $\overline{\mathtt{t}}_{max}$ is possible only through a rule with an unbound associated interval (a constraint $\mathtt{t}_b \le t$ with no upper bound), we only consider authorizations and rules in TAB with an unbound associated interval. The proof is articulated in three main steps:

(1) $C_i \stackrel{\rightarrow}{\ne} C_{i+1}$ implies that there exists a pair of rules in $P''_{(t \le \overline{\mathtt{t}}_{max} + (i+1) \cdot \mathtt{P}_{max})}$ instantiating the same rule in TAB for two instants t and t' in the i-th period with $t < t'$, such that (i) either the rule operator is ASLONGAS and the rule with t fires while the one with t' doesn't, or (ii) the operator is UPON and the rule with t does not fire while the one with t' does. Intuitively, this statement says that, in the i-th period, either an ASLONGAS rule "expires" or an UPON rule "triggers". To prove this statement we first note that if only explicit authorizations and WHENEVER rules are present in TAB, then $C_i \stackrel{\rightarrow}{=} C_{i+1}$.
Hence, $C_i \stackrel{\rightarrow}{\ne} C_{i+1}$ implies the existence of a pair of rules in $P''_{(t \le \overline{\mathtt{t}}_{max} + (i+1) \cdot \mathtt{P}_{max})}$, corresponding to the same ASLONGAS rule R_a or to the same UPON rule R_u, and of an authorization A such that the rules derive $valid(t, \mathtt{A})$ and not $valid(t + \mathtt{P}_{max}, \mathtt{A})$, or viceversa. Consider first the ASLONGAS case with $R_a =$ `[begin, ` ∞`]`, P, A ASLONGAS $\mathcal{A}$, and let t' with $t < t' \le t + \mathtt{P}_{max}$ be the first instant, satisfying the periodicity constraint, for which a rule in $P''_{(t \le \overline{\mathtt{t}}_{max} + (i+1) \cdot \mathtt{P}_{max})}$ corresponding to R_a cannot derive $valid(t', \mathtt{A})$. Thus, a literal L (corresponding to an element of the $\mathcal{A}$ expression) in the body of the rule is false for time t'. t' could be in the i-th period in which case the above statement would be proved, or in the $(i+1)$-th period. In this last case, since it is the first instant, we have that literal L is true for $t' - \mathtt{P}_{max}$ and false for t'. Then, we can recursively apply the same reasoning for L as for $valid(t, \mathtt{A})$. It is easily seen that, eventually, either we find an ASLONGAS rule that "expires" in the i-th period, or we find a literal derived by an UPON rule.
A similar reasoning can be done for the UPON rule case, i.e., we consider the first instant t' for which the rules in $P''_{(t \le \overline{\mathtt{t}}_{max} + (i+1) \cdot \mathtt{P}_{max})}$ corresponding to R_u *can* derive $valid(t', \mathtt{A})$. Eventually, either we find an UPON rule that "triggers" in the i-th period, or we find a literal derived by an ASLONGAS rule. Hence, the statement in item 1 holds.

(2) Given two pairs (C_i, C_{i+1}) and (C_k, C_{k+1}) such that $C_i \stackrel{\rightarrow}{\ne} C_{i+1}$ and $C_k \stackrel{\rightarrow}{\ne} C_{k+1}$, the rules expiring (triggering, resp.) in the i-th period and in the k-th period, as stated above, correspond to different ASLONGAS (UPON resp.) derivation rules in TAB. Indeed, consider the case of ASLONGAS. We assume, without loss of generality, that $i < k$. Let R_a be one of the ASLONGAS derivation rules expiring in the i-th period, and let $R_{t'}$ be the rule in $P''_{(t \le \overline{\mathtt{t}}_{max} + (k+1) \cdot \mathtt{P}_{max})}$ corresponding to R_a for the first

instant t' in the i-th period (as chosen above), such that $R_{t'}$ cannot be fired to derive $valid(t', \mathtt{A})$. Hence, at least one of the literals in $R_{t'}$ is false. We know that the literals corresponding to temporal constraints in $R_{t'}$ evaluate to true. It is easily seen from the semantics of ASLONGAS that all the other literals will be present in all $R_{t''}$ with $t'' \geq t'$ and t'' satisfying the periodicity constraint of R_a. Suppose, by contradiction, that a rule $R_{\overline{t}}$ expires in the $(k+1)$-th period, i.e., a rule in $P''_{(t \leq \overline{\mathtt{t}}_{max}+(k+1)\cdot \mathtt{P}_{max})}$ corresponding to the same R_a in TAB with $\overline{t}$ an instant in the k-th period. Then, all literals in $R_{\overline{t}}$ would have to evaluate to true, including those that evaluated to false in $R_{t'}$. This is clearly a contradiction. The arguments for UPON are similar and we omit them. This proof immediately leads to the result that the maximum number of "different" C_is is limited by one plus the maximum number of ASLONGAS and UPON rules in TAB. Indeed, each rule can be responsible for (at most) one relation $C_i \not\overset{\rightarrow}{=} C_{i+1}$.

(3) $C_i \overset{\rightarrow}{=} C_{i+1}$ implies $C_i \overset{\rightarrow}{=} C_j$ for each $j > i$. To prove this statement it is sufficient to show that $C_i \overset{\rightarrow}{=} C_{i+1}$ implies $C_{i+1} \overset{\rightarrow}{=} C_{i+2}$. Suppose by contradiction that $C_i \overset{\rightarrow}{=} C_{i+1}$ but $C_{i+1} \not\overset{\rightarrow}{=} C_{i+2}$. By the item 1 above, this means that there exists a pair of rules in $P''_{(t \leq \overline{\mathtt{t}}_{max}+(i+2)\cdot \mathtt{P}_{max})}$ instantiating the same rule in TAB for two instants t and t' in the $(i+1)$-th period with $t < t'$, such that (i) either the rule operator is ASLONGAS and the rule with t fires while the one with t' doesn't, or (ii) the operator is UPON and the rule with t does not fire while the one with t' does. Consider case (i), and let $valid(t, \mathtt{A})$ be the atom derived by the rule. Hence, $valid(t, \mathtt{A}) \in C_{i+1}$ and $valid(t', \mathtt{A}) \notin C_{i+1}$. However, since $C_i \overset{\rightarrow}{=} C_{i+1}$, then $valid(t' - \mathtt{P}_{max}, \mathtt{A}) \notin C_i$. This means that rule R_a expired in the i-th period and this contradicts the fact that the same rule can derive $valid(t, \mathtt{A})$ in the $(i+1)$-th period. Consider case (ii), and let $valid(t', \mathtt{A})$ be the atom derived by the rule. Hence, $valid(t', \mathtt{A}) \in C_{i+1}$ and $valid(t, \mathtt{A}) \notin C_{i+1}$. However, since $C_i \overset{\rightarrow}{=} C_{i+1}$, then $valid(t' - \mathtt{P}_{max}, \mathtt{A}) \in C_i$. This means that rule R_u triggered in the i-th period and this contradicts the fact that the same rule cannot derive $valid(t, \mathtt{A})$ in the $(i+1)$-th period.

From items 2 and 3, we can easily conclude the thesis. □

In the following, we denote with `max-time` the constant $\overline{t}_{max} + \overline{k} \cdot \mathtt{P}_{max}$, where $\overline{k}$ is the finite constant defined in Lemma 3. This constant will be used in many of the following proofs.

The next two lemmas are essential steps toward the proofs of Theorems 1,2, and 3.

LEMMA 4. *Let $\mathtt{A}_m$ be an authorization appearing in* TAB_{CNS}*. Then, either the CSD algorithm returns FALSE or authorization $\mathtt{A}_m$ is in one and only one level for each time instant between $\mathtt{t}_{min}$ and* `max-time`*. Formally, $\forall \mathtt{A}_m, t$ ($\mathtt{A}_m$ appears in TAB_{CNS} and $\mathtt{t}_{min} \leq t \leq$* `max-time`*) $\Rightarrow \exists\,!\, L_i$ such that $\langle \mathtt{A}_m, \Xi_{m,i}\rangle \in L_i$ and t satisfies $\Xi_{m,i}$.*

PROOF Suppose that the algorithm does not return FALSE. After execution of step 4, each authorization $\mathtt{A}_m$ in TAB_{CNS} appears in level L_1 for each t such that $\mathtt{t}_{min} \leq t \leq$ `max-time`. Any change in levels, made either directly in step 5.1 or through a call of procedure move in steps 5.2 or 5.3, can be always reduced to a pair of delete-add operations: the first deletes $\langle \mathtt{A}_m, \Xi\rangle$ from some level l, the second add $\langle \mathtt{A}_m, \Xi\rangle$ at some level k. Deleting $\langle \mathtt{A}_m, \Xi\rangle$ from level l means updating $\Xi_{m,l}$ to be $\Xi_{m,l} \wedge^* \neg^* \Xi$. Adding $\langle \mathtt{A}_m, \Xi\rangle$ to level L_k means updating $\Xi_{m,k}$ to be $\Xi_{m,k} \cup \Xi$. Consider a t such that t satisfies $\Xi_{m,l}$ before the execution of such a pair of delete-add operations. Suppose t satisfies Ξ. Then, after execution of these operations, t does not satisfy $\Xi_{m,l}$ and t satisfies $\Xi_{m,k}$. Suppose t does not satisfy Ξ. Then, after execution of these operations, t satisfies $\Xi_{m,l}$ and t does not satisfy $\Xi_{m,k}$. Hence, after these operations $\langle \mathtt{A}_m, t\rangle$ still appears in exactly one level. □

Given Lemma 4, in the following, we refer to the level i such that t satisfies $\Xi_{m,i}$ as $L(\mathtt{A}_m[t])$.

LEMMA 5. *Let $\langle x, \Xi_{\mathtt{R}} \rangle$ be an element in TAB$_{CNS}$ such that $x = \mathtt{A}_m \ \langle \textsc{op} \rangle \mathcal{A}$, and let $\mathtt{A}_n$ be an authorization appearing in $\mathcal{A}$. Then either, the CSD algorithm returns FALSE or the following implications hold:*

—*$\forall t$ satisfying $\Xi_{\mathtt{R}}$: $L(\mathtt{A}_m[t]) \geq L(\mathtt{A}_n[t])$. The disequality is strong ($>$) if $\mathtt{A}_n$ appears preceded by $\neg$ in $\mathcal{A}$;*

—*if* OP = PASTOP, *then:*
$\forall t, t'$ satisfying $\Xi_{\mathtt{R}}, t < t'$: $L(\mathtt{A}_m[t']) \geq L(\mathtt{A}_n[t])$. The disequality is strong ($>$) if either $\mathtt{A}_n$ appears preceded by a $\neg$ in $\mathcal{A}$ or OP = ASLONGAS.

PROOF Suppose the algorithm does not return FALSE. Then `top-level` $\leq$ `max-level` and the algorithm has terminated because there were no changes in the last cycle of the **repeat-until** statement in step 5. Since all the elements in TAB$_{CNS}$ are evaluated in each **repeat-until** cycle, element $\langle x, \Xi_{\mathtt{R}} \rangle$ has also been evaluated. We prove the lemma by proving that: *i)* after each evaluation of $\langle x, \Xi_{\mathtt{R}} \rangle$ by the algorithm, the implications held and *ii)* no evaluations of the other elements in TAB$_{CNS}$ during the last iteration of the algorithm can have changed the property in *i)*. The proof is based on the fact, which can be trivially proven, that after the execution of move($\mathtt{A}_h$,Ξ,lev), $L(\mathtt{A}_h[t]) \geq lev$ for all t satisfying Ξ.
i) Suppose OP = WHENEVER. Consider an instant t satisfying $\Xi_{\mathtt{R}}$ and let l be the level such that $l = L(\mathtt{A}_n[t])$. Then t satisfies $\Xi = \Xi_{\mathtt{R}} \wedge^* \Xi_{n,l}$. Hence, according to the call made to procedure move(), after execution of step 5.2, $L(\mathtt{A}_m[t]) = l{+}1 > l = L(\mathtt{A}_n[t])$ if $\mathtt{A}_n$ appears in $\mathcal{A}$ preceded by $\neg$, and $L(\mathtt{A}_m[t]) = l = L(\mathtt{A}_n[t])$ otherwise. Hence the implications above are satisfied. Suppose OP = $\langle$PASTOP$\rangle$. Let $PC_1, \ldots PC_k$ be the periodicity constraints in $\Xi_{\mathtt{R}}$. Consider t, t' satisfying $\Xi_{\mathtt{R}}$ such that $t < t'$ and let l be the level such that $l = L(\mathtt{A}_n[t])$. Then there exists a $\mathtt{t}_l \leq t$ such that $\mathtt{t}_l$ satisfies $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l}$. Hence, according to the call made to procedure move(), after execution of step 5.3, $L(\mathtt{A}_m[t]) = L(\mathtt{A}_m[t']) = l{+}1 > l = L(\mathtt{A}_n[t])$ if $\mathtt{A}_n$ appears in $\mathcal{A}$ preceded by $\neg$. Otherwise, $L(\mathtt{A}_m[t]) = L(\mathtt{A}_m[t']) = l = L(\mathtt{A}_n[t])$ if OP = UPON, whereas $L(\mathtt{A}_m[t]) = L(\mathtt{A}_m[t']) = l + 1 > l = L(\mathtt{A}_n[t])$ for $\mathtt{t}_l < t$ and $L(\mathtt{A}_m[t_l]) = l = L(\mathtt{A}_n[\mathtt{t}_l])$ if OP = ASLONGAS. Hence the implications above are satisfied.
ii) Levels are changed either directly in substep 5.1 or, through procedure move, in substeps 5.2 and 5.3. In substep 5.1, pairs $\langle \mathtt{A}_m, \Xi \rangle$ are possibly deleted from a given level h and added to level $l + 1$, with $h < l + 1$. Procedure move() deletes a pair $\langle \mathtt{A}_m, \Xi \rangle$ from a level i and adds it to level lev, with $i \leq lev$. Then, pairs $\langle \mathtt{A}_m, \Xi \rangle$ can be moved only to higher levels, i.e., the level of an authorization for an instant is never lowered. Then, the property in *i)* can be changed, in the last iteration of the algorithm, only by raising the level of $\mathtt{A}_n[t]$ after the evaluation of $\langle x, \Xi_{\mathtt{R}} \rangle$. But this cannot be since it would contradict the assumption that no changes have been made in the last cycle of the **repeat-until** statement in step 5. □

Before proving Theorem 1 it is important to note that TAB$_{CNS}$ is only a compact representation of the authorizations and rules in TAB. Then, the levels in which the authorizations appearing in TAB$_{CNS}$ are partitioned by the CSD algorithm apply also to the authorizations appearing in TAB. Given this, in the following, we consider TAB and TAB$_{CNS}$ as equivalent.

PROOF of Theorem 1.

Theorem 1 descends directly from Lemma 5 and from the fact that, according to the definition of $\hookrightarrow$ and $\overset{+}{\hookrightarrow}$: $t \leq t'$ and

—if $t = t'$ then there exists a rule $\mathtt{R}$=([begin,end], P, $\mathtt{A}_m$ $\langle$OP$\rangle\mathcal{A}$) in TAB with $t \in \{[\mathtt{t_b}, \mathtt{t_e}]\} \cap \Pi(\mathtt{P})$ and $\mathtt{A}_n$ appears in $\mathcal{A}$. Moreover $\mathtt{A}_n$ appears preceded by $\neg$ in $\mathcal{A}$ if $\mathtt{A}_n$ strictly affects $\mathtt{A}_m$;

—if $t < t'$ then there exists a rule $\mathtt{R}$=([begin,end], P, $\mathtt{A}_m$ $\langle$PASTOP$\rangle\mathcal{A}$) in TAB with $t \in \{[\mathtt{t_b}, \mathtt{t_e}]\} \cap \Pi(\mathtt{P})$, and $\mathtt{A}_n$ appears in $\mathcal{A}$. Moreover, if $\mathtt{A}_n$ strictly affects $\mathtt{A}_m$, then either OP = ASLONGAS or $\mathtt{A}_n$ appears preceded by $\neg$ in $\mathcal{A}$.

□

PROOF of Theorem 2.

We need to prove that either the algorithm returns FALSE or for each pair of authorizations $\mathtt{A}_m$ and $\mathtt{A}_n$ appearing in TAB such that $\mathtt{s}(\mathtt{A}_m) = \mathtt{s}(\mathtt{A}_n), \mathtt{o}(\mathtt{A}_m) = \mathtt{o}(\mathtt{A}_n), \mathtt{m}(\mathtt{A}_m) = \mathtt{m}(\mathtt{A}_n), \mathtt{pn}(\mathtt{A}_m) =$'+', $\mathtt{pn}(\mathtt{A}_n) =$'−', at the end of the execution $L(\mathtt{A}_m[t]) > L(\mathtt{A}_n[t])$ for each instant t such that $\mathtt{t}_{min} \leq t \leq$ `max-time`. Suppose that the algorithm does not return FALSE. Then, `top-level` $\leq$ `max-level` and the algorithm has terminated because there was no change in the last cycle of the `repeat-until` statement of step 5. Since all negative authorizations are considered at each iteration of step 5 (substep 5.1), $\mathtt{A}_n$ has also been considered. We prove the theorem by proving that: *i)* after the evaluation of $\mathtt{A}_n$ by the algorithm, $L(\mathtt{A}_m[t]) > L(\mathtt{A}_n[t])$ for any t such that $\mathtt{t}_{min} \leq t \leq$ `max-time`, and *ii)* nothing in the execution of the last iteration of the algorithm can have changed the property in step *i)*.

i) Consider a t such that $\mathtt{t}_{min} \leq t \leq$ `max-time` and let l be the level such that $l = L(\mathtt{A}_n[t])$. Then t satisfies Ξ_i for some set S_i at level l. Since each authorization appears at some level for every instant between $\mathtt{t}_{min}$ and `max-time`, t satisfies $\Xi_{m,h}$, for some level h. If $h > l$ the implication is trivially satisfied, since levels above l are not changed by substep 5.1. Suppose $h \leq l$. Then, t satisfies $\Xi = \Xi_{m,h} \wedge^* \Xi_i$. Hence, after the execution of the delete-add operations, $L(\mathtt{A}_m[t]) = l + 1 > l = L(\mathtt{A}_n[t])$.

ii) The proof is the same as that of point *ii)* in Lemma 5. □

PROOF of Theorem 3.

i) The only step that may introduce a loop in the algorithm is step 5. Step 5 is a **repeat-until** cycle composed of three substeps. These substeps iterate either considering all the levels down to level 1 or considering all the elements in a finite set. Hence, they necessarily terminate. Consider the outer **repeat-until** cycle. Suppose that it never stops. Then, at each iteration, the statement in the **repeat-until** evaluates to true, i.e., changes have occurred in some level and `top-level` $\leq$ `max-level`. Changes can only move authorizations to higher levels and every time a new level is created `top-level` is increased by 1. If changes keep occurring at every iteration, eventually `top-level` will reach `max-level`, and the **repeat-until** cycle terminates, which contradicts the assumption. Hence, the algorithm terminates.

ii) We first prove that if the TAB contains a critical set then the algorithm returns FALSE. Suppose that a critical set exists in the TAB, i.e., a rule $\mathtt{A}_m$ and an instant t exists such that $\mathtt{A}_m[t] > \mathtt{A}_m[t]$. Then, at least one of the conditions in Definition 13 is satisfied. (Note that the algorithm considers only the time instants up to `max-time`. However, there is no need to consider all time instants up to ∞. If a dependency between two authorizations exists for a time instant after `max-time` it also exists for a time instant lower than or equal to `max-time`.)

Suppose that the first condition is satisfied. A sequence $\mathtt{A}_m[t]$=$\mathtt{A}_1[t], \ldots, \mathtt{A}_{k-1}[t], \mathtt{A}_k[t]$=$\mathtt{A}_m[t]$ exists such that each element in the sequence affects the successor and there exists one that strictly affects it. By Theorem 1, each $\hookrightarrow$ relation implies a $\leq$ relationships on the resulting levels and each $\overset{+}{\hookrightarrow}$ relation implies a $<$ relationships among levels. Then, we should have $L(\mathtt{A}_m[t]) < L(\mathtt{A}_m[t])$, which cannot be since, according to Lemma 4 $L(\mathtt{A}_m[t])$ is unique.

Suppose that the second condition is satisfied. Two sequences $\mathtt{A}_m[t]$=$\mathtt{A}_1[t], \ldots, \mathtt{A}_l^-[t]$ and $\mathtt{A}_{l+1}^+[t], \ldots$ $\mathtt{A}_k[t]$= $\mathtt{A}_m[t]$ exist such that $\mathtt{s}(\mathtt{A}_l^-)$=$\mathtt{s}(\mathtt{A}_{l+1}^+)$, $\mathtt{o}(\mathtt{A}_l^-)$=$\mathtt{o}(\mathtt{A}_{l+1}^+)$, and $\mathtt{m}(\mathtt{A}_l^-)$=$\mathtt{m}(\mathtt{A}_{l+1}^+)$ and each element in each sequence affects the successor and there exists one that strictly affects it. Note that the time instant must be the same along the two sequences since, by definition of $\hookrightarrow$, and $\overset{+}{\hookrightarrow}$ the time instant on the right hand side of the relationships is always either greater than or equal to the time instant on the left hand side and by assumption $\mathtt{A}_m[t]$ initiates the first sequence and terminates the second. Then, from Theorem 1, $L(\mathtt{A}_m[t]) < L(\mathtt{A}_l[t])$ and $L(\mathtt{A}_{l+1}[t]) < L(\mathtt{A}_m[t])$. Moreover, from Theorem 2 $L(\mathtt{A}_l[t]) < L(\mathtt{A}_{l+1}[t])$. Hence, again we should obtain $L(\mathtt{A}_m[t]) < L(\mathtt{A}_m[t])$, which cannot be because of the uniqueness of $L(\mathtt{A}_m[t])$ (Lemma 4).

The proof for the last case of Definition 13 descends directly from the previous ones.

We now prove that if the algorithm returns FALSE, then a critical set exists in TAB. Suppose the algorithm returns FALSE. Then, `top-level` $>$ `max-level`. `top-level` is increased by 1 by the algorithm whenever a pair $\langle \mathtt{A}_m, \Xi\rangle$ needs to be put at a level $l+1 >$ `top-level` (substep 5.1 of the algorithm or procedure move() called in substeps 5.2 and 5.3). We first show that the request to delete $\langle \mathtt{A}_m, \Xi\rangle$ from a given level and put it to level $l+1$ implies that, for each instant t' satisfying Ξ either:

1) pair $\langle \mathtt{A}_n, \Xi_{n,l}\rangle \in L_l$ exists such that $\mathtt{s}(\mathtt{A}_m)$=$\mathtt{s}(\mathtt{A}_n)$, $\mathtt{o}(\mathtt{A}_m)$=$\mathtt{o}(\mathtt{A}_n)$, $\mathtt{m}(\mathtt{A}_m)$=$\mathtt{m}(\mathtt{A}_n)$, $\mathtt{pn}(\mathtt{A}_n)$= = '$-$', $\mathtt{pn}(\mathtt{A}_m)$='+', and t' satisfies $\Xi_{n,l}$, or

2) pair $\langle \mathtt{A}_n, \Xi_{n,l}\rangle \in L_l$ exists such that $\mathtt{A}_n[t] \overset{+}{\hookrightarrow} \mathtt{A}_m[t']$ for some t satisfying $\Xi_{n,l}$.

Suppose that the level is changed in substep 5.1. Then, t' satisfies $\Xi_{m,h} \wedge^* \Xi_i$. According to the definition of Ξ_i, there exists $\Xi_{n,l}$ such that t' satisfies $\Xi_{n,l}$, $\langle \mathtt{A}_n, \Xi_{n,l}\rangle \in S_i$, and $\mathtt{pn}(\mathtt{A}_n)$ ='$-$'. Moreover, since $\mathtt{A}_m$ satisfies the condition in the **for** cycle, $\mathtt{s}(\mathtt{A}_m)$=$\mathtt{s}(\mathtt{A}_n)$, $\mathtt{o}(\mathtt{A}_m)$=$\mathtt{o}(\mathtt{A}_n)$, and $\mathtt{m}(\mathtt{A}_n)$=$\mathtt{m}(\mathtt{A}_n)$. Then, condition 1 is verified.

Suppose that the level is changed by a call of procedure 'move()' made in substep 5.2. Then, t' satisfies $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l}$ and $\Xi_{\mathtt{R}}$ and n are such that an element $\langle \mathtt{A}_m \text{WHENEVER}\mathcal{A}, \Xi_{\mathtt{R}}\rangle$ was evaluated and $\mathtt{A}_n$ appeared in $\mathcal{A}$ preceded by $\neg$. Then, condition 2 is satisfied with $t = t'$.

Suppose that the level is changed in step 5.3 by the evaluation of an element $\langle \mathtt{A}_m\langle\text{PASTOP}\rangle\mathcal{A}, \Xi_{\mathtt{R}}\rangle$. Let $PC_1, \ldots, PC_k$ be the periodicity constraints in $\Xi_{\mathtt{R}}$. Procedure 'move()' is called to move $\mathtt{A}_m$ to level $l+1$ if there exists an authorization $\mathtt{A}_n$ such that $\langle \mathtt{A}_n, \Xi_{n,l}\rangle \in L_l$, $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l} \neq \emptyset$ and t' satisfies $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l}$, and either $\mathtt{A}_n$ appears in $\mathcal{A}$ preceded by $\neg$, or OP $=$ ASLONGAS. In both cases, condition 2 is satisfied with $t = \mathtt{t}_l$, where $\mathtt{t}_l$ is the first instant satisfying $\Xi_{\mathtt{R}} \wedge^* \Xi_{n,l}$.

Since `top-level` $>$ `max-level`, `top-level` has been increased by 1 at least `max-level` times. This means that `max-level` times a relationship of type 1 or 2 above, which required the creation of a new level, has been found by the algorithm. Suppose all relationships were of type 2. Then, a chain of `max-level` $\overset{+}{\hookrightarrow}$ relations $\mathtt{A}_u[t] \overset{+}{\hookrightarrow} \ldots \overset{+}{\hookrightarrow} \mathtt{A}_v[t'']$ exists. Since the total number of different elements $\mathtt{A}_x[t_y]$ is `max-level`, an authorization $\mathtt{A}_m$ and an instant t exists such that $\mathtt{A}_m[t]$ appears twice in the chain. Then, according to Definition 14, a critical set exists, represented by the sub-chain starting from $\mathtt{A}_m[t]$ and ending to the other $\mathtt{A}_m[t]$.

Suppose that k relationships, among the `max-level` found which caused the creation of a new level, were of type 1. Then, k chains of $\overset{+}{\hookrightarrow}$ relationships exist where the last element $\mathtt{A}_v[t'']$ in chain h and the first element $\mathtt{A}_u[t'']$ in chain $h+1$ are such that $\mathtt{s}(\mathtt{A}_v)$=$\mathtt{s}(\mathtt{A}_u)$, $\mathtt{o}(\mathtt{A}_v)$=$\mathtt{o}(\mathtt{A}_u)$, $\mathtt{m}(\mathtt{A}_v)$=$\mathtt{m}(\mathtt{A}_u)$, $\mathtt{pn}(\mathtt{A}_v)$='$-$', and $\mathtt{pn}(\mathtt{A}_u)$='+'. Then, again, since the number of different elements $\mathtt{A}_x[t_y]$ is `max-level`, an authorization $\mathtt{A}_m$ and an instant t exists such that $\mathtt{A}_m[t]$ appears twice either in the same chain, or in two different chains. In the first case, again a critical set exists because of the sub-chain of $\overset{+}{\hookrightarrow}$ relationships going from $\mathtt{A}_m[t]$ to the other $\mathtt{A}_m[t]$. In the second case a critical set exists represented by the subsequence of chains going from the first $\mathtt{A}_m[t]$ that appears to the other instance of $\mathtt{A}_m[t]$, according to Definition 14. □

Proof of Theorem 4.

We first show that the absence of critical sets implies the local stratification of P'_{TAB} (as defined just after Lemma 1).

Lemma 6. *Given a* TAB *with no critical sets, the logic program* P'_{TAB} *is locally stratified.*

Proof Since TAB has no critical set, the CSD algorithm gives a partition in a finite number of *levels* of the pairs $\langle \mathtt{A}, t\rangle$ where A is an authorization in TAB and t an instant in $\{t_{min}, \ldots, \mathtt{max\text{-}time}\}$. Using this partition, we define a partition $H_1, \ldots, H_\alpha$ of the (ground) Herbrand base of P'_{TAB}, where α is a countable ordinal. It is obtained applying the following steps in the given order:

(1) each atom of the form $t \leq t' \leq t''$, $t \leq t' < t''$, $t \equiv_k c$, and $\mathrm{CNSTR}(\mathtt{P}, t)$ is in H_1;

(2) if $valid(t,\mathtt{A})$ is in the Herbrand Base, but A does not appear in TAB, then $valid(t,\mathtt{A})$ is in H_1;

(3) each atom $once\_valid^f(t, t, \mathcal{A})$ and $once\_not\_valid^f(t, t, \mathcal{A})$ is in H_1;

(4) each atom $valid(t,\mathtt{A})$ with $t \leq$`max-time` is in H_k if $\langle \mathtt{A}, \Xi\rangle$ with t satisfying Ξ is assigned to level $k/3$ by the CSD algorithm;

(5) each atom $valid(t,\mathtt{A})$ with $t >$`max-time` is in H_k if $valid(r,\mathtt{A})$ is in H_s, where $r = ((t - \bar{\mathtt{t}}_{max})\ \mathbf{MOD}\ \mathtt{P}_{max}) + (\mathtt{max\text{-}time} - \mathtt{P}_{max})$, and $k = s + \mathtt{n\text{-}auth} \cdot \mathtt{P}_{max} \cdot ((t - \bar{\mathtt{t}}_{max})\ \mathbf{DIV}\ \mathtt{P}_{max})$; If $s \leq 3$ then $valid(t,\mathtt{A})$ is in H_s;

(6) each atom $valid^f(t, \mathcal{A})$ is in H_k where $k = \mathbf{max}\{j \mid (valid(t,\mathtt{A}_i) \in H_j \text{ and } \mathtt{A}_i \in \mathcal{A}) \text{ or } (valid(t,\mathtt{A}_i) \in H_{j-1} \text{and} \neg \mathtt{A}_i \in \mathcal{A})\}$;

(7) each atom $once\_valid^f(t'', t, \mathcal{A})$ with $t'' < t$ is in H_k with $k = \mathbf{max}\{j \mid valid^f(t', \mathcal{A}) \in H_j \text{ and } t'' \leq t' \leq t\}$;

(8) each atom $once\_not\_valid^f(t'', t, \mathcal{A})$ with $t'' < t$ is in H_k where $k = 1 + \mathbf{max}\{j \mid valid^f(t', \mathcal{A}) \in H_j \text{ and } t'' \leq t' < t\}$;

(9) each atom $denied(t,\mathtt{s},\mathtt{o},\mathtt{m})$ is in H_k where $k = \mathbf{max}\{j \mid valid(t,\mathtt{s},\mathtt{o},\mathtt{m},-,\mathrm{g}) \in H_j$ for any grantor $g\}$;

(10) any other atom is in H_1.

By definition, P'_{TAB} is locally stratified if, for each of its rules $C \leftarrow L_1, \ldots, L_n, \mathbf{not} B_1, \ldots,$ $\mathbf{not}\ B_m$, with $C \in H_k$, we have $L_i \in \bigcup_{j \leq k} H_j$ for $i = 1, \ldots, n$ and $B_l \in \bigcup_{j<k} H_j$ for $l = 1, \ldots, m$. This condition is satisfied considering the partition shown above. For brevity, here we report the proof for only two types of rules in P'_{TAB}:

a) Consider a rule corresponding to a single positive authorization in TAB. We have $C = valid(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1)$, $L_1 = \mathtt{t}_b \leq t \leq \mathtt{t}_e$, and $L_2 = \mathrm{CNSTR}(\mathtt{P}, t)$. By step 1, $L_1, L_2 \in H_1$ while $k \geq 1$. $B_1 = denied(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1)$ and, by step 9, $B_1 \in H_w$ with $w = \mathbf{max}\{j \mid valid(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathrm{g}) \in H_j\}$. If $w \neq 1$ and $t \leq$ `max-time`, then, by step 4, $valid(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathrm{g}) \in H_j$ if the pair $\langle(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathrm{g}), \Xi\rangle$ with t satisfying Ξ was assigned to level $j/3$ by the CSD algorithm. Similarly, $C \in H_k$ because of step 4, and because the pair $\langle(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1), \Xi'\rangle$ with t satisfying Ξ' was assigned to level $k/3$ by the CSD algorithm. Theorem 2 ensures that the level of a positive authorization is always greater than that of the corresponding negative authorization for the same instant (if that negative authorization is not in TAB, by step 2, its corresponding $valid()$ is in H_1). Hence, $k/3 > j/3$ and we can easily conclude that $k > w$. If $t >$`max-time` (this implies that $\mathtt{t}_e = \infty$), by step 5, if $C \in H_k$ it means that $valid(\mathrm{r},\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1) \in H_s$ with $r = ((t - \bar{\mathtt{t}}_{max})\ \mathbf{MOD}\ \mathtt{P}_{max}) + (\mathtt{max\text{-}time} - \mathtt{P}_{max})$, and $k = s + \mathtt{n\text{-}auth} \cdot \mathtt{P}_{max} \cdot ((t - \bar{\mathtt{t}}_{max})\ \mathbf{DIV}\ \mathtt{P}_{max})$ if $k > 3$, and $k = s$ otherwise. Considering B_1, the index w of its stratum is given, as shown above, by step 9. In this case ($t >$`max-time`), for any grantor $\mathtt{g}$ such that $valid(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}) \in H_j$, by step 5, $valid(\mathrm{r},\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\ \mathtt{g}) \in H_{s'}$ with $j = s' + \mathtt{n\text{-}auth} \cdot \mathtt{P}_{max} \cdot ((t - \bar{\mathtt{t}}_{max})\ \mathbf{DIV}\ \mathtt{P}_{max})$ if $j > 3$ and $H_j = H_{s'}$ if $j \leq 3$. Since $r \leq$ `max-time`, Theorem 2 can be applied, ensuring that $\frac{s}{3} > \frac{s'}{3}$, and, hence, we have $k > j$. Since, by step 9 w is the maximum among these j's, we conclude $k > w$.

b) Consider now a rule corresponding to an aslongas rule in TAB. If the head of the rule $\mathtt{A}_1$, is a positive authorization, the corresponding clause has a negative literal $denied()$. The proof that its stratum is less than k is identical to what shown for case a) above. If $\mathtt{A}_1$ is negative, $C = valid(t,\mathtt{A}_1)$, $L_1 = \mathtt{t}_b \leq t \leq \mathtt{t}_e$, $L_2 = \textsc{cnstr}(\mathtt{P}, t)$, $L_3 = valid^f(t, \mathcal{A})$, and $B_1 = once\_not\_valid^f(\mathtt{t}_b, t, \mathcal{A})$. C, by the stratification given by the CSD algorithm, and by steps (4) and (5) above, belongs to H_k with $k \geq 3$. By step 1, $L_1, L_2 \in H_1$. It is also easy to show that L_3 is assigned to a stratum equal or below that of C. Let's consider the more critical case of B_1. If $\mathtt{t}_b = t$ then $B_1 \in H_1$ by step 3, while $k \geq 3$. Otherwise ($t > \mathtt{t}_b$), by step 8, $B_1 \in H_s$ where $s = 1 + \mathbf{max}\{m \mid valid^f(t', \mathcal{A}) \in H_m \text{ and } \mathtt{t}_b \leq t' < t\}$. For each $\mathtt{t}_b \leq t' < t$, $valid^f(t', \mathcal{A}) \in H_m$ implies, by step 6, $valid(t',\mathtt{A}_i) \in H_j$ with $j \leq m$ for each $\mathtt{A}_i \in \mathcal{A}$ and $valid(t',\mathtt{A}_l) \in H_r$ with $r < m$ for each $\neg\mathtt{A}_l \in \mathcal{A}$. We now distinguish two cases: (a) $t \leq$ `max-time`, and (b) $t >$ `max-time`. Considering (a), we also have $t' <$ `max-time` and, by step 4, $\langle \mathtt{A}_i, \Xi_i \rangle$ with t' satisfying Ξ_i is assigned to level $j/3$ by the CSD algorithm, and each $\langle \mathtt{A}_l, \Xi_l \rangle$ with t' satisfying Ξ_l is assigned to level $r/3$. Lemma 5 states that in this case, the level assigned by the CSD algorithm to $\langle \mathtt{A}_1, \Xi_1 \rangle$ with t satisfying Ξ_1 is strictly greater than the level of any $\langle \mathtt{A}_i, \Xi_i \rangle$ with $\mathtt{A}_i \in \mathcal{A}$ or $\neg\mathtt{A}_i \in \mathcal{A}$ and t' satisfying Ξ_i for each $\mathtt{t}_b \leq t' < t \leq$ `max-time`. By step 4, $\langle \mathtt{A}_1, \Xi_1 \rangle$ with t satisfying Ξ_1 is assigned to level $k/3$. Hence, by Lemma 5, $k/3 > j/3$ and $k/3 > r/3$ for each j and r considered above. Since $k/3$ and $j/3$ must be natural numbers greater than 1, $k > j+2$ and $k > r+2$. Since m can be at most equal to j or to $r+1$ and $s = 1 + \mathbf{max}\{m \mid valid^f(t', \mathcal{A}) \in H_m \text{ and } \mathtt{t}_b \leq t' < t\}$ we can conclude that $k > s$.

We are left with case (b), i.e., $t >$`max-time`. In this case we should consider step 5 instead of step 4 used above. Thus, if $C \in H_k$, then $valid(z,\mathtt{A}_1)$ is in H_w, where $z = ((t - \bar{\mathtt{t}}_{max})\ \mathbf{MOD}\ \mathtt{P}_{max}) + (\mathtt{max\text{-}time} - \mathtt{P}_{max})$, and $k = w + \mathtt{n\text{-}auth} \cdot \mathtt{P}_{max} \cdot ((t - \bar{\mathtt{t}}_{max})\ \mathbf{DIV}\ \mathtt{P}_{max})$. Similarly, the $valid()$ atoms corresponding to literals in $\mathcal{A}$ are assigned to strata based on the strata assigned to corresponding atoms for instants in the last maximum period. It is easily seen that, Lemma 5 can still be applied, since these values are (like z above) smaller than `max-time`. Then, the relation on the CSD levels imposed by the lemma can be raised to values greater than `max-time` considering how strata for atoms with these values are assigned in step 5.

The proof for the other types of rules is analogous. □

To conclude the proof of Theorem 4, note that if a program is locally stratified, then it has a *well founded model*, and that model is the unique stable model [Gelder et al. 1991]. Since Lemma 6 states that P'_{TAB} is locally stratified, and P_{TAB} has the same stable models as P'_{TAB}, we can conclude that P_{TAB} has a unique ground stable model.

□

Proof of Theorem 5.

By Definition 12, proving point *ii)* of the thesis is equivalent to prove: given $\bar{t} \in Z^+$, for all authorization $\mathtt{A}$, and sets of constraints Ξ, Ω, $(valid(t, \mathtt{A}), \Xi) \in M_P$, with $\bar{t}$ satisfying Ξ, iff $\exists \langle \mathtt{A}, \Omega \rangle$ in the TAB$_{EXT}$ computed by the algorithm, such that $\bar{t}$ satisfies Ω.

We prove point *ii)* by showing that the algorithm computes a finite representation of $G_{P''}$. Note that, the second step of Algorithm 2 consider only instants less than a finite constant. This finite constant represents the maximum value that variable *current_max* can assume during the execution of step 2. *current_max* is initially set equal to $\bar{\mathtt{t}}_{max} + 2 \cdot \mathtt{P}_{max}$ and incremented by $\mathtt{P}_{max}$ at each iteration until either it reaches the value $\bar{\mathtt{t}}_{max} + \hat{k} \cdot \mathtt{P}_{max}$, where $\hat{k}$ is the first positive integer such that $\forall \langle \mathtt{A}, \Omega \rangle$ in TAB$_{EXT}$: $\Omega \wedge^* \{(\{true\}, \{ \bar{\mathtt{t}}_{max} + (\hat{k}-2) \cdot \mathtt{P}_{max} < t \leq \bar{\mathtt{t}}_{max} + (\hat{k}-1) \cdot \mathtt{P}_{max} \})\} \overset{\rightarrow}{=}^* \Omega \wedge^* \{(\{true\}, \{ \bar{\mathtt{t}}_{max} + (\hat{k}-1) \cdot \mathtt{P}_{max} < t \leq \bar{\mathtt{t}}_{max} + \hat{k} \cdot \mathtt{P}_{max} \})\}$, or it reaches the value $\mathtt{max\text{-}time} = \bar{\mathtt{t}}_{max} + \bar{k} \cdot \mathtt{P}_{max}$. In the following, we denote with $\mathtt{max}$ the finite constant $\bar{\mathtt{t}}_{max} + k^* \cdot \mathtt{P}_{max}$, where k^* is the minimum between $\bar{k}$ and $\hat{k}$. Then, the third step of the algorithm extends the result of the second step to infinity. Therefore the thesis can be proved by the following steps: we first prove that the second step of the algorithm computes a finite representation of $G_{P''_{(t \leq \mathtt{max})}}$ (Lemma 7) and thus of $G_{P_{(t \leq \mathtt{max})}}$. Then, we show that the extension performed by the third step of the algorithm computes M_P.

LEMMA 7. *The second step of Algorithm 2 computes* $M_{P''_{(t \leq \mathtt{max})}}$.

SKETCH OF PROOF. In the following we use P'' for $P''_{(t \leq \mathtt{max})}$. It is easy to show that P'' is locally stratified. In the proof of Theorem 4 we have given the local stratification of P'. The stratification of P'' can be simply obtained assigning stratum k to a rule, if its head atom is in stratum k as constructed in the stratification of P'. Empty strata can then be collapsed. In Appendix A we have defined an algorithm, based on a fix-point iteration method, to compute the model of a stratified Datalog$^{\mathbf{not}, \equiv Z, < Z}$ program. Moreover, Theorem 6 guarantees that for Datalog$^{\mathbf{not}, \equiv Z, < Z}$ programs in which gap-order constraints are on a finite subset of the integer the algorithm always terminates returning a non-ground representation of the unique ground model of the program. The method used in Algorithm 3 and the result of Theorem 6 naturally extend to locally stratified Datalog$^{\mathbf{not}, \equiv Z, < Z}$ programs, and therefore to program P'', whose gap-order constraints are defined on the finite set $\{0, \ldots, \mathtt{max}\}$. Algorithm 2 performs exactly the procedure defined by Algorithm 3 trying to consider multiple rules in a single step using operations on constraints. Each iteration of step 2.2 computes $M_{P''_{(t \leq current\_max)}}$. Each iteration of the **repeat-until** cycle in step 2.2 computes the fixed point of derivations for a certain stratum i. Step 2.2.a computes the set of rules applicable at a certain stratum. Note that elements in X_i, where i is the stratum, denote the authorizations/rules in TAB that correspond to clauses of $P''_{(t \leq current\_max)}$ of level i. More precisely, if $\langle x, \Xi \rangle \in X_i$, where $x = \mathtt{A}_m$ or $x = \langle \mathtt{A}_m \langle \text{OP} \rangle \mathcal{A} \rangle$, then the set of clauses corresponding to $\langle x, \Xi \rangle$ in $P''_{(t \leq current\_max)}$ contains at least a clause of level i. This is easily proved since the stratification of P'' is done according to the output of the CSD algorithm, and X_i is computed using the same output. After the execution of step 2.2.a the constraint Θ denotes the set of rules of $P''_{(t \leq current\_max)}$ corresponding to $\langle x, \Xi \rangle$, which are applicable in stratum i: one for each t satisfying Θ.

It is easy to show that for each element in X_i the application of steps 2.2.b and 2.2.c is equivalent to try to fire the equivalent rules in $P''_{(t \leq current\_max)}$ using the same procedure as Algorithm 3. The cases for explicit authorizations and WHENEVER rules are straightforward. Consider an UPON rule, that is, suppose that $\langle x, \Xi \rangle = \langle \mathtt{A}_m \text{ UPON } \mathcal{A}, \Xi \rangle$. For the sake of simplicity, we suppose that $\mathtt{A}_m$ is a negative authorization. The prove for positive authorizations is analogous. Let $\Xi = \{(PC_1, GC), \ldots, (PC_k, GC)\}$ and let $\mathcal{A} = \mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$,

$n \in Z^+$ and $\mathtt{C}_i = \bigwedge_{j=1}^{k} \neg \mathtt{A}_{ji} \wedge \bigwedge_{l=k+1}^{m} \mathtt{A}_{li}$, $k \in [0,m]$, $m \in Z^+$, $i = 1,\ldots,n$. For each conjunct $\mathtt{C}_i$ the set of rules of level i of $P''_{(t \leq current\_max)}$ corresponding to $\langle \mathtt{A}_m$ UPON $\mathcal{A}, \Xi\rangle$ is as follows: $\{valid(\bar{t},\mathtt{A}_m) \leftarrow PC_i/\bar{t}, GC/\bar{t}, \mathbf{not}(valid(\hat{t},\mathtt{A}_{ji})),\ldots,\mathbf{not}(valid(\hat{t},\mathtt{A}_{ki})), valid(\hat{t},\mathtt{A}_{li}),\ldots,valid(\hat{t},\mathtt{A}_{mi}) \mid \bar{t}$ satisfies Θ, $(PC_i, GC) \in \Xi$, $PC_i/\bar{t}$ and $GC/\bar{t}$ do not contain any trivially false constraints, $\hat{t} \leq \bar{t}$ and $\hat{t}$ satisfies $\Xi\}$, where $PC_i/\bar{t}$ (resp. $GC/\bar{t}$) denotes the conjunction of simple periodicity constraints (resp. gap-order constraints) obtained by replacing the unique temporal variable appearing in PC_i (resp. GC) with constant $\bar{t}$. The above set is equivalent to the rule: $valid(t,\mathtt{A}_m) \leftarrow \Xi \wedge^* \Theta$, $\Xi/t' \wedge \{(true, t' \leq t)\}, \mathbf{not}(valid(t',\mathtt{A}_{ji})),\ldots,\mathbf{not}(valid(t',\mathtt{A}_{ki}))$, $valid(t',\mathtt{A}_{li}),\ldots,valid(t',\mathtt{A}_{mi})$, where Ξ/t' is obtained by replacing the unique temporal variable in Ξ with t'. By construction of Θ, the above rule is equivalent to: $valid(t,\mathtt{A}_m) \leftarrow \Theta$, $\Xi/t' \wedge \{(true, t' \leq t)\}, \mathbf{not}(valid(t',\mathtt{A}_{ji})),\ldots,$
$\mathbf{not}(valid(t',\mathtt{A}_{ki}))$, $valid(t',\mathtt{A}_{li}),\ldots,valid(t',\mathtt{A}_{mi})$. Thus the set of rules of level i of $P''_{(t \leq current\_max)}$ corresponding to $\langle \mathtt{A}_m$ UPON $\mathcal{A}, \Xi\rangle$ is: $\{valid(t,\mathtt{A}_m) \leftarrow \Theta, \Xi/t' \wedge \{(true, t' \leq t)\}, \mathbf{not}(valid(t',\mathtt{A}_{ji})),\ldots \mathbf{not}(valid(t',\mathtt{A}_{ki})), valid(t',\mathtt{A}_{li}),\ldots,valid(t',\mathtt{A}_{mi}) \mid i = 1,\ldots,n\}$. Step 2.2.b computes the first instant $\bar{t}$ such that: *i)* $\bar{t}$ satisfies Ξ and *ii)* $\mathcal{A}$ is valid in TAB_{EXT} at time $\bar{t}$. Then it takes the intersection of Θ with $\{(true, t \geq \bar{t})\}$. It is immediate to verify that this is equivalent to evaluate the above set of rules using Algorithm 3. A similar proof can be obtained for ASLONGAS rules. The **repeat-until** cycle in step 2.2 ends when the fixed point for the current stratum is reached. Step 2.2 ends when the fixed point for the upper stratum is reached, that is, when the model of $M_{P''_{(t \leq current\_max)}}$ has been computed. Then, step 2 is iteratively executed till *current-max* reaches either the value `max-time` or the value $\bar{\mathtt{t}}_{max} + \hat{k} \cdot \mathtt{P}_{max}$, that is, till *current_max* reaches the value `max`. Hence, we can conclude the thesis. □

We are now ready to prove the theorem.

i) (Termination) The only step that may introduce an infinite loop is step 2. Step 2 consists of an outer **repeat-until** cycle which is iterated until either variable *success* evaluates to true or *current_max* reaches the value `max-time`, and an inner **repeat-until** cycle which is executed till TAB_{EXT} does not change. Termination of the inner **repeat-until** cycle is guaranteed by using a finite constant as an upper bound in the gap-order constraints and computing TAB_{EXT} only up to that value. Termination of the outer **repeat-until** is ensured by Lemma 3. Hence, the algorithm terminates.
ii) We first prove that if $\exists(valid(t,\mathtt{A}),\Xi) \in M_P$ such that $\bar{t}$ satisfies Ξ, then $\exists\langle \mathtt{A},\Omega\rangle \in \text{TAB}_{EXT}$ with $\bar{t}$ satisfying Ω. By definition of ground model, $(valid(t,\mathtt{A}),\Xi) \in M_P$ and $\bar{t}$ satisfies Ξ imply $valid(\bar{t},\mathtt{A}) \in G_P$. By Lemma 1, $valid(\bar{t},\mathtt{A}) \in G_{P_{(t \leq \bar{t})}}$ and by Lemma 2, $valid(\bar{t},\mathtt{A}) \in G_{P''_{(t \leq \bar{t})}}$. By Lemma 7, the second step of the algorithm computes $M_{P''_{(t \leq \mathtt{max})}}$. Hence, if $\bar{t} \leq$`max`, the TAB_{EXT} computed by the algorithm includes an element $\langle \mathtt{A},\Omega\rangle$, such that $\bar{t}$ satisfies Ω. Let us suppose $\bar{t} >$`max`. By definition, $\mathtt{max} = \bar{\mathtt{t}}_{max} + k^* \cdot \mathtt{P}_{max}$, where k^* is the minimum between $\bar{k}$ (i.e., the finite constant defined in Lemma 3) and $\hat{k}$. By Lemma 7 and by the definition of $\hat{k}$, this implies that k^* is equal to the minimum between $\bar{k}$ and the first positive integer $\hat{k}$ greater than two such that $C_{\hat{k}-1} \stackrel{\rightarrow}{=} C_{\hat{k}}$. Let us first suppose `max` $= \bar{\mathtt{t}}_{max} + \bar{k} \cdot \mathtt{P}_{max}$. Let $n \in Z^+$ be the first positive integer such that $\bar{t} < \bar{\mathtt{t}}_{max} + n \cdot \mathtt{P}_{max}$. $valid(\bar{t},\mathtt{A}) \in G_{P''_{(t \leq \bar{t})}}$ implies $valid(\bar{t},\mathtt{A}) \in G_{P''_{(t \leq \bar{\mathtt{t}}_{max} + n \cdot \mathtt{P}_{max})}}$. Thus, $valid(\bar{t},\mathtt{A}) \in C_n$. By Lemma 3, $C_n \stackrel{\rightarrow}{=} C_{\bar{k}}$. This implies $valid(\bar{t} - (n - \bar{k}) \cdot \mathtt{P}_{max},\mathtt{A}) \in C_{\bar{k}}$. Lemma 7 ensures that the TAB_{EXT} computed by the second step of the algorithm includes $\langle \mathtt{A},\Omega'\rangle$, with $\hat{t}$ satisfying Ω', where $\hat{t}$ is the instant in $[\bar{\mathtt{t}}_{max} + (\bar{k} - 1) \cdot \mathtt{P}_{max} + 1, \bar{\mathtt{t}}_{max} + \bar{k} \cdot \mathtt{P}_{max}]$, such that $\hat{t} = \bar{t} - (n - \bar{k}) \cdot \mathtt{P}_{max}$. If $\hat{t}$ satisfies Ω', then $\exists(PC, GC) \in \Omega'$ such that $\hat{t}$ satisfies both PC and GC. Since $\hat{t} > \bar{\mathtt{t}}_{max} + (\bar{k} - 1) \cdot \mathtt{P}_{max}$, and $\hat{t}$ satisfies GC, then GC is of the form:

$t' \leq t \leq t''$, where $\mathtt{t}_{max} + (\overline{k}-1)\cdot \mathtt{P}_{max} < \hat{t} \leq t'' \leq \mathtt{t}_{max} + \overline{k}\cdot \mathtt{P}_{max}$ and $t' \leq \mathtt{t}_{max}$. Thus step 3 of the algorithm replaces GC with $GC' = t' \leq t$. Thus, $\overline{t} = \hat{t} + (n-\overline{k})\cdot \mathtt{P}_{max}$ satisfies GC'. Moreover, $\hat{t}$ satisfies PC implies that $\overline{t}$ satisfies PC. Hence, $\exists\langle \mathtt{A}, \Omega\rangle \in \text{TAB}_{EXT}$ such that $\overline{t}$ satisfies Ω. Let us now suppose $\mathtt{max} = \mathtt{t}_{max} + \hat{k}\cdot \mathtt{P}_{max}$. In Lemma 3, we have shown that $C_i \overset{\rightarrow}{=} C_{i+1}$ implies $C_i \overset{\rightarrow}{=} C_j$, for each $j > i$. Thus, $valid(\overline{t}, \mathtt{A}) \in C_n$ implies $valid(\overline{t} - (n-\hat{k})\cdot \mathtt{P}_{max}, \mathtt{A}) \in C_{\hat{k}}$. Hence, using the same reasoning we have done for $\mathtt{max} = \mathtt{t}_{max} + \overline{k}\cdot \mathtt{P}_{max}$ we can conclude the thesis.
We are left to prove that if $\exists\langle \mathtt{A}, \Omega\rangle \in \text{TAB}_{EXT}$ such that $\overline{t}$ satisfies Ω, then $\exists(valid(t, \mathtt{A}), \Xi)) \in M_P$ with $\overline{t}$ satisfying Ξ. Let us first suppose $\overline{t} \leq \mathtt{max}$. By Lemma 7, $\exists(valid(t, \mathtt{A}), \Xi') \in M_{P''_{(t\leq \mathtt{max})}}$ such that $\overline{t}$ satisfies Ξ'. Thus, $valid(\overline{t}, \mathtt{A}) \in G_{P''_{(t\leq \mathtt{max})}}$. By Lemma 2, $valid(\overline{t}, \mathtt{A}) \in G_{P_{(t\leq \mathtt{max})}}$, and by Lemma 1, $valid(\overline{t}, \mathtt{A}) \in G_P$. Therefore, $\exists(valid(t, \mathtt{A}), \Xi) \in M_P$ such that $\overline{t}$ satisfies Ξ.
Let us now suppose $\overline{t} > \mathtt{max}$. It is easy to show that $\overline{t}$ satisfies Ω implies that there exists an instant $\hat{t} = \overline{t} - n\cdot \mathtt{P}_{max}$, $n \in Z^+$, $\mathtt{max} - \mathtt{P}_{max} < \hat{t} \leq \mathtt{max}$, such that $\hat{t}$ satisfies Ω. By Lemma 7, the second step of the algorithm computes $M_{P''_{(t\leq \mathtt{max})}}$. Hence, $\exists(valid(t, \mathtt{A}), \Xi') \in M_{P''_{(t\leq \mathtt{max})}}$ such that $\hat{t}$ satisfies Ξ'. Thus, $valid(\hat{t}, \mathtt{A}) \in G_{P''_{(t\leq \mathtt{max})}}$. Let us first suppose that $\mathtt{max} = \mathtt{t}_{max} + \overline{k}\cdot \mathtt{P}_{max}$. Hence, $valid(\hat{t}, \mathtt{A}) \in C_{\overline{k}}$. Thus, by Lemma 3 $valid(\overline{t}, \mathtt{A}) \in C_{\overline{k}+n}$, and thus $valid(\overline{t}, \mathtt{A}) \in G_{P''_{(t\leq \mathtt{t}_{max}+(\overline{k}+n)\cdot \mathtt{P}_{max})}}$. Then, by Lemma 2 $valid(\overline{t}, \mathtt{A}) \in G_{P_{(t\leq \mathtt{t}_{max}+(\overline{k}+n)\cdot \mathtt{P}_{max})}}$. It is sufficient to apply Lemma 1 to conclude the thesis. Suppose now that $\mathtt{max} = \mathtt{t}_{max} + \hat{k}\cdot \mathtt{P}_{max}$. In Lemma 3, we have shown that $C_i \overset{\rightarrow}{=} C_{i+1}$ implies $C_i \overset{\rightarrow}{=} C_j$, for each $j > i$. Thus, $valid(\hat{t}, \mathtt{A}) \in C_{\hat{k}}$ implies $valid(\overline{t}, \mathtt{A}) \in C_{\hat{k}+n}$, since $\overline{t} = \hat{t} + n\cdot \mathtt{P}_{max}$. Hence, using the same reasoning we have done for $\mathtt{max} = \mathtt{t}_{max} + \overline{k}\cdot \mathtt{P}_{max}$ we can conclude the thesis. $\square$

REFERENCES

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A calculus for access control in distributed systems. *ACM Transactions on Programming Language and Systems 15*, 4 (September), 706–734.

BERTINO, E., BETTINI, C., FERRARI, E., AND SAMARATI, P. 1996c. On using materialization strategies for a temporal authorization model. In *Post-SIGMOD Workshop on Materialized Views: Techniques and Applications Proceedings* (June 1996), pp. 34–81.

BERTINO, E., BETTINI, C., FERRARI, E., AND SAMARATI, P. 1996b. Supporting periodic authorizations and temporal reasoning in database access control. In *22nd International Conference on Very Large Databases (VLDB'96) Proceedings* (August 1996), pp. 472–483.

BERTINO, E., BETTINI, C., FERRARI, E., AND SAMARATI, P. 1996a. A temporal access control mechanism for database systems. *IEEE Transactions Knowledge and Data Engineering 8*, 1 (February), 67–80.

BERTINO, E., BETTINI, C., FERRARI, E., AND SAMARATI, P. 1997. Decentralized administration for a temporal access control model. *Information Systems 22*, 4, 223–248.

BERTINO, E., SAMARATI, P., AND JAJODIA, S. 1993. Authorizations in relational database management systems. In *First ACM Conference on Computer and Communications Security Proceedings* (November 1993). ACM.

DATE, C. 1995. *An Introduction to Database Systems, 6th Edition.* Addison-Wesley Publishing Company, Reading, Massachusetts.

FALASCHI, M., LEVI, G., MARTELLI, M., AND PALAMIDESSI, C. 1988. A new declarative semantics for logic languages. In *Fifth International Conference and Symposium on Logic Programming Proceedings* (1988), pp. 993–1005. IEEE.

FERRARI, E. 1998. *Access Control Mechanisms for Database Systems: Formal Models and Architectural Aspects.* Phd Thesis, Dipartimento di Scienze dell'Informazione, Università di Milano.

Foundation, O. S. 1993. *OSF/Motif Programmer's Guide*. Prentice Hall.

Gelder, A. V., Ross, K., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38*, 3 (July), 620–650.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Fifth International Conference on Logic Programming Proceedings* (1988), pp. 1070–1080.

Gottlob, G., Marcus, S., Nerode, Salzer, and Subrahmanian, V. 1996. A non-ground realization of the stable and well-founded semantics. *Theoretical Computer Science 166*, 1&2, 221–262.

Gupta, A., Mumick, I., and Subrahmanian, V. 1993. Maintaining views incrementally. In *ACM SIGMOD International Conference on Management of Data Proceedings* (May 1993), pp. 157–166.

Informix Software, I. 1993. *The Informix Guide to SQL: Reference.*

Jajodia, S., Samarati, P., Subrahmanian, V., and Bertino, E. 1997. A unified framework for enforcing multiple access control policies. In *ACM SIGMOD International Conference on Management of Data Proceedings* (May 1997). ACM.

Lu, J., Ludascher, B., Schu, J., and Subrahmanian, V. 1996. *Well-founded views in constraint databases: incremental materialization and maintenance.* Technical Report, University of Maryland.

Lu, J., Moerkotte, G., Schu, J., and Subrahmanian, V. 1995. Efficient maintenance of materialized mediated views. In *ACM SIGMOD International Conference on Management of Data Proceedings* (May 1995). ACM.

Niezette, M. and Stevenne, J. 1992. An efficient symbolic representation of periodic time. In *First International Conference on Information and Knowledge Management Proceedings* (1992).

Revesz, P. 1993. A closed form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science 116*, 1, 117–149.

Revesz, P. 1995. Safe stratified Datalog with integer order programs. In *First International Conference on Principles and Practice of Constraint Programming Proceedings* (1995), pp. 154–169.

Steiner, J. G., Neuman, C., and Schiller, J. I. 1988. Kerberos: An authentication service for open network systems. In *USENIX Conference Proceedings* (1988), pp. 191–202. USENIX.

Toman, D., Chomicki, J., and Rogers, D. 1994. Datalog with integer periodicity constraints. In *International Logic Programming Symposium Proceedings* (1994), pp. 189–203.

Woo, T. and Lam, S. 1993. Authorizations in distributed systems: A new approach. *Journal of Computer Security 2*, 2&3, 107–136.