Supporting Periodic Authorizations and Temporal Reasoning in Database Access Control

Elisa Bertino Claudio Bettini Elena Ferrari Pierangela Samarati
Dipartimento di Scienze dell'Informazione
Università di Milano, Italy
{bertino,bettini,ferrarie,samarati}@dsi.unimi.it

Abstract

Several formal models for database access control have been proposed. However, little attention has been paid to temporal issues like authorizations with limited validity or obtained by deductive reasoning with temporal constraints. We present an access control model in which authorizations contain periodic temporal intervals of validity. An authorization is automatically granted in the time intervals specified by a periodic expression 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 prove the uniqueness of the set of implicit authorizations derivable at a given instant from the explicit ones, and we propose an algorithm to compute the global set of valid authorizations. The resulting model provides a high degree of flexibility and allows to express several protection requirements that cannot be expressed in traditional access control models.

1 Introduction

As an increasing number of applications entrust their data to database systems, the need for access control mechanisms increases. Most commercial DBMS provide an authorization mechanism by using which users are given access authorizations to objects under different modes, such as read or write. Upon a data access request from a user, the authorization mechanism checks whether the user is authorized for the access.

Authorization mechanisms, such as the ones supported by commercial DBMS, 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 usually limited in time or may hold only for specific periods of time. Because a typical commercial DBMS does not provide any temporal authorization mechanism, implementing authorization management at application program level is the only solution for supporting temporal authorizations. However, such a solution is largely inadequate because it makes authorization specification and management very difficult, if at all possible.

Even more crucial is the need for periodic authorizations. Indeed, in many organizations, authorizations given to users must be tailored to the pattern of their activities within the organization. Therefore, users must have access authorizations only for the time periods in which they are expected to need the data. As an example of periodic authorization, consider part-time staff that should be authorized to read a given file only each working day between 9 a.m and 12 a.m. Periodic authorizations are also very important when dealing with execution authorizations for application programs. Controlling the time periods during which specific application programs can be invoked is very useful for optimizing resource usage. Programs, whose execution is very resource-expensive, could be assigned 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

Proceedings of the 22nd VLDB Conference Mumbai(Bombay), India, 1996

for periodic authorizations, the solution of implementing them as part of application programs is not viable.

When developing a temporal authorization model several issues must be addressed, including the definition of a formal semantics for the model, the development of strategies for efficient access control, and tools for authorization administration. Some of those issues have been addressed as part of the development of a temporal authorization model, presented in [2]. Under that model, authorizations contain temporal intervals of validity; an authorization is automatically revoked when the associated temporal interval expires. The model also provides rules for the automatic derivation of new authorizations from those explicitly specified. A formal semantics has been defined for temporal authorizations and derivation rules, based on the semantics of logic programs with negation. Moreover, strategies have been developed, based on view materialization approaches, to support efficient authorization checking.

However, our previous model does not provide periodic authorizations, which are - we believe - an essential ingredient of a temporal access control mechanism. The current paper complements our previous work with periodic access authorizations and rules. This is a major extension, both for the practical relevance of periodic expressions in specifying authorizations and for the involved theoretical and performance issues. In particular, the formal semantics used in the current extended model is based on Datalog programs with negation and periodicity and order constraints. The materialization strategy, proposed for the previous model, has been substantially extended to deal with periodicity and order constraints. We have also added new temporal operators (UPON and UPON-NOT) to cover interesting protection requirements that were not expressible in [2].

To the best of our knowledge, this authorization model is the first one proposing features such as temporal derivation rules and periodic authorizations. Relevant related work has been carried out in the framework of the Kerberos system [6]. Kerberos, based on client-server architecture, provides the notion of *ticket*, needed for requiring a service to the server, with an associated validity time. The validity time is used to save the client from the need to acquire a ticket for each interaction with the server. The ticket mechanism is not used to grant accesses to the resources managed by the system. Rather, it is only used to denote that a client has been authenticated by the authentication server. Thus, the scope of the temporal ticket mechanism is very different from our access control model. From the side of logical formalisms for security specifications, Woo and Lam in [9] have proposed a very general formalism for expressing authorization rules. Their language does not have explicit constraints to deal with temporal information, but has almost the same expressive power of first order logic. We believe that for the sake of efficiency, it is important to devise more restricted languages focusing only on relevant properties. The temporal authorization model we propose in this paper is a step in this direction.

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. Section 4 specifies the semantics of our model and proves its main formal properties. In Section 5 an algorithm for deriving the set of implicit and explicit authorizations is presented. Section 6 concludes the paper. Finally, Appendix A illustrates the Datalog extension that we use to represent the semantics of our rules.

2 Preliminaries: Representation of periodic expressions

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 to describe the semantics of periodic authorizations and rules, to prove formal properties of our model, and to perform deductive reasoning.

The symbolic formalism is essentially the one proposed by Niezette and Stevenne in [4], based on the notions of *calendars* and *periodic expressions*.

A calendar is defined as a set of consecutive intervals. Each interval of a calendar is numbered by a natural number, called *index* of the interval, in such a way that successive intervals are numbered by successive natural numbers. *Days*, *Months*, and *Years* are example of calendars representing respectively the set of all the days, the months, and the years, starting from a given time instant. Calendars can also be finite. For instance, the calendar *Years-from-1980-to-1992* represents the set of all the years between 1980 and 1992. We use symbol $\top$ to denote the special calendar having a single time interval (indexed by 1) and including the whole time line. Given two calendars C_1 and C_2, we say that C_1 is 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. New calendars can be dynamically constructed from the existing ones.[1] In our model, we postulate the existence of a set of predefined calendars containing *Hours*, *Days*, *Weeks*, *Months*, and *Years*.

Calendars can be combined to represent more general sets of periodic intervals, not necessarily contiguous, as, for instance, the set of *Mondays* or the set of

[1] We refer to [4] for details of the construction.

The third hours 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 2.1 (Periodic expression) *A periodic expression is defined as* $\mathtt{P} = \sum_{i=1}^{n} O_i.C_i \rhd r.C_d$, *where* $O_1 \in 2^{\mathbb{N}} \cup \{all\}$, $O_i \in 2^{\mathbb{N}}$ $i = 2, \ldots, n$, C_i *and* C_d *are calendars for* $i = 1, \ldots, n$, $C_d \sqsubseteq C_n$, *and* $r \in \mathbb{N}$.

In practice, O_1 is omitted when its value is *all*, whereas when O_i is a singleton it is represented by its unique element. $r.C_d$ is omitted when it is equal to $1.C_n$. Table 1 illustrates a set of periodic expressions and their meaning.

Each periodic expression P is a symbolic representation of a set of time intervals $\Pi(\mathtt{P})$.[2] For example, if P is the last expression in Table 1, then $\Pi(\mathtt{P})$ is the set of time intervals starting with the tenth hour (9 to 10a.m.) of the second, third, fourth, fifth, and sixth day of every week. Each interval has a duration of 3 hours.

Symbolic expressions, while convenient for the user, are not easy to manipulate in the deductive process. Hence, when an expression has been given by the user, we translate it into a different formalism. This formalism is based on sets of periodicity constraints over integer numbers and it is inspired by the work in [7]. Periodicity constraints denote infinite periodic sets of integers.

Definition 2.2 (Simple Periodicity Constraint) *Let* K *be a finite set of natural numbers,* x *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:* $x \equiv_k c$.

Periodicity constraint $x \equiv_k c$ 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 $x \equiv_k (y + c) \forall y = 0, \ldots, u$ as a compact representation for the disjunction of simple constraints: $x \equiv_k c \vee x \equiv_k c + 1 \vee \ldots \vee x \equiv_k c + u$.

Conjunction of simple periodicity constraints can be represented by means of *periodicity graphs* [7], in which each node represents a variable or the constant 0, and an edge labeled (k, c) exists between 0 and x iff the constraint $x \equiv_k c$ belongs to the conjunction.

Another type of constraints will be useful to specify periodic authorizations.

Definition 2.3 (Gap-order Constraint) *Let* u, l *be integers,* c *a non-negative integer, and* x, y *integer variables. A gap-order constraint is a formula of the form* $l < x$, $x < u$, $x = y$, *or* $x + c < y$.

[2] We refer to [4] for the formal definition of $\Pi()$.

Conjunction of Gap-order constraints can be represented by means of *gap graphs* [5], that is, by means of graphs where the nodes represent the variables and the lower and the upper bound of the constraints, and edges represent gaps. Equality constraints are represented by undirected edges labeled by "=". For each pair of nodes a single edge, either representing equality or a gap, is allowed. The operations of conjunction ($\wedge$) and complement ($\neg$) of gap graphs are defined in [5], while the operation of conjunction of periodicity graphs is defined in [7]. For lack of space we do not report these definitions in the paper. However, the complement of periodicity graphs has never been defined and we do it here. Let G be a periodicity graph representing n constraints. For each $x \equiv_k c$ represented by G, consider the constraints $x \equiv_k r$ with $r = 0, \ldots, c-1, c+1, \ldots, k-1$. $\neg G$, the complement of G, is the disjunction of the $n * (k-1)$ periodicity graphs representing these constraints. Operations on gap graphs are analogous.

Example 2.1 Consider the sets of periodicity constraints $C_1 = \{x \equiv_7 1, y \equiv_{10} 1\}$ and $C_2 = \{x \equiv_{14} 1\}$. C_1 can be represented by means of a periodicity graph G_1 with nodes $x, y, 0$, and with two edges: $(0, x)$, with label $(7, 1)$, and $(0, y)$, with label $(10, 1)$. Similarly C_2 can be represented by a graph G_2 with nodes x and 0 and with a unique edge $(0, x)$, with label $(14, 1)$. $G_1 \wedge G_2$ is the graph with nodes x, y, and 0, and the edges: $(0, x)$ with label $(14, 1)$ and $(0, y)$ with label $(10, 1)$. $\neg G_1$ is the disjunction of the periodicity graphs corresponding to the constraints: $x \equiv_7 0$, $x \equiv_7 2$, ..., $x \equiv_7 6$, $y \equiv_{10} 0$, $y \equiv_{10} 2$, ..., $y \equiv_{10} 9$.

It is easily shown [1] that any symbolic periodic expression can be translated into a set of simple periodicity constraints. Each constraint in that set will have the form $t \equiv_{Periodicity(\mathtt{P})} c$, where $Periodicity(\mathtt{P})$ is the number n of units of the basic granularity that identifies the periodicity with which the time intervals in $\Pi(\mathtt{P})$ repeat themselves. For example, if *Hours* is the basic granularity, the periodicity of the first, fourth, and fifth expressions in Table1 is 168 (a week expressed in hours). The constants c must be such that all possible solutions of the disjunction of the periodicity constraints in the set are equivalent to the instants in the time intervals of $\Pi(\mathtt{P})$. For example, the expression *Weeks* + 2.*Days*, identifying *Mondays*, is translated into $t \equiv_{168} (y + 24)$ $\forall y = 0, \ldots, 23$. 168 is the number of hours in a week (the periodicity of Mondays), 24 is the distance in hours of the first Monday from the beginning of the period (the displacement of Mondays), and $0, \ldots, 23$ are the 24 hours within each Monday (the duration of each Monday).

Symbolic periodic expressions together with simple gap-order constraints are used by users to specify peri-

periodic expression	meaning
Weeks + {2,6}.*Days*	Mondays and Fridays
Months + 20.*Days*	The twentieth day of every month (Pay-days)
Years + 7.*Months* ▷ 3.*Months*	Summer-time
Weeks + {2,...,6}.*Days*	Working-days
Weeks + {2,...,6}.*Days* + 10.*Hours* ▷ 3.*Hours*	Each Working day between 9 a.m. and 12 a.m.

Table 1: Example of periodic expressions

odic access authorizations and rules, while their "fully" constraint counterpart is used to express the semantics of authorizations and rules, as well as for proving formal properties of the model and to perform access control. For simplicity, in the following periodicity and gap graphs will be denoted with the set of constraints they represent.

3 Periodic authorizations and rules

Our model allows to express periodic authorizations, that is, authorizations for a user to access an object in specific time intervals specified by a periodic expression. Moreover, a time interval is associated with each authorization, imposing lower and upper bounds to the potentially infinite set of instants denoted by the periodic expression. We refer to an authorization together with its time interval and periodic expression as *periodic authorization*. In the following U denotes the set of users, O the set of objects, and M the set of access modes.

We start by introducing the definition of authorization.

Definition 3.1 (Authorization) *An authorization is a 5-tuple* (s,o,m,pn,g), *with* s, g$\in U$, o$\in O$, m$\in M$, pn $\in \{+,-\}$.

Tuple (s,o,m,pn,g) states that user s has been authorized (if pn = '+') or denied (if pn = '−') for privilege m on object o by user g.

Definition 3.2 (Periodic Authorization) *A periodic authorization is a triple* (time,period,auth), *where* time *is a time interval* [min,max], *such that* min *and* max *denote respectively instants* t_b *and* t_e *with* $0 \leq t_b \leq t_e$, period *is a periodic expression, and* auth *is an authorization.*

Triple ([min,max],P,(s,o,m,pn,g)), states that user g had granted an authorization to user s for access mode m on object o, that holds for *each* instant in Π(P) limited by the interval $[t_b,t_e]$, where t_b and t_e are the instants corresponding respectively to min and max. The beginning and ending points of the interval associated with an authorization can be specified in one of the basic calendars. We use the notation 1/1/94:08 to represent 8a.m. on 1/1/94. When 1/1/94 is used as a minimum, it denotes the first instant of the first day of January 1994, while, as a maximum, it denotes the last instant of 1/1/94. We use symbol ∞ for max to denote a periodic authorization that spans from the starting time of its interval to infinity.

For example, the periodic authorization ([1/1/94,∞],Mondays[3],(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. Note that a non periodic authorization, that is, an authorization that holds continuously for a specific set of time instants can be expressed by a periodic authorization using $\top$ as the period component.

The model also allows to specify *derivation rules* from which other authorizations can be derived. The derivation is based on temporal propositions, used as rules, which allow new periodic authorizations to be derived on the basis of the presence or the absence of other periodic authorizations. Like authorizations, derivation rules have an associated time interval and a periodicity, representing the set of instants in which the derivation rule can be applied.

Definition 3.3 (Derivation rule) *A derivation rule is defined as* (time, period, A_1 ⟨OP⟩ A_2), *where* time=[min,max] *is the time interval associated with the rule, such that* min *and* max *denote respectively instants* t_b *and* t_e *with* $0 \leq t_b \leq t_e$, period *is a periodic expression,* A_1 *and* A_2 *are authorizations, and* ⟨OP⟩ *is one of the following operators:* WHENEVER, ASLONGAS, UPON, WHENEVER-NOT, UNLESS, UPON-NOT.

Rule ([min,max], P, (s_1,o_1,m_1,pn_1,g_1) ⟨OP⟩ (s_2,o_2,m_2,pn_2,g_2)) states that for each instant in[4] Π(P)$\cap\{[t_b,t_e]\}$, user s_1 is authorized (if pn_1 = '+') or denied (if pn_1 = '−') for access mode m_1 on object o_1 according to the presence or absence (depending on the ⟨OP⟩) of the authorization (s_2,o_2,m_2,pn_2,g_2). A derivation rule in which P=$\top$ represents a rule which can

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

[4]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 intersection operation ($T_1 \cap T_2$) has the usual semantics defined for sets.

be applied for each instant in [min,max]. A derivation rule with max = ∞ can be applied from the starting time of its time interval up to infinity. A simple extension to the above syntax allows to use a special symbol (*) instead of an authorization user, object or modifier with the meaning that any value in the corresponding domain can be used. This provides a compact form to express a set of derivation rules [2]. For simplicity we do not consider this extension in the rest of the paper.

Figure 1 illustrates an example of periodic authorizations and derivation rules.

We now give the intuitive semantics of the different kinds of derivation rules allowed by our model. The formal semantics will be given in the next section. 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.

- ([min,max], P, A_1 WHENEVER A_2). We can derive A_1 for each instant in $\Pi(P) \cap \{[t_b, t_e]\}$ for which A_2 is given or derived.
  For instance, rule R_1 in Figure 1 states that summer-staff can read document for every instant in Summer-time, from 1/1/1995, in which staff can do it.

- ([min,max],P, A_1 ASLONGAS A_2). We can derive A_1 for each instant t in $\Pi(P) \cap \{[t_b, t_e]\}$ such that A_2 is either given or derived for each time instant in $\Pi(P) \cap \{[t_b, t_e]\}$ from the first one up to t.
  For instance, rule R_3 in Figure 1 states that Jim can read document in a working day starting from 7/20/95 if summer-staff has been authorized for it for each working day from 7/20/95.

- ([min,max],P; A_1 UPON A_2). We can derive A_1 for each instant t in $\Pi(P) \cap \{[t_b, t_e]\}$ if there exists an instant $t' < t$ in $\Pi(P) \cap \{[t_b, t_e]\}$ such that A_2 is either given or derived at time t'.
  For instance, rule R_5 in Figure 1 states that technical-staff is forbidden to write report for every instant from the first time in [95,∞] at which technical-staff is allowed to read report-evaluation.

- ([min,max], P, A_1 WHENEVER-NOT A_2). We can derive A_1 for each instant in $\Pi(P) \cap \{[t_b, t_e]\}$ for which A_2 is neither given nor derived.
  For instance, rule R_6 in Figure 1 states that Ann can read pay-checks each working day in 1995 and 1996 in which Tom is not allowed to write pay-checks.

- ([min,max], P, A_1 UNLESS A_2). We can derive A_1 for each instant t in $\Pi(P) \cap \{[t_b, t_e]\}$ such that A_2 is neither given nor can be derived for each instant in $\Pi(P) \cap \{[t_b, t_e]\}$ from the first one up to t.
  For instance, rule R_2 in Figure 1 states that temporary-staff can read document each working day starting from 1/1/95 until the first working day summer-staff will be authorized for that.

- ([min,max],P, A_1 UPON-NOT A_2). We can derive A_1 for each instant t in $\Pi(P) \cap \{[t_b, t_e]\}$ such that there exists an instant $t' < t$ in $\Pi(P) \cap \{[t_b, t_e]\}$ in which A_2 is neither given nor derived.
  For instance, rule R_4 in Figure 1 states that technical-staff can write report each working day from the first working day in [95,∞] in which manager does not have the authorization to write guidelines.

Example 3.1 Consider the authorizations and rules in Figure 1. Among the authorizations that can be derived are:

- (temporary-staff,document,read,+,Sam) for each working day in [1/1/95,6/30/95], from rules R_2 and R_1, and authorization A_1.

- (Jim,document,read,+,Sam) for each working day in [7/20/95,10/30/95], from rules R_3 and R_1, and authorization A_1.

4 Formal semantics

In this section we formalize the semantics of periodic authorizations and derivation rules. First, it is necessary to point out that the possibility of expressing negative authorizations introduces potential conflicts among authorizations. Indeed, a positive authorization states that an access must be granted whereas a negative authorization states that an access should be denied. A conflict therefore exists every time both a positive and a negative authorization exist for the same subject. We solve this conflict according to the denials-take-precedence principle. For instance, consider the authorizations and rules in Figure 1. From R_4 and A_2 we derive the authorization for technical-staff to write report for each working day in [5/21/95,∞]. From R_5 and A_4, we derive a negative authorization for the same access for all instants in [10/1/95,∞]. Thus, technical-staff will be allowed for the access only for the working days in [5/21/95,9/30/95].

The formal semantics considers this fact.

Definition 4.1 (Temporal Authorization Base) *A Temporal Authorization Base* (TAB) *is a set of periodic authorizations and derivation rules.*

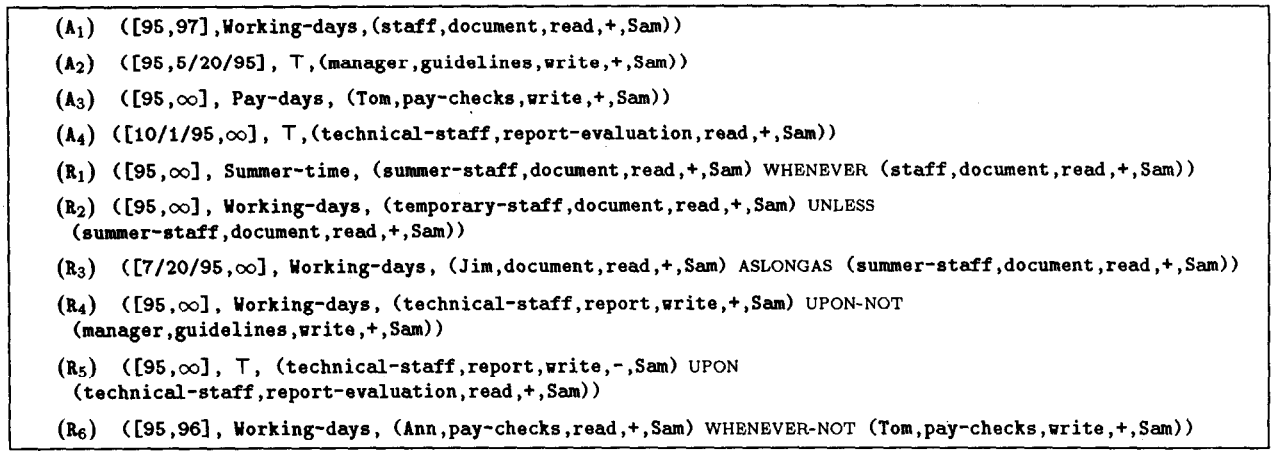

Figure 1: An example of authorizations and derivation rules

The semantics of a TAB is given as a set of clauses in a Datalog$^{\mathbf{not},\equiv Z,<Z}$ program corresponding to TAB. 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 will be actually a very restricted class of Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs: the only predicate symbols are $F(), F_N(), F_P(), G()$, and CNSTR(), a limited set of non-temporal constants $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,-,\mathtt{P},\ldots)$ is provided to denote users, objects, access modes, sign of authorizations, and periodic expressions.[5] Periodicity and order constraints only involve temporal variables and do not use the + function.

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 $\Xi = \{(G_1,H_1),\ldots,(G_m,H_m)\}$ a set of constraints on the temporal variables of B. Each G_i is a periodicity graph and each H_i is a gap graph. Ξ is a disjunction of these pairs, i.e., it is satisfied if there exists i such that both G_i and H_i are satisfied. Each constrained interpretation has an equivalent (possibly infinite) Herbrand interpretation containing only ground atoms.

Table 2 reports the clause/set of clauses in Datalog$^{\mathbf{not},\equiv Z,<Z}$ corresponding to each type of authorization/rule allowed by our model.[6] Intuitively, the predicate $F()$ is used to represent the authorizations at specific instants. The fact that $(F(t,\mathtt{A}),\Xi)$ belongs to an interpretation means that A is valid according to that interpretation at all instants t satisfying Ξ. The predicates $G(), F_N()$ and $F_P()$ are auxiliary predicates, used to avoid quantification. Intuitively, $G(t,\mathtt{s},\mathtt{o},\mathtt{m})$ is true in an interpretation if there is at least one negative authorization, with the same s,o,m, valid at instant t according to that interpretation. $F_N(t'',t,\mathtt{P},\mathtt{A})$ is true in an interpretation if there is at least an instant t' with $t'' \leq t' < t$ and t' in the set denoted by P at which authorization A is not valid according to that interpretation. $F_P(t'',t,\mathtt{P},\mathtt{A})$ is true in an interpretation if there is at least an instant t' with $t'' \leq t' < t$ and t' in the set denoted by P at which authorization A is valid according to that interpretation.

We denote the Datalog$^{\mathbf{not},\equiv Z,<Z}$ program corresponding to a TAB with P_{TAB}. We consider *stable model semantics* of logic programs with negation [3] to identify the models of P_{TAB}. The notion of constrained interpretation presented above naturally extends to constrained (non-ground) stable models.

Definition 4.2 (Valid Authorization) *Let M be a model of P_{TAB}. An authorization A is said to be valid at time $\bar{t}$ with respect to M if $(F(t,\mathtt{A}),\Xi)$ is contained 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 t.*

4.1 Restrictions on rules

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

Example 4.1 Consider the following rules:
$(\mathtt{R}_1)$ $([\mathtt{min},\mathtt{max}],\mathtt{P},\mathtt{A}_1$ WHENEVER-NOT $\mathtt{A}_2)$

[5] Note that when P appears as a predicate argument it denotes a non-temporal constant that we associate with a periodic expression.

[6] For brevity, we use the form $\mathtt{t}_b \leq t \leq \mathtt{t}_e$ as a shortcut for the conjunction of the two gap-order constraints $c_1 < t$ and $t < c_2$ with $c_1 = \mathtt{t}_b - 1$ and $c_2 = \mathtt{t}_e + 1$. Similarly, constraint $t'' \leq t' < t$ is a shortcut for the disjunction (using two clauses) of $t' = t''$ and $t'' < t' < t$.

[min, max], P, (s,o,m,−,g) :
$F(t,\mathtt{s},\mathtt{o},\mathtt{m},-,\mathtt{g}) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t)

[min, max],$\mathtt{t}_e$], P, (s,o,m,+,g) :
$F(t,\mathtt{s},\mathtt{o},\mathtt{m},+,\mathtt{g}) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), **not**$(G(t,\mathtt{s},\mathtt{o},\mathtt{m}))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1)$ WHENEVER $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1)$ WHENEVER $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$, **not**$(G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1)$ ASLONGAS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$, **not**$(F_N(\mathtt{t}_b, t,\mathtt{P},\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2))$

[min, max],P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1)$ ASLONGAS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$, **not**$(F_N(\mathtt{t}_b, t,\mathtt{P},\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2))$, **not**$(G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1)$ UPON $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F_P(\mathtt{t}_b, t,\mathtt{P},\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2)$

[min, max],P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1)$ UPON $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F_P(\mathtt{t}_b, t,\mathtt{P},\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2)$, **not**$(G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1)$ WHENEVER-NOT $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), **not**$(F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1)$ WHENEVER-NOT $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), **not**$(F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2))$, **not**$(G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1)$ UNLESS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), **not**$(F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2))$, **not**$(F_P(\mathtt{t}_b, t,\mathtt{P}, \mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1)$ UNLESS $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t) , **not**$(F(t,\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2))$, **not**$(F_P(\mathtt{t}_b, t,\mathtt{P},\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2))$, **not**$(G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1))$

[min, max], P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1)$ UPON-NOT $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,-,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F_N(\mathtt{t}_b, t,\mathtt{P},\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2)$

[min, max],P, $(\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1)$ UPON-NOT $(\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2,\mathtt{pn},\mathtt{g}_2)$:
$F(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1,+,\mathtt{g}_1) \leftarrow \mathtt{t}_b \leq t \leq \mathtt{t}_e$, CNSTR(P, t), $F_N(\mathtt{t}_b, t,\mathtt{P},\mathtt{s}_2,\mathtt{o}_2,\mathtt{m}_2, \mathtt{pn},\mathtt{g}_2)$, **not**$(G(t,\mathtt{s}_1,\mathtt{o}_1,\mathtt{m}_1))$

Auxiliary clauses:

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

$\{$CNSTR(P, t) $\leftarrow t \equiv_{periodicity(\mathtt{P})} y\}$
$\forall y$ such that $t \equiv_{periodicity(\mathtt{P})} y \Rightarrow t \in \Pi(\mathtt{P})$

$\{F_P(t'', t,\mathtt{P}, s, o, m, pn, g) \leftarrow t'' \leq t' < t$, CNSTR(P, t'), $F(t', s, o, m, pn, g)\}$
$\forall$ distinct P appearing in an UNLESS/UPON rule

$\{F_N(t'', t,\mathtt{P}, s, o, m, pn, g) \leftarrow t'' \leq t' < t$, CNSTR(P, t'),**not**$(F(t', s, o, m, pn, g))\}$
$\forall$ distinct P appearing in an ASLONGAS/UPON-NOT rule

Table 2: Semantics of periodic authorizations and rules

(R_2) ([min, max], P, A_2 WHENEVER-NOT A_1)
Suppose that there are no explicit authorizations for A_1 or A_2 in the TAB and these are the only rules. If we consider first R_1 we derive authorization A_1 for each instant in $\{[t_b, t_e]\} \cap \Pi(P)$, and we cannot derive A_2. If we consider first R_2, we derive A_2 for the same time intervals and not A_1. Hence, we have two different sets of derived authorizations. In this case there is no reason to give preference to one set or the other.

From the point of view of the semantics, the property of always having a unique set of valid authorizations is guaranteed only if all the models of the program corresponding to the TAB identify the same set of valid authorizations at any instant (or equivalently, there exists a *unique* ground stable model equivalent to all the models of P_{TAB}). In the rest of this section we formally define restrictions on sets of rules in order to guarantee a unique ground model for P_{TAB}. We also give an algorithm for checking the satisfaction of these restrictions.

In the following, we use the term *negative operator* to refer to WHENEVER-NOT, UNLESS, and *past operator* (PASTOP) to refer to UNLESS, ASLONGAS, UPON-NOT and UPON. Moreover, we use symbols A_i as a shortcut for the 5-tuple $(s_i, o_i, m_i, pn_i, g_i)$, while A_i^+ forces $pn_i = +$ and A_i^- forces $pn_i = -$.

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

- if there is a rule ([min,max], P, $A_m\langle$OP$\rangle$ A_n) in TAB, where $\langle$OP$\rangle$ is an arbitrary operator, then $A_n[t] \hookrightarrow A_m[t]$ for each $t \in \{[t_b, t_e]\} \cap \Pi(P)$. The $\hookrightarrow$ relation represents a dependency of A_m at instant t from A_n at the same instant. When $\langle$OP$\rangle$ is a negative operator we say that $\hookrightarrow$ represents a strict dependency.

- if there is a rule ([min,max], P, A_m $\langle$PASTOP$\rangle$ A_n) in TAB, then $A_n[t] \hookrightarrow A_m[t']$ for each $t, t' \in \{[t_b, t_e]\} \cap \Pi(P)$, $t < t'$. If PASTOP is equal to ASLONGAS, UNLESS or UPON-NOT then $\hookrightarrow$ represents a strict dependency.

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

Definition 4.3 (Priority) *An authorization A_n at time t has higher priority than an authorization A_m at time t' (written $A_n[t] > A_m[t']$) if one of the following conditions holds:*

- *a sequence $A_n[t]=A_1[t] \hookrightarrow \ldots \hookrightarrow A_{k-1}[t''] \hookrightarrow A_k[t']=A_m[t']$ exists such that at least one of the $\hookrightarrow$ relationships is a strict dependency,*

- *two sequences $A_n[t]=A_1[t] \hookrightarrow \ldots \hookrightarrow A_l^-[t'']$ and $A_{l+1}^+[t''] \hookrightarrow \ldots \hookrightarrow A_k[t']= A_m[t']$ exist such that $s(A_l^-)=s(A_{l+1}^+)$, $o(A_l^-)=o(A_{l+1}^+)$, and $m(A_l^-)=m(A_{l+1}^+)$,*[7]

- *an authorization A_l and an instant t'' exist such that $A_n[t] > A_l[t'']$ and $A_l[t''] > A_m[t']$.*

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

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

Definition 4.4 (Critical set) *A* TAB *contains a critical set of rules if and only if an authorization A_m in* TAB *and an instant t exist such that A_m at instant t has priority over itself ($A_m[t] > A_m[t]$).*

The CSD (Critical Set Detection) algorithm, described in the next subsection, can be used to recognize and reject 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 *graph version*, denoted as TAB′, as the set of pairs of the form $\langle x, \Xi \rangle$, where x is either an authorization or rule in TAB and Ξ the set of pairs (G, H) representing the temporal constraints associated with it in TAB. Essentially, a temporal authorization ($[t_b, t_e]$, P, A_m) is mapped into the pair $\langle A_m, \{(H, G_1), \ldots, (H, G_k)\}\rangle$, where H represents constraint $\{t_b \leq t \leq t_e\}$ and $G_1, \ldots, G_k$ represent the periodicity constraints corresponding to P. If A_m is specified more than once in TAB with different temporal constraints the set Ξ associated with A_m in TAB′ is the union of the sets corresponding to the different constraints. Derivation rules are transformed in an analogous way. In the following, given an authorization A_m in TAB, Ξ_m denotes the constraints associated with A_m in TAB′. Analogously, Ξ_R denotes the constraints associated with rule R in TAB′.

We introduce the operations of conjunction ($\wedge^*$) and complement ($\neg^*$) between sets of pairs (G, H). Let $\Xi = \{(G_1, H_1), \ldots, (G_m, H_m)\}$ and let $\Xi' = \{(G'_1, H'_1), \ldots, (G'_k, H'_k)\}$. $\Xi \wedge^* \Xi'$ is the disjunctive normal form of the result obtained by the conjunction of the formulas corresponding to Ξ and Ξ'. For instance $\{(G_1, H_1), (G_2, H_2)\} \wedge^* \{(G'_1, H'_1)\} = \{(G_1 \wedge$

[7] We use the notation s(A),o(A),m(A),pn(A) to denote respectively the subject, the object, the privilege and the sign in A.

$G'_1, H_1 \wedge H'_1), (G_2 \wedge G'_1, H_2 \wedge H'_1)\}$. The operation of complement ($\neg^*$) can be defined in a similar way.

The algorithm for detecting critical sets receives as input TAB', i.e. the graph version of TAB. It returns FALSE if either a critical set exists or the number of levels exceeds a fixed upper bound. Otherwise, it returns a sequence of levels $\langle L_1, \ldots, L_k \rangle$ representing a finite partition of the set of pairs $\langle \mathtt{A}, t \rangle$ for each authorization $\mathtt{A}$ in TAB' and $\mathtt{t}_{min} \leq t \leq \mathtt{t}_{max}$, where $\mathtt{t}_{min}$ and $\mathtt{t}_{max}$ are respectively the minimum and maximum constant (included ∞) appearing in TAB'. Each level L_i is a set of pairs $\langle \mathtt{A}_j, \Xi_{j,i} \rangle$, where $\Xi_{j,i}$ denotes the temporal constraints associated with $\mathtt{A}_j$ at level L_i. Intuitively, authorizations appearing at lower levels for a certain set of instants have higher priority for evaluation than authorizations appearing at higher levels (for the same or for a different set of instants).

The algorithm starts by putting at level 1 all the authorizations for all time instants between $\mathtt{t}_{min}$ and $\mathtt{t}_{max}$. Then, it considers the dependencies caused by negative authorizations and rules and possibly moves authorizations up in levels. Moving authorization $\mathtt{A}_m$ from a level h to a level k with constraints Ξ means updating $\Xi_{m,h}$ to be $\Xi_{m,h} \wedge^* \neg^* \Xi$ and updating $\Xi_{m,k}$ to be $\Xi_{m,k} \cup \Xi$.[8] The process 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**. **max-level** is an upper bound chosen as the number of authorizations in TAB' multiplied by ($\overline{\mathtt{t}}_{max}$ - $\mathtt{t}_{min}$ + $\mathtt{P}_{max}$ + 1), where $\overline{\mathtt{t}}_{max}$ is the maximum *finite* constant appearing in TAB' and $\mathtt{P}_{max}$ is the least common multiple (lcm) of all the periodicities appearing in TAB' (excluding $\top$).

The algorithm guarantees that if a critical set exists, it will be detected. Intuitively, in case of a critical set, the algorithm cycles over some priority relationship and soon it reaches the **max-level** upper bound.

When the algorithm reaches a fix-point before reaching **max-level**, it returns the levels that have been generated. These levels obey the priority relationship: If a dependency $\mathtt{A}_n[t] \hookrightarrow \mathtt{A}_m[t']$ exists in TAB, then $\mathtt{A}_m[t']$ appears at a level higher than or equal to $\mathtt{A}_n[t]$. The level is necessarily higher if the dependency is strict. Moreover, a positive authorization $\mathtt{A}_m$ appears at level higher than any negative authorization $\mathtt{A}_n$ with same subject, object, and access mode for the same time instant. The algorithm guarantees also that for each authorization $\mathtt{A}_m$, each instant t, $\mathtt{t}_{min} \leq t \leq \mathtt{t}_{max}$ satisfies the constraints $\Xi_{m,l}$ of exactly one level l. A detailed description of the algorithm can be found in [1].

[8] If $\mathtt{A}_m$ does not appear at level k before the operation, it is inserted and $\Xi_{m,k}$ initialized to Ξ.

Example 4.2 Consider a TAB containing the following authorizations and rules:
([95,∞], $\top$, $\mathtt{A}_1$)
($\mathtt{R}_1$) ([1/20/97,98], Mondays,$\mathtt{A}_1$UPON$\mathtt{A}_2^+$)
($\mathtt{R}_2$) ([96,97],Working-days,$\mathtt{A}_2^-$WHENEVER-NOT$\mathtt{A}_3$)
The corresponding TAB' contains the following pairs.[9]
$\langle \mathtt{A}_1, \{(\{\text{true}\}, \{1/1/95 \leq t\})\} \rangle$
$\langle \mathtt{A}_1\text{UPON}\mathtt{A}_2^+, \{(\{t \equiv_7 1\}, \{1/20/97 \leq t \leq 12/31/98\})\} \rangle$
$\langle \mathtt{A}_2^-\text{WHENEVER-NOT}\mathtt{A}_3, \{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/97\})\} \rangle$

Authorizations $\mathtt{A}_1$, $\mathtt{A}_2^+$, $\mathtt{A}_3$, and $\mathtt{A}_2^-$ are initially inserted at level 1 with constraints $\{(\{\text{true}\}, \{1/1/95 \leq t\})\}$.

The algorithm then cycles moving authorizations up in level as follows.
1st iteration:
For $\mathtt{A}_2^-$ at level 1: move $\langle \mathtt{A}_2^+, \{(\{\text{true}\}, \{1/1/95 \leq t\})\} \rangle$ to level 2.
For rule $\mathtt{R}_1$: move $\langle \mathtt{A}_1, \{(\{t \equiv_7 1\}, \{1/20/97 \leq t \leq 12/31/98\})\} \rangle$ to level 2.
For rule $\mathtt{R}_2$: move $\langle \mathtt{A}_2^-, \{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/97\})\} \rangle$ to level 2.
2nd iteration:
For $\mathtt{A}_2^-$ at level 2: move $\langle \mathtt{A}_2^+, \{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/97\})\} \rangle$ to level 3.
For rule $\mathtt{R}_1$: move $\langle \mathtt{A}_1, \{(\{t \equiv_7 1\} \{1/20/97 \leq t \leq 12/31/98\})\} \rangle$ 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 2.

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

Theorem 4.1 *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

In our model, the control of whether an access request can be authorized may require the evaluation of several rules. For this reason we adopt a materialization approach to enforce access control. Under such an

[9] For simplicity, here and in the following examples we assume *Days* as our finest granularity and Sunday 1/1/95 as our instant zero.

Level 1:
$\langle A_1, \{(\{\text{true}\}, \{1/1/95 \leq t \leq 1/19/97\}), (\{t \not\equiv_7 1\}, \{1/20/97 \leq t \leq 12/31/98\}), (\{\text{true}\}, \{1/1/99 \leq t\})\}\rangle$
$\langle A_2^-, \{(\{\text{true}\}, \{1/1/95 \leq t \leq 1/31/95\}), (\{t \not\equiv_7 (y+1), y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/97\}), (\{\text{true}\}, \{1/31/97 \leq t\})\}\rangle$
$\langle A_3, \{(\{\text{true}\}, \{1/1/95 \leq t\})\}\rangle$
Level 2:
$\langle A_2^-, \{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/97\})\}\rangle$
Level 3:
$\langle A_1, \{(\{t \equiv_7 1\}\{1/20/97 \leq t \leq 12/31/98\})\}\rangle$
$\langle A_2^+, \{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/97\})\}\rangle$

Figure 2: An example of levels returned by the CSD algorithm

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

In the following we illustrate how to compute, given a TAB, the corresponding valid authorizations. We start with the following definition.

Definition 5.1 (Temporal Authorization Base Extent) *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'. Each A_k is associated with a set of constraints Ω_k; $\langle A_k, \Omega_k \rangle$ is in TAB_{EXT} if authorization A_k is valid at each instant t satisfying Ω_k.

Given two sets of constraints Ξ and Ξ', we say that Ξ is *shift-equivalent* to Ξ' (written $\Xi \rightleftarrows^* \Xi'$), if the instants denoted by Ξ are a transposition of the instants denoted by Ξ' on the time axis. Formally:
$\Xi \rightleftarrows^* \Xi'$ if $\exists t' \in \mathbb{N}$ such that $t+t'$ satisfies Ξ' whenever t satisfies Ξ.

Figure 3 presents an algorithm to compute TAB_{EXT}. The algorithm is based on the model computation for (locally) stratified $\text{Datalog}^{\text{not},\equiv_Z,<_Z}$ programs given in Appendix A. This computation is represented in Algorithm 5.1 by an iteration of the **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 a finite constant can always be found, such that the computed TAB_{EXT} can be extended (Step 3 of Algorithm 5.1) to the actual TAB_{EXT} (possibly including ∞). This finite constant cannot be easily determined before running the algorithm, and this is the reason for the **repeat-until** cycle. In particular, the algorithm considers two contiguous time intervals after $\bar{t}_{max}$, of length equal to the maximum periodicity in TAB (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, it proceeds with another iteration of Step 2, generating a larger TAB_{EXT} using the constant of the previous iteration incremented by P_{max} (Step 2.1). We have proved that for any TAB the algorithm terminates.

Theorem 5.1 *i) Algorithm 5.1 terminates and ii) an authorization* A *is valid at time* $\bar{t}$ *if and only if there exists* $\langle A, \Omega \rangle$ *in* TAB_{EXT} *such that* $\bar{t}$ *satisfies* Ω.

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

Example 5.1 Consider the TAB in Example 4.2. The levels computed by the CSD algorithm are illustrated in Figure 2. We now apply the algorithm for TAB_{EXT} generation. At the first iteration of the **repeat-until** cycle $k = 2$ and *current_time*=1/14/99 (12/31/98 + 2 *Weeks*). Let $\text{TAB}_{EXT}^{(i)}$ be the TAB_{EXT} resulting from the evaluation of level L_i. We have:

$$\begin{aligned}
\text{TAB}_{EXT}^{(1)} &= \{\langle A_1, \{(\{\text{true}\}, \{1/1/95 \leq t \leq 1/19/97\}), \\
&\quad (\{t \not\equiv_7 1\}, \{1/20/97 \leq t \leq 12/31/98\}), \\
&\quad (\{\text{true}\}, \{1/1/99 \leq t \leq 1/14/99\})\}\rangle\} \\
\text{TAB}_{EXT}^{(2)} &= \text{TAB}_{EXT}^{(1)} \cup \{\langle A_2^-, \{(\{t \equiv_7 (y+1) \\
&\quad \forall y = 0, \ldots, 4\}, \{1/1/96 \leq t \leq 12/31/97\})\}\rangle\} \\
\text{TAB}_{EXT}^{(3)} &= \text{TAB}_{EXT}^{(2)} \cup \{\langle A_1, \{(\{t \equiv_7 1\}, \\
&\quad \{1/20/97 \leq t \leq 12/31/98\})\}\rangle\}
\end{aligned}$$

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 1/7/99$< \bar{t} \leq$1/14/99. Hence:

$$\begin{aligned}
\text{TAB}_{EXT} &= \{\langle A_1, \{(\{\text{true}\}, \{1/1/95 \leq t\})\}\rangle, \\
&\quad \langle A_2^-, \{(\{t \equiv_7 (y+1) \forall y = 0, \ldots, 4\}, \\
&\quad \{1/1/96 \leq t \leq 12/31/97\})\}\rangle\}
\end{aligned}$$

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.

Algorithm 5.1
INPUT: The output $\langle L_1, \ldots, L_k \rangle$ of the CSD Algorithm and TAB'.
OUTPUT: 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 $X_i \subseteq$ TAB' containing all $\langle \mathtt{A}_m, \Xi_m \rangle$ and $\langle \mathtt{R}, \Xi_{\mathtt{R}} \rangle$ ($\mathtt{R}$=$\mathtt{A}_m$ (OP) $\mathtt{A}_n$) such that $\mathtt{A}_m$ appears in L_i
      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** $x = \mathtt{A}_m$ (OP) $\mathtt{A}_n$:
               If OP = WHENEVER: reassign to Θ the conjunction of Ξ_n and Θ
               If OP = WHENEVER-NOT: reassign to Θ the conjunction of $\neg^*\Xi_n$ and Θ
               If OP is a past operator:
                  Let t be the unique variable appearing in Θ
                  Case OP **of**:
                     UPON: reassign to Θ the conjunction of $\{(true, t > \bar{t})\}$ and Θ, where $\bar{t}$ is the first instant satisfying Ξ and Ξ_n
                     UPON-NOT: reassign to Θ the conjunction of $\{(true, t > \bar{t})\}$ and Θ, where $\bar{t}$ is the first instant satisfying Ξ and $\neg^*\Xi_n$
                     ASLONGAS: reassign to Θ the conjunction of $\{(true, t < \bar{t})\}$ and Θ, where $\bar{t}$ is the first instant satisfying Ξ and $\neg^*\Xi_n$
                     UNLESS: reassign to Θ the conjunction of $\{(true, t < \bar{t})\}$ and Θ, where $\bar{t}$ is the first instant satisfying Ξ and Ξ_n
            (c) **If** $\mathtt{A}_m$ is a positive authorization: reassign to Θ the conjunction of Θ with the complement of each element in the set $\{\Xi_k \mid \mathtt{pn}(\mathtt{A}_k)=\text{`}-\text{'}, \mathtt{s}(\mathtt{A}_k)=\mathtt{s}(\mathtt{A}_m), \mathtt{o}(\mathtt{A}_k)=\mathtt{o}(\mathtt{A}_m), \mathtt{m}(\mathtt{A}_k)=\mathtt{m}(\mathtt{A}_m)\}$
            (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}
      $\Omega \wedge^* \{(true, \bar{\mathtt{t}}_{max}+(k-2) \cdot \mathtt{P}_{max} < t \leq \bar{\mathtt{t}}_{max}+(k-1) \cdot \mathtt{P}_{max})\} \stackrel{?}{=}^*$
      $\Omega \wedge^* \{(true, \bar{\mathtt{t}}_{max}+(k-1) \cdot \mathtt{P}_{max} < t \leq \bar{\mathtt{t}}_{max}+k \cdot \mathtt{P}_{max})\}$:
      $success := true$
   Until *success*
3. **For** each $\langle \mathtt{A}, \Omega \rangle$ in TAB_{EXT}: substitute with ∞ each value $\bar{t}$ such that $\bar{\mathtt{t}}_{max}+(k-1) \cdot \mathtt{P}_{max} < \bar{t} \leq \bar{\mathtt{t}}_{max}+k \cdot \mathtt{P}_{max}$

Figure 3: An algorithm for TAB_{EXT} generation

6 Conclusions

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 model results 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 time constraints, periodicity constraints, and derivation rules can be represented. We have defined such a model in this paper, and proved the main properties it satisfies. Moreover, we have given algorithms for controlling the consistency of an authorization base and for determining all authorizations derivable from it. Administrative operations to remove or add authorizations and derivation rules can be easily defined based on [2]. Further work includes the development of materialization strategies for the incremental maintenance of TAB_{EXT} upon execution of administrative operations.

References

[1] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. Supporting periodic authorizations and temporal reasoning in database access control. Technical report, DSI - University of Milano, Italy, 1996.

[2] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. A temporal access control mechanism for database systems. *IEEE Trans. on Knowledge and Data Engineering*, 8(1), February 1996.

[3] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc.* 5^{th} *Intl. Conf. on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988.

[4] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proc.* 1^{th} *Intl. Conf. on Information and Knowledge Management*, Baltimore, MD, November 1992.

[5] P.Z. Revesz. A Closed Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.

[6] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. USENIX Conf.*, pages 191–202, Dallas, TX, 1988.

[7] D. Toman, J. Chomicki, and D.S. Rogers. Datalog with Integer Periodicity Constraints. In *Proc. Intl. Logic Programming Symposium*, pages 189–203. MIT Press, 1994.

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

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

A 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*$^{\equiv Z,<Z}$ [7] with non monotonic negation [8], however, to our knowledge, it was never considered in the literature. *Datalog*$^{\mathbf{not},\equiv Z,<Z}$ programs are defined as follows.

Definition A.1 (*Datalog*$^{\mathbf{not},\equiv Z,<Z}$ Program) *A Datalog*$^{\mathbf{not},\equiv Z,<Z}$ *program P is a finite set of (function-free) clauses of the form*
$B \leftarrow D_1, \ldots, D_m, \mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}, C_1, C_2$
where $B, D_1, \ldots, D_{m+n}$ are atoms, C_1 is a satisfiable periodicity constraint, C_2 is a satisfiable gap-order constraint, and **not** *represents non monotonic negation.*

Bottom-up evaluation of Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs requires to perform operations on gap-graphs and periodicity graphs. To this purpose we need the operations of conjunction ($\wedge^*$) and complement ($\neg^*$) on sets of graphs illustrated in Subsection 4.2. We also need the operations of subsumption and projection (π), for periodicity graphs combined with gap graphs defined in [7]. Intuitively, if x, y, and z are the nodes in both G and H, then $\pi_{xy}(G, H)$ returns a set of pairs (G_i, H_i) obtained from (G, H) by dropping node z and all the edges ending in z, after computing all the constraints (edge's labels) implied by the edges to be dropped. The π operation also discards any resulting pair that is inconsistent. The subsumption operation has its intuitive semantics: a pair (G_1, H_1) is subsumed by (G_2, H_2) (having the same set of variables) if any assignment satisfying (G_1, H_1) satisfies also (G_2, H_2). Operations of subsumption and projection can be easily extended to sets of pairs (G, H), similarly to $\wedge^*$ and $\neg^*$.

Periodicity and gap graphs 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 set of pairs (G, H) denoting the disjunction of the corresponding constraints.

Given a *Datalog*$^{\mathbf{not},\equiv Z,<Z}$ program P we can define an operator 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 A.2 (TP$^{\mathbf{not},\equiv Z,<Z}$ operator) *Let P be a Datalog*$^{\mathbf{not},\equiv Z,<Z}$ *program and I a $(\equiv, <)$ interpretation.*

$$
\begin{aligned}
TP^{\mathbf{not},<Z,\equiv Z}(I) = I \cup \{\ & (B, \Xi) : B \leftarrow D_1, \ldots, D_m, \\
& \mathbf{not} D_{m+1}, \ldots, \mathbf{not} D_{m+n}, \\
& C_1, C_2 \in P \\
& (D_i, \Xi_i) \in I,\ \forall i = 1, \ldots, m \\
& \Theta = \Xi_1 \wedge^* \ldots \wedge^* \Xi_m \wedge^* \\
& \wedge^* \neg^* (\Xi_{m+1}) \wedge^* \ldots \wedge^* \neg^* (\Xi_{m+n}) \\
& \wedge^* \{(G_{C_1}, H_{C_2})\} \\
& \Xi = \pi^*_{Var(B)}(\Theta) \\
& (B, \Xi) \textit{ is not subsumed by } I\ \}
\end{aligned}
$$

where G_{C_1} is a periodicity graph corresponding to C_1, H_{C_2} is a gap graph corresponding to C_2 and $Var(B)$ denotes the set of variables in atom B. The nodes of the periodicity and gap graphs are renamed using the variable names in the associated atoms of the clauses.

If we restrict our attention to stratified (or locally stratified) [8] Datalog$^{\mathbf{not},\equiv Z,<Z}$ programs the following procedure based on the fixpoint iteration method can be used to evaluate programs.

Algorithm A.1 (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.*[10]

$I := \emptyset$
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 A.1 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 [5]. However, it is easily shown [1] that if gap-order constraints are on a finite subset of the integers, Algorithm A.1 terminates returning a non-ground representation of the unique (ground) model of the program.

[10] P_i contains rules of strata i, $i = 1, \ldots, n$.