**CERIAS Tech Report 2003-01**

# GENERALIZED TEMPORAL ROLE BASED ACCESS CONTROL MODEL (GTRBAC) PART II *Expressiveness and Design Issues*

by James B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

# Generalized Temporal Role Based Access Control Model (GTRBAC) Part II *Expressiveness and Design Issues*

James B. D. Joshi[#], Elisa Bertino[*], Usman Latif[@],Arif Ghafoor[#]

[#]CERIAS and School of Electrical and Computer Engineering,
[@]CERIAS and Department of Computer Science,
Purdue University, West Lafayette, IN, USA
{*joshij, ghafoor*}*@ecn.purdue.edu,*
*usman@purdue.edu*
[*]Dipartimento di Scienze dell' Informazione, Universita' di Milano,
Milano, Italy
*bertino@dsi.unimi.it*

## Abstract

*The Generalized Temporal Role Based Access Control* (*GTRBAC*) *model introduces a large set of temporal constraint expressions that facilitates the specification of a comprehensive access control policy. However, the issue of its expressiveness has not been investigated earlier. In this paper, we present an exhaustive analysis of the expressiveness of the constructs provided by GTRBAC and prove that the set of constraints is not minimal by showing that there is a subset of GTRBAC constraints that is sufficient to express all access constraints that can be expressed using the full set. We formally present the minimality result for the GTRBAC constraint set and argue that, although the complete set of constraints in GTRBAC is not minimal, having such an extensive set is advantageous from the perspective of user convenience and the lower complexity of constraint representation. Based on our analysis, we present a set of design guidelines that can considerably enhance security management.*

**Index Terms:** access control, role based, temporal constraint, access policy.

*Portions of this work were supported by the sponsors of the Center for Education and Research in Information Assurance and Security* (CERIAS)

## 1 Introduction

Role based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access control (DAC and MAC) models [7, 8, 9, 15, 18], which have some inherent limitations [9]. Several beneficial features such as policy neutrality, support for least privilege, efficient access control management, are associated with RBAC models [6, 10, 18]. Such features make RBAC better suited for handling access control requirements of diverse organizations. Furthermore, the concept of role is associated with the notion of functional roles in an organization, and hence RBAC models provide intuitive support for expressing organizational access control policies [6]. RBAC models have also been found suitable for addressing security issues in the Internet environment [2, 9, 16], and show promise for newer heterogeneous multidomain environments that raise serious concerns related to access control across domain boundaries [10].

One of the important aspects of access control is that of time constraining accesses to limit resource use. Such constraints are essential for controlling time-sensitive activities that may be present in various applications such as workflow management systems (WFMSs), where various workflow tasks, each having some timing constraints, need to be executed in some order. Use of RBAC has been found very suitable for such workflow applications [4]. To address general time-based access control needs, Bertino *et al*. propose a Temporal RBAC model (TRBAC), which has been generalized recently in the related paper [11]. The Generalized-TRBAC (GTRBAC) model recognizes that temporal constraints are an important feature that orthogonally applies to all aspects of a role system, that is, to role themselves, to permissions assigned to roles, and to role use permissions given to users. GTRBAC thus incorporates a set of language constructs for the specification of various temporal constraints on roles, including constraints on their activations as well as on their enabling times, user-role assignments and role-permission assignments. In particular, GTRBAC makes a clear distinction between role *enabling* and role *activation*. A role is *enabled* if a user can acquire the permissions assigned to it, but no one has done so. An *enabled* role becomes *active* when a user acquires the permissions assigned to the role in a session. By contrast, a *disabled* role cannot be activated by any user. Therefore, constraints on enabling/disabling roles specify when roles can actually be used or not used by subjects. An open issue in the GTRBAC model is that of its expressiveness. It is important to understand if its constraint set is minimal and if it is not then how it can be beneficial from a practical perspective.

In the related paper [11], we presented a generalized TRBAC model and addressed issues related to specification and modeling. We showed through various examples from real world applications how GTRBAC's temporal constraint expressions fulfill diverse access control requirements. Furthermore, in [12], we introduce the various kinds of temporal hierarchy that can exist in a GTRBAC model and analyze how hierarchy-related features differ in different types of hierarchy. The different types of hierarchy we have identified are: the *I*-hierarchy that only allows permission-inheritance semantics; the *A*-hierarchy that allows only activation semantics; and *IA*-hierarchy that allows both inheritance and activation semantics. Furthermore, a notion of *AC*-equivalence is introduced in [12], that establishes applicability of one hierarchy type to replace another hierarchy type. In particular, it has been shown that not all hierarchies in which different hierarchical relations co-exist are *AC*-equivalent. Our formal analysis in [12] further shows that in presence of timing constraints on various entities, the separation of the *permission-usage* and the *role-activation* semantics provides a basis for capturing various inheritance semantics of role hierarchies. We show that these hierarchies can further be divided into sub-types, to account for the subtle effects of temporal constraints.

However, a key issue that has not been addressed so far is that of the expressiveness of the GTRBAC model. It is also relevant to investigate how its structure and semantics can be used in devising proper constraint design guidelines for efficiently expressing access policies. It is important to determine if having so many constraints in the GTRBAC model is at all beneficial from the practical access control policy design perspective. In other words, it is necessary to first investigate if there is a smaller set of constraints that have the same expressive power as the current set of constraints do, and then to understand whether the exhaustive set of GTRBAC constraints is beneficial for practical applications.

In this paper we formally address key issues related to the expressive power of the GTRBAC model and its impact on the design of access constraints. As a novel contribution of this paper, we show through extensive analysis that GTRBAC is in fact not minimal and there exists a subset of GTRBAC constraints consisting of only temporal constraints on role enabling and activation constraints that have the same expressive power as does the complete set of GTRBAC constraints. However, we show that using such a minimal constraint set for specifying all types of constraint requirements may not be efficient and intuitive. Based on such results, we argue that the current set of constraints is much more flexible in terms of the user-convenience and simpler constraint representation. Based on our analysis, we provide a set of design guidelines that is aimed towards improving efficient and convenient use of various constraints to represent the overall access control policies.

The paper is organized as follows. In section two, we briefly present the constraint model of GTRBAC. In section three, we discuss issues related to permission inheritance and role activation in role hierarchies under the GTRBAC model. In section four, we analyze the expressiveness of the GTRBAC model and present the minimality results as well as the constraint design guidelines based on them. We then present related work in section five and our conclusions in section six.

## 2 Generalized Temporal Access Control Model (GTRBAC)

The GTRBAC model proposed in [11] is an extension of the TRBAC model [5]. The model introduces the separate notion of role enabling and role activation and provides constraints and event expressions associated with both. An enabled role indicates that a valid user can activate it, whereas an activated role indicates that at least one user has activated it in a session. Constraints in GTRBAC allows the specification of the following:

1. *Temporal constraints on role enabling/disabling*: These constraints allows one to specify the intervals and durations in which a role is enabled. When a role is enabled, the permissions assigned to it can be acquired by a user by activating it. When duration is specified, the enabling/disabling of a role is initiated by a constraint enabling expression that may be separately specified at run-time by an administrator or by a trigger.

2. *Temporal constraints on user-role and role-permission assignments*: These constraints provide constructs to express either a specific interval or a duration in which a user or a permission is assigned to a role.

3. *Activation constraints*: These constraints allow one to specify how a user should be restricted in the actual activation of a role. These include, for example, specifying what is the total duration a user is allowed to activate a role, how many users can be allowed to activate a particular role, etc.

4. *Run-time events*: A set of run-time events allows an administrator to dynamically initiate GTRBAC events, or enable duration or activation constraints. Another set of run-time events allow users to make activation requests to the system

5. *Constraint enabling expressions*: GTRBAC includes events that enable or disable duration constraints and role activation constraints. The duration constraints may be on role enablings, user-role assignments or role-permission assignments.

6. *Triggers*: Triggers allow expressing dependencies among GTRBAC events.

Table 1 summarizes the constraint types and expressions of the GTRBAC model. The GTRBAC model extends the safety notion of the TRBAC model to show that there exists an execution model for it. The periodic expression of form (*I*, *P*) used in the constraint expressions are based on those in [3, 14]. The function *Sol*(*I*, *P*) as defined in [5] is used to determine all the time instants denoted by the interval expression (*I*, *P*). *D* expresses the duration specified for a constraint. For more details, we refer the readers to [11].

Table 1. Constraint Expressions

| Constraint categories | Constraints | | Expression | Set/Type |
|---|---|---|---|---|
| *Periodicity Constraint* | User-role assignment | | (I, P, pr:$\texttt{assign}_U/\texttt{deassign}_U$ *r* to *u*) | $C_{Urp}$ |
| | Role enabling | | (I, P, pr:$\texttt{enable/disable}$ *r*) | $C_{Rp}$ |
| | Role-permission assignment | | (I, P, pr:$\texttt{assign}_P/\texttt{deassign}_P$ *p* to *r*) | $C_{PRp}$ |
| *Duration Constraints* | User-role assignment | | ([(I, P)\| D], $D_U$, pr:$\texttt{assign}_U/\texttt{deassign}_U$ *r* to *u*) | $C_{Urd}$ |
| | Role enabling | | ([(I, P)\| D], $D_R$, pr:$\texttt{enable/disable}$ *r* ) | $C_{Rd}$ |
| | Role-permission assignment | | ([(I, P)\| D], $D_P$, pr:$\texttt{assign}_P/\texttt{deassign}_P$ *p* to *r*) | $C_{PRd}$ |
| *Duration Constraints on Role Activation* | Total active role duration | Per-role | ([(I, P)\| D], $D_{active}$, [$D_{default}$], $\texttt{active}_{R\_total}$ *r*) | $C^a{}_{dr}$ |
| | | Per-user-role | ([(I, P)\| D], $D_{uactive}$, *u*, $\texttt{active}_{UR\_total}$ *r*) | $C^a{}_{dur}$ |
| | Max role duration per activation | Per-role | ([(I, P)\| D], $D_{max}$, $\texttt{active}_{R\_max}$ *r* ) | $C^a{}_{mr}$ |
| | | Per-user-role | ([(I, P)\| D], $D_{umax}$, *u*, $\texttt{active}_{UR\_max}$ *r*) | $C^a{}_{mur}$ |
| *Cardinality Constraint on Role Activation* | Total no. of activations | Per-role | ([(I, P)\| D], $N_{active}$, [$N_{default}$], $\texttt{active}_{R\_n}$ *r* ) | $C^a{}_{nr}$ |
| | | Per-user-role | ([(I, P)\| D], $N_{uactive}$, *u*, $\texttt{active}_{UR\_n}$ *r*) | $C^a{}_{nur}$ |
| | Max. no. of concurrent activations | Per-role | ([(I, P)\| D], $N_{max}$, [$N_{default}$], $\texttt{active}_{R\_con}$ *r*) | $C^a{}_{nnr}$ |
| | | Per-user-role | ([(I, P)\| D], $N_{umax}$, *u*, $\texttt{active}_{UR\_con}$ *r*) | $C^a{}_{nmur}$ |
| *Trigger* | $E_1 ,\ldots, E_n , C_1 ,\ldots, C_k \rightarrow pr{:}E$ after $\Delta t$ | | | $C_{tr}$ |
| *Constraint Enabling* | pr:$\texttt{enable/disable}$ *c* where $c \in \{(D, D_x, pr{:}E), (C), (D, C)\}$ | | | $C_c$ |
| *Run-time Requests* | *Users' activation request* | | (s:(de)activate *r* for *u* after $\Delta t$)) | $C_u$ |
| | *Administrator's run-time request* | | (pr:$\texttt{assign}_U/\texttt{de-assign}_U$ *r* to *u* after $\Delta t$) | $C_{admin}$ |
| | | | (pr:$\texttt{enable/disable}$ *r* after $\Delta t$) | $C_{admin}$ |
| | | | (pr:$\texttt{assign}_P/\texttt{de-assign}_P$ *p* to *r* after $\Delta t$) | $C_{admin}$ |
| | | | (pr:$\texttt{enable/disable}$ *c* after $\Delta t$) | $C_{admin}$ |

We illustrate with an example the GTRBAC specification of an access control policy. Table 2 contains the GTRBAC specification of a hospital's access policy. Groupings labeled 1, 2, 3, 4 and a, b, c, d are used simplify the discussion.

In 1a, the enabling times of DayDoctor and NightDoctor roles are specified as a periodicity constraint. For simplicity we use *DayTime* and *NightTime* instead of their (*I*, *P*) forms. In 1b, different users are assigned to the two doctor roles. *Adams* can assume the DayDoctor role on

*Mondays*, *Wednesdays* and *Fridays*, whereas *Bill* can assume the DayDoctor role on *Tuesdays*, *Thursdays*, *Saturdays* and *Sundays*. Similarly, *Alice* and *Ben* are assigned to the NightDoctor role on the different days of the week. Furthermore, in 1c, the assignment indicates that *Carol* can assume the DayDoctor role everyday between 10 am and 3pm.

In 2a, users *Ami* and *Elizabeth* are assigned roles NurseInTraining and DayNurse respectively, without any periodicity or duration constraints, that is, the assignment is valid at all times. 2b specifies a duration constraint of 2 *hours* on the enabling time of the NurseInTraining role, but this constraint is valid for only 6 *hours* after the constraint *c*1 is enabled. Because of this, *Ami* will be able to activate the NurseInTraining role at the most for two hours whenever the role is enabled.

Table 2. Example GTRBAC access policy for a medical information System

| | | |
|---|---|---|
| 1 | a. | (*DayTime*, enable DayDoctor), (*NightTime*, enable NightDoctor) |
| | b. | ((M, W, F), assign$_U$ *Adams* to DayDoctor), ((T, Th, S, Su), assign$_U$ *Bill* to DayDoctor),<br>((M, W, F), assign$_U$ *Alice* to NightDoctor), ((T, Th, S, Su), assign$_U$ *Ben* to NightDoctor) |
| | c. | ([10am, 3pm], assign$_U$ *Carol* to DayDoctor) |
| 2 | a. | (assign$_U$ *Ami* to NurseInTraining)<br>(assign$_U$ *Elizabeth* to DayNurse) |
| | b. | *c*1 = (6 *hours*, 2 *hours*, enable NurseInTraining) |
| 3 | a. | (enable DayNurse $\rightarrow$ enable *c*1) |
| | b. | (activate DayNurse for *Elizabeth* $\rightarrow$ enable NurseInTraining after 10 *min*) |
| | c. | (enable NightDoctor $\rightarrow$ enable NightNurse after 10 *min*)<br>(disable NightDoctor $\rightarrow$ disable NightNurse after 10 *min*) |
| | d. | (enable DayDoctor $\rightarrow$ enable DayNurse after 10 *min*)<br>(disable DayDoctor $\rightarrow$ disable DayNurse after 10 *min*) |
| 4 | a. | (10, active$_{R\_n}$ DayNurse) |
| | b. | (5, active$_{R\_n}$ NightNurse) |
| | c. | (2 *hours*, active$_{R\_total}$ NurseInTraining) |

The constraints in 3 are triggers. Trigger 3a indicates that constraint *c*1 is enabled once the DayNurse is enabled, which means now the NurseInTraining role can be enabled within the next 6 *hours*. Trigger 3b indicates that 10 *min* after *Elizabeth* activates the DayNurse role, the NurseInTraining role is enabled for a period of 2 *hours*. This shows that a nurse in training will have access to the system only if *Elizabeth* is present in the system, that is, she is acting as a training supervisor. It is possible that *Elizabeth* activates the DayNurse role a number of times within 6 hours after the DayNurse role is enabled, and hence the NurseInTraining role will be enabled as many times if these activations (by *Elizabeth*) are more than 2 hours apart. This will allow *Ami* to activate the NurseInTraining role a number of times. To prevent this, there is also an activation constraint 4c on the NurseInTraining role restricting its total activation time to 2 *hours*. The remaining triggers in 3 show that the DayNurse and NightNurse roles are enabled (disabled) 10 *min* respectively after the DayDoctor and NightDoctor roles are enabled (disabled). The constraint set 4 shows some activation constraints. 4a says that there can be at most 10 users activating DayDoctor role at a time, whereas 4b shows that there can be at most 5 users activating the NightDoctor role at a time.

In a GTRBAC system, the reservoir of all constraints is termed as its *Temporal Constraint and Activation Base* (*TCAB*). All the constraints of a *TCAB* must be satisfied before a user is authorized to access an object. We note that a *TCAB* does not contain run-time requests as they are events requested by users/administrators at arbitrary times. Thus, we see that a *TCAB* can be represented as a tuple $T = (C_{URp}, C_{Rp}, C_{PRp}, C_{URd}, C_{Rd}, C_{PRd}, C^{a}_{dr}, C^{a}_{dur}, C^{a}_{mr}, C^{a}_{mur}, C^{a}_{nr}, C^{a}_{nur}, C^{a}_{nmr}, C^{a}_{nmur}, C_{tr}, C_{c})$ where each component is a constraint type as depicted in `Table 1.` Here, we use a constraint type name to also refer to the set containing constraints of that type, for example $C_{URp}$ also refers to the set containing the periocitiy constaints on user-role assignments.

## 3 Temporal hierarchies and Inheritance semantics in GTRBAC

In this section, we briefly summarize the relevant background on the temporal hierarchies and various inheritance semantics that are possible in a GTRBAC system. For a more rigorous formal discussion, the reader is referred to [12].

In [12], we identify and formally define three types of temporal hierarchies in a GTRBAC model. These are similar to the hierarchies proposed in [19], which are considered in a different context and do not address temporal issues related to them. The three types of temporal hierarchies are *inheritance-only* hierarchy denoted by $\geq^{t}$ (*I*-hierarchy) , *activation-only* hierarchy denoted by $\succcurlyeq^{t}$ (*A*-hierarchy) and *inheritance and activation* hierarchy denoted by $\succsim^{t}$ ( *IA*-hierarchy ). An *I*-hierarchy only includes permission-inheritance semantics, which allows a senior to inherit all its juniors' permissions. However, it restricts a user assigned to the senior from activating any of the junior roles unless s/he is explicitly assigned to them. An *A*-hierarchy only allows activation semantics, according to which a user assigned to the senior role can also activate its junior roles; however, the user can acquire only the permissions of the activated role. Finally, *IA*-hierarchy includes both the inheritance and activation semantics, thus, allowing a senior role to inherit all its junior roles' permissions, as well as permitting any user assigned to the senior role to also activate the junior roles within the same or different session.

Examples of the three hierarchies are illustrated in Figure 1, where the Software Engineer role is senior to the Programmer role. In Figure 1(a) and 1(b), the combination of roles that a user *u* who is assigned only to Software Engineer role, can activate is {(Software Engineer), (Software Engineer, Programmer) (Programmer)}. However, the permissions associated with the same combination in the two cases are not exactly the same. For example, if *u* activates the Software Engineer role, s/he can acquire permissions of both the roles' if it is an *IA*-hierarchy (Figure 1(a)), whereas, whereas, s/he acquire only Software Engineer role's permissions if it is an *A*-hierarchy (Figure 1(b)). We note that in *IA*-hierarchy, *u* acquires the same set of permissions by activating role combinations (Software Engineer, Programmer) and (Software Engineer), so u can simply activate a single role ((Software Engineer role) if he wishes to acquire the permissions of both the roles.

Under the role hierarchy reported in Figure 1(c), the user can activate only the Software Engineer role unless the user is also explicitly assigned to the Programmer. However, he acquires maximal permissions, that is, permissions assigned to both the roles.

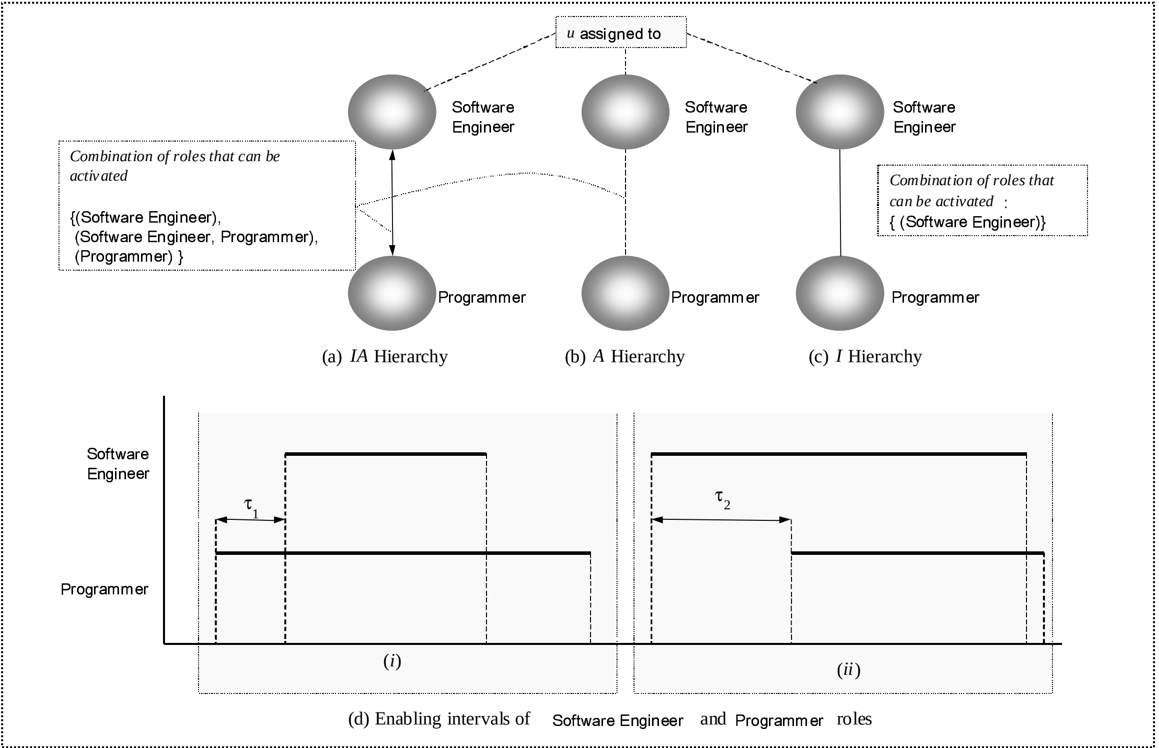


Figure 1. An example hierarchy

Figure 1(d) shows two cases where enabling times of the two roles are different. When enabling intervals of hierarchically related roles differ in such a fashion, we need to consider how we allow inheritance and activation semantics in intervals where only one role is enabled. This issue is briefly discussed in Section 3.1.

## 3.1 Enabling Constraints and Temporal Role Hierarchy

A hierarchy in presence of various temporal constraints becomes dynamic as permissions and users can be assigned/de-assigned to any junior roles at times when a senior role is enabled. Furthermore, there are activation constraints that need to be accounted for when either of the hierarchy types is considered. Here, we consider the effect of the presence of role enabling constraints on both *inheritance* and *activation* hierarchies.

**Inheritance-only hierarchy (*I*-hierarchy)**

Based on the temporal characteristics of hierarchically related roles, such as their enabling times as shown in Figure 1(d), the following two subtypes of an *I*-hierarchy can be identified:

1. *Unrestricted I*-hierarchy ($I_u$-hierarchy): In this case, the permissions of a junior role are inherited by its senior role whenever the senior role is enabled. For example, in interval $\tau_2$ of Figure 1(d)-(*ii*), the Software Engineer role is enabled but the Programmer role is not enabled. However, the unrestricted inheritance semantics permits the Software Engineer role to inherit the permissions of the Programmer role even in this interval.
2. *Restricted I*-hierarchy ($I_r$-hierarchy): Here, the permissions of a junior role are inherited by its senior role only in intervals in which both the roles are enabled. Hence, in interval $\tau_2$ of Figure 1, the Programmer role's permissions are not inherited by the Software Engineer role in interval $\tau_2$.

In Figure 1(d)-(*i*), we see that the enabling interval of Software Engineer role is a subset of that of the Programmer role. In this case, whenever *u* activates the Software Engineer role s/he also acquires the permissions of the Programmer role, because at that time the Programmer role is also enabled. Thus, in interval $\tau_1$, *u* cannot acquire any permissions of the Programmer role even if it is enabled, as the Software Engineer role is disabled at that time.

**Activation-only hierarchy (*I*-hierarchy)**

Similar to an *I*-hierarchy, two subtypes of an *A*-hierarchy can be identified:

1. *Unrestricted A*-hierarchy ($A_u$-hierarchy): In this type of hierarchy, a user assigned to a senior role can activate its junior role at anytime the junior role is enabled. For example, in Figure 1(d)-(*i*), a user assigned to the Software Engineer role can activate the Programmer role even in the interval $\tau_1$, in which the Software Engineer role is actually not enabled.

2. *Restricted A*-hierarchy ($A_r$-hierarchy): In this case, a user assigned to a senior role can activate its junior role only at time intervals in which both are enabled. For example, in interval $\tau_1$, a user assigned to the Software Engineer role cannot activate the Programmer role because the Software Engineer role is disabled in this interval.

In Figure 1(d)-(*ii*), we see that the Programmer role is disabled in interval $\tau_2$. Hence, a user assigned to the Software Engineer role cannot activate it, as a role cannot be activated unless it is enabled first.

**General inheritance hierarchy (*IA*-hierarchy)**

As an *IA*- inheritance incorporates both the *permission inheritance* and *role-activation* semantics of *I*-hierarchy and an *IA*-hierarchy, it also has the same subtypes – an *unrestricted* and a *restricted* form. For instance, in interval $\tau_1$, the *IA*-hierarchy can benefit from the use of role-activation semantics and activate the junior role using the *unrestricted* semantics. Similarly, in interval $\tau_2$, the *inheritance* semantics can be used and inheritance through the senior role using *unrestricted* semantics. The restricted *IA*-hierarchy allows inheritance or activation semantics only in the overlapping enabling intervals of the hierarchically related roles. The restricted and unrestricted forms fo each hierarchy type is summarized in `Table 3`.

`Table 3.` Inheritance semantics

| $r_1$ is senior of $r_2$→ / ↓Hierarchy Type | | $\tau$ <br> $r_1$ `disabled` <br> $r_2$ `enabled` | $\tau$ <br> $r_1$ `enabled` <br> $r_2$ `disabled` |
|---|---|---|---|
| *Inheritance* | $I_u$ | No inheritance in $\tau$ | Inheritance in $\tau$ (*by activating* $r_1$) |
| | $I_r$ | No inheritance in $\tau$ | No inheritance in $\tau$ |
| *Activation* | $A_u$ | Inheritance in $\tau$ (*by activating* $r_2$) | No inheritance in $\tau$ |
| | $A_r$ | No inheritance in $\tau$ | No inheritance in $\tau$ |
| *General Inheritance* | $IA_u$ | Inheritance in $\tau$ (*by activating* $r_2$) | Inheritance in $\tau$ (*by activating* $r_1$) |
| | $IA_r$ | No inheritance in $\tau$ | No inheritance in $\tau$ |

## 3.2 Activation Constraints and Role Hierarchy

In [12], we show that an activation constraint can either be *permission-oriented* or *user-oriented*. If an activation constraint is *permission-oriented*, it implies that the activation constraint is aimed at constraining the use of permissions acquired through the role on which the activation constraint is defined. For example, if a *permission-oriented* cardinality constraint of *n* is defined on a role then it means that the goal is to limit the number of times the associated permissions are used to *n* times. As *A*-hierarchies do not allow a user to acquire junior roles' permissions without explicit activation of the junior roles, *A*-hierarchies are suited for permission-oriented activation constraints. Similarly, an activation constraint can also be *user-oriented*. For example, if the same cardinality constraint is *user oriented*, then it implies that the aim of the constraint is to limit the number of users that can activate at a time. Here, one is not concerned how many times the associated permissions are used. For example, the permissions of a role may be used by different users through different roles of an *I*-hierarchy because of inheritance. Table 4 summarizes the suitability of various kind of hierarchy for activation constraint that are either *user-oriented* or *permission-oriented*.

Table 4. Cardinality constraints and hierarchy

| **Hierarchy** | **Activation constraints** | |
|---|---|---|
| | *User-oriented* | *Permission-oriented* |
| *I-or IA-hierarchy* | Suitable | Not suitable |
| *A-hierarchy* | Not suitable | Suitable |

As the concept of a role as a "*set of permission*" is the most prevalent one, it means that the *permission-oriented* activation constraints lend itself closer to an RBAC model than the *user-oriented* activation constraints. As we can see from the table, an *A*-hierarchy is particularly suitable for such *permission-oriented* activation constraints.

We note that the role-activation semantics actually incorporates some implicit notion of permission inheritance as the actual activation of a junior role allows its permissions to be acquired by a user assigned to its senior. An advantage of *A*-hierarchy is that it allows all combination of roles to be activated in a session. This allows a user to acquire, in a session, all levels of granularity of permissions sets associated with the roles. This is not possible in a hierarchy that has permission-inheritance semantics. This is because an *I*-hierarchy only allows all permissions to be acquired whenever the role that a user is assigned to is activated, such combinations. Similarly, an *I*-hierarchy a combination of roles that contain a senior and a junior is redundant (as shown in the case of Figure 1) in terms of what permissions that the user can acquire. The issue of granularity of permissions set that can be acquired makes *A*-hierarchy most suitable mechanism to address the issue of the principle of least privilege, particularly because dynamic separation of duty can be additionally applied to hierarchically related roles to restrict a particular set of roles to be activated [12, 19]. An obvious disadvantage of an A-hierarchy is that explicit activations each role in a role set is essential to acquire a certain set of permissions. However, such difficulties may be overcome by practical techniques such as identifying role sets that a user usually activate at once and use group names.

An important conclusion in [12] is that all monotype hierarchies (that contains only one hierarchical relation between roles in a hierarchy) are *AC*-equivalent. Two *AC*-equivalent hierarchies allow the same set of maximum permissions to be acquired by a user. It has also been shown in [12], except for a special case of a mixed hierarchy, all other mixed hierarchies (that applies different hierarchical relations between different role pairs of the hierarchy) that most of the mixed hierarchy types are *AC*-equivalent to the monotypes. The special case is the one that has a linear *I*-type hierarchy that preceded an *A*-type hierarchy in a linear chain of hierarchically related roles. We refer to [12], for a detailed formal treatment. The effect of such structure is that the maximal set of permissions has smaller size than that are possible in monotypes.

In the remaining part of the paper, whenever we talk about a GTRBAC system we assume that *A*-hierarchy is used. This is because we believe *A*-hierarchy is theoretically the best representative of all of monotypes as well as most of the mixed types of hierarchies. Furthermore, *A*-hierarchy is much more suitable for the *permission-oriented* activation constraints, which we believe should bear higher emphasis in a GTRBAC system because of the notion of a role as a "*set of permissions*".

Before we present our analysis of the expressiveness of the GTRBAC model, we present an example of A-hierarchy that also includes activation constraints. Figure 3 depicts an example of an activation constraint in a role hierarchy of type $A_r$. Here, MV1, MV2 and MV3 are seniors to role MV. There is a *per-role* total duration activation constraint specified for MV. Furthermore, assume that MV1, MV2 and MV3 are enabled at all times. Thus, at anytime a user assigned to MV1, MV2 or MV3 can activate the MV role. Hence in Figure 2, the total activation duration allowed to the users *A*, *B* and *C*, if they are the only users assigned to the three senior roles, is 600 hours. Furthermore, since the default value is not specified, any of the users *A, B* or *C* may use all 600 hours.

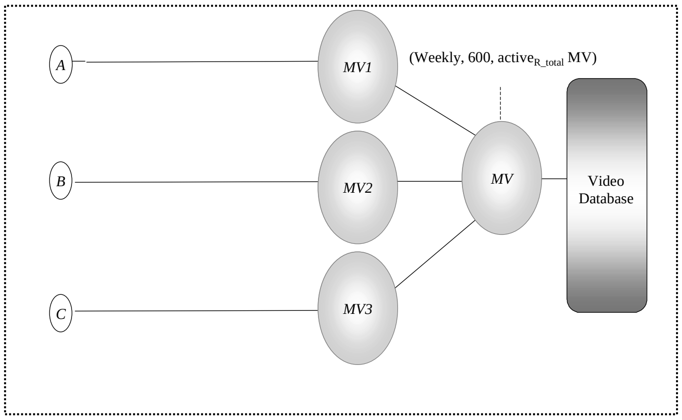


Figure 3. Constraint in a hierarchy

## 4. Expressiveness of GTRBAC and Design Considerations

We have introduced a comprehensive set of temporal constraints in GTRBAC in the related paper [11]. A pertinent question is whether such an exhaustive set of temporal constraints is required at all, or is there a minimal set of constraint types that have the same expressive power as the set containing all the constraint types introduced in this paper. By now, it should be clear that the set of GTRBAC constraint types may not be minimal. In this section, we show formally that the set of GTRBAC constraint types is indeed, not minimal. By introducing the notion of *activity-equivalence* or *a-equivalence*, we show that there exists a minimal set of constraint types that have an expressive power equivalent to the set of all the GTRBAC constraint types. However, we show through an extensive analysis that even though such a minimal set exists, the complete set of GTRBAC constraints provides with better alternatives for representing access constraints. Such alternatives allow one to favor user convenience and lower complexity of representation over the use of the minimal set of constraints.

### 4.1 Minimality of GTRBAC

Given a GTRBAC system, as we have seen earlier, its *TCAB* $T$ can be represented as ($C_{URp}$, $C_{Rp}$, $C_{PRp}$, $C_{URd}$, $C_{Rd}$, $C_{PRd}$, $C^{a}{}_{dr}$, $C^{a}{}_{dur}$, $C^{a}{}_{mr}$, $C^{a}{}_{mur}$, $C^{a}{}_{nr}$, $C^{a}{}_{nur}$, $C^{a}{}_{nmr}$, $C^{a}{}_{nmur}$, $C_{tr}$, $C_{c}$). In the discussion below, we use a shorter version, such as $T = (C_{Rp}, C_{URp})$, when only $C_{Rp}$ and $C_{URp}$ are nonempty sets of constraints. The behavior of a GTRBAC system depends on $T$, the set of users `Users`, the set of roles `Roles`, the set of permissions `Permissions`, and the role hierarchy *RH* (we denote it as $\prec$). Note that for reasons mentioned in Section 3, we consider *RH* to be an activation hierarchy. Therefore, we can use the tuple ($T$, `Users`, `Roles`, `Permissions`, *RH*) to indicate a GTRBAC configuration. For simplicity, we will include only those parameters of a configuration that are essential for the context at hand. For example, we will write ($T$, `Roles`, $RH$) to represent a configuration when the discussion that follows considers an unchanging set of `Users` and `Permissions`. We will also use the notation ($u \underset{t}{\overset{C}{\Rightarrow}} p$) to read "*u acquires permission p at time t under configuration C*". Next, we define the notion of *a-equivalence* between two GTRBAC configurations.

**Definition 4.1.1** (**Activity-equivalence or a-equivalence**): *Given a GTRBAC system with two configurations* $C_1 = (T_{1,}$ `Users`, `Roles`$_1$, `Permissions`, $RH_1)$ and $C_2 = (T_2$, `Users`, `Roles`$_2$, `Permissions`, $RH_2)$ (`Users` *and* `Permissions` *are the same in both the configurations*), *the configurations* $C_1$ *and* $C_2$ *are said to be a-equivalent* (*written as* $C_1 \approx C_2$) *if, for all pairs* ($u$, $p$) *such that* $u \in$ `Users`, $p \in$ `Permissions`, *the following condition holds*: ($u \underset{t}{\overset{C_1}{\Rightarrow}} p$) *iff* ($u \underset{t}{\overset{C_2}{\Rightarrow}} p$). *Furthermore, if* $C_1 \approx C_x$ *and* $C_x \approx C_2$,, *then* $C_1 \approx C_2$ (*transitivity*).

The *a-equivalence* between two configurations of a GTRBAC system indicates that a user can perform the same accesses under the two configurations. Hence, by replacing configuration $C_1$ by $C_2$, we do not change the accesses that are allowed for each individual user.

The GTRBAC system contains many constraint types. We show next that the set of constraint types is not minimal, i.e., some constraint types can be removed without reducing the expressive power of the GTRBAC constraint system. For example, the temporal constraints on assignments can be expressed by using temporal constraint on roles (possibly new ones). Using *a-equivalence* between GTRBAC configurations, we will show that there is a minimal representation that uses only periodicity and duration constraints on roles, and the *per-role* activation constraints. However, we will still need the *default* assignments that simply assign users or permissions to roles without specifying any temporal restriction. Although *default* assignments can be considered as a special case of periodicity constraints, we will consider it a special constraint type (non-temporal constraint) represented by $C_d$.

```
Algorithm Transform1
Input :$C_{in}$; Output : $C_{out}$
1.  $C_{out}$ ={$T'$, Roles', $RH'$}= $C_{in}$={$T$, Roles, $RH$};
2.  FOR each $c = (X, pr$:assign/deassign $p$ to $r) \in T$, where $X$ = {($I, P$), ([($I, P$)|, $D_x$], $D$)} DO
3.      Create a unique role $r_i$;
4.      Replace all occurrences of {$X, pr$:assign/deassign $p$ to $r$} by {$X, pr$:enable/disable $r_i$} in $T'$
5       Add default assignment assign/deassign $p$ to $r_i$ to $T'$
6.      FOR each trigger $TR \in T'$, where $TR$ = "$E_1$ ,…, $E_n$ , $C_1$ ,…, $C_k \rightarrow$ $pr$:$E_{n+1}$ after $\Delta t$" DO
7.        Replace $TR$ by $TR'$ = "$E'_1$ ,…, $E'_n$ , $C'_1$ ,…, $C'_k \rightarrow$ $pr$:$E'_{n+1}$ after $\Delta t$", such that,($i$ =1 to $n$+1, $j$ = 1 to $k$) &
8.            IF ($E_i$== "assign/deassign $p$ to $r$") THEN $E'_i$ = "enable/disable $r_i$" ;
9.            ELSE $E'_i$ = $E_i$ ;
10.           IF ($C_j$ == "assigned/deassigned $p$ to $r$" ) THEN $C'_j$ = "enabled/disabled $r_i$" ;
12.           ELSE $C'_j$ = $C_j$;
13.     //ENDFOR
14.     Roles' = Roles' $\cup$ {$r_i$};
15.     FOR each role $r_j \in$ Roles such that {$r_j \prec r$} DO
16.         $RH' = RH' \cup \{r_j \prec r_i\}$; $RH' = RH' - \{r_j \prec r\}$
17.     //ENDFOR
18.     $RH' = RH' \cup \{r_i \prec r\}$;                          // Note: all are A-hierarchy
19. // ENDFOR
20. return $C_{out}$;
```

Figure 3. Algorithm Transform1

```
Algorithm Transform2
Input : $C_{in}$; Output : $C_{out}$
1.  $C_{out}$ = $C_{in}$ (i.e., {$T'$, Roles', $RH'$}={$T$, Roles, $RH$}); $S = \emptyset$;
2.  FOR each $c = (X, pr$:assign/deassign $u$ to $r) \in T$, where $X$ = {($I, P$), ([($I, P$)|, $D_x$], $D$)} DO
3.      Create a unique role $r_i$;  $S = S \cup (u, r, r_i)$      // function $getSu_i(S, u, r)$ used in line returns $r_i$ r(NIL if eturns
4.      Replace all occurrences of {$X, pr$:assign/deassign $u$ to $r$} by {$X, pr$:enable/disable $r_i$} in $T'$
5       Add default assignment "assign/deassign $u$ to $r_i$ to $T'$ "
6.      FOR each trigger $TR \in T'$, where $TR$ = "$E_1$ ,…, $E_n$ , $C_1$ ,…, $C_k \rightarrow$ $pr$:$E_{n+1}$ after $\Delta t$" DO
7.          Replace $TR$ by $TR'$ where $TR'$ == "$E'_1$ ,…, $E'_n$ , $C'_1$ ,…, $C'_k \rightarrow$ $pr$:$E'_{n+1}$ after $\Delta t$" such that
8.            IF ($E_i$=="assign/deassign $u$ to $r$") THEN $E'_i$ := "enable/disable $r_i$" ;
9.            ELSE $E'_i$ := $E_i$ ;
10.           IF ($C_j$=="assigned/deassigned $u$ to $r$" ) THEN $C'_j$ := "enabled/disabled $r_i$" ;
12.           ELSE $C'_j$ := $C_j$;
13      // ENDFOR
14.     Roles' = Roles' $\cup$ {$r_i$};
15.     FOR each role $r_j \in$ Roles such that {$r \prec r_j$} DO
16.         $RH' = RH' \cup \{r_i \prec r_j\}$; $RH' = RH' - \{r \prec r_j\}$;
17.     //ENDFOR
18.     $RH' = RH' \cup \{r \prec r_i\}$;
19. //ENDFOR
20. // Handle all the per-role-activation constraints
21. FOR each pair ($u, r$) such that there is an activation constraint ($X$, $Y_u$, $u$, active$_{UY}$ $r$) $\in T'$
22.       where    $X \in$ {($I, P$), $D$}, $Y_u \in$ {$D_{uactive}$, $D_{umax}$, $N_{uactive}$, $D_{umax}$}and
23.                active$_{UY}$ = {active$_{UR\_total}$, active$_{UR\_max}$ , active$_{UR\_n}$, active$_{UR\_con}$} DO
24.     IF ($r_i$:=$getSu_i(S, u, r)$ == NIL) THEN Create a unique role $r_i$,     // $getSu_i(S, u, r)$ == NIL means that
25.     FOR each $c$ = ($X$, $Y_u$, $u$, active$_{UY}$ $r$) $\in T'$ DO              // there was no u, r assignment in line 1
26.         Let $c'$ =($X$, $Y_u$, active$_{UY}$ $r_i$);
27.         Replace $c$ in $T'$ by $c'$ where $c'$ =($X$, $Y_u$, active$_{UY}$ $r_i$);   //Note that old c will not be in T'
28.         Replace all occurrences of "enable $c$" by "enable $c'$ "
29      //ENDFOR
30      IF ($r_i$ was created new in Line 24) THEN
31.         Role' = Role' $\cup$ {$r_i$};
32.         FOR each role $r_j \in$ Roles such that {$r \prec r_j$}DO
33.             $RH' = RH' \cup \{r_i \prec r_j\}$; $RH' = RH' - \{r \prec r_j\}$;
34.         //ENDFOR
35.         $RH' = RH' \cup \{r \prec r_i\}$;                      // Note: all are A-hierarchy
36. //ENDFOR
37. return $C_{out}$;
```

Figure 4. Algorithm Transform2

The two algorithms shown in Figures 3 and 4 are used to replace certain constraint types in a configuration by other types to produce an *a-equivalent* configuration. Algorithm `Transform1` takes in a GTRBAC configuration and produces an *a-equivalent* configuration with all the temporal constraints on role-permission assignments removed. Similarly, the algorithm `Transform2` produces a new configuration that is *a-equivalent* to the input configuration $C_{in}$, with all user-role assignments and *per-user-role* activation constraints removed. The following two lemmas formally show that the transformation done by each algorithm is correct.

**Lemma 4.1.1 (Correctness of `Transform1`)**: *Given an input configuration* $C_{\text{in}}$*, algorithm* `Transform1` *produces* $C_{\text{out}}$ *such that there are no temporal role-permission assignments in* $C_{\text{out}}$*, and* $C_{\text{in}} \approx C_{\text{out}}$.

**Lemma 4.1.2 (Correctness of `Transform2`)**: *Given an input configuration* $C_{\text{in}}$*, algorithm* `Transform2` *produces* $C_{\text{out}}$ *such that there are no temporal user-role assignments and per-user-role activation constraints in* $C_{\text{out}}$*, and* $C_{\text{in}} \approx C_{\text{out}}$.

We use the following notion of *minimal constraint set* (MCS) to express the fact that there is an *a-equivalent* configuration that has the minimum number of constraint types.

**Definition 4.1.2** (**Minimal Constraint Set**): *Let MCS*($T$) *be the set of constraint types in TCAB* $T$*, and CS* = {$C_1$, $C_2$, .. $C_n$} *be an a-equivalent set of configurations such that* $C_i$ = ($T_i$, `Roles` $_i$, $RH_i$) *for* $i$ = 1, 2, …, $n$*. We say that MCS*($T_i$) *is the* minimal constraint set (MCS) *of CS for* $i \in$ {1, 2, …, $n$}, *if there exists no other configuration* $C_j$ = ($T_j$, `Roles`$_j$, $RH_j$), *such that* $i \neq j$ *and*

$$MCS\,(T_j) \subset MCS\,(T_i).$$

The definition implies that a minimal constraint set is the one that has the least number of temporal constraint types. Note that the role set and hierarchy may be altered to reduce the number of constraint types. We are now ready to present the minimality result for GTRBAC system, which is expressed by the following theorem.

**Theorem 4.1 (Minimality of GTRBAC)**: *Let* $C_1$ *be a* GTRBAC *configuration, such that* $\{C_d, C_{Rp}, C_{Rd}, C^a_r, C_{tr}, C_c\} \subseteq MCS(T_1)$; *There exists a* GTRBAC *configuration* $C_2$ *such that*:

a. $C_1 \approx C_2$, and

b. *$MCS(T_2)$ $=\{C_d, C_{Rp}, C_{Rd}, C^a{}_r, C_{tr}, C_c\}$, (note that $C^a{}_r$ represents per-role activation constraint types), and*

c. *$MCS$ ($T_2$) is a minimal constraint set,*

Theorem 4.1 shows that the set of GTRBAC constraints is not minimal because, a set of default assignments, periodic and duration constraints on role enabling (disabling), and *per-role* activation constraints can be used to represent any access constraint that GTRBAC constraints can represent. We can see from the transformation algorithms that replacing temporal constraints on assignments by temporal constraints on roles, in general, increases the number of roles and the complexity of role hierarchy, which may not be desirable. This is because algorithms `transform1` and `transform2` creates a new role for each temporal assignment that they replace. This may not be very intuitive and efficient as it means there will be as many new roles as there are temporal assignments. This results in a worst case where a role is created for each user (or permission) in the system. A more intuitive and practical approach would be to create a least number of roles such that the enabling/disabling intervals for them are non-overlapping. For example, if there is a Doctor role and each of the *n* users are assigned to it for either day time or night time (or both), then, instead of creating *n* new roles, we can simply create DayDoctor and NightDocor roles and assign all the *n* users to one or the other (or both). Thus, to create such temporally non-overlapping roles, we must first divide *n* periodic expressions into temporally non-overlapping set of periodic expressions such as, *Daytime* and *Nighttime*.

We next provide formal definitions and algorithms to generate such a disjoint set of roles that replace a set of temporal assignments. We first introduce the formal notions of *containment, equivalence, overlapping* and *disjunction* between a pair of periodic expressions. Note that, an arbitrary set of intervals can be represented by a periodic expression. This is possible because, each such expression can be formulated, at the worst as a periodic expression that lists every starting point and the smallest calendar as a duration.

**Definition 4.2.1 (Containment/Equivalence/Overlapping/Disjunction of periodic Expressions) :** *Let $PE_1$=($I_1$, $P_1$) and $PE_2$=($I_2$, $P_2$) be two periodic expressions, then*

1. *$PE_1$ is said to be* contained *in $PE_2$ (written as $PE_1 \subseteq PE_2$), if the following conditions hold*

$$\text{for all } t, (t \in Sol(I_1, P_1) \rightarrow t \in Sol(I_2, P_2));$$

2. *$PE_1$ and $PE_2$ are said to be* equivalent (*written as: $PE_1 = PE_2$*) *if*

   $$(PE_1 \subseteq PE_2) \wedge (PE_2 \subseteq PE_1);$$

3. *$PE_1$ and $PE_2$ are said to be* overlapped (*written as $PE_1 \otimes PE_2$*) *if the following condition holds*:

   $\exists\, t_1, t_2$ *such that*

   - $(t_1, t_2) \in \Pi(P_1)$, i.e. $t_1, t_2$ *are end points of an interval* in $P_1$, *and*
   - $\exists\, t_a, t_b$,
     - $t_1 < t_a < t_b < t_2$, *and*
     - $(t_a, t_b \in Sol(I_1, P_1) \rightarrow (t_a \in Sol(I_2, P_2) \wedge t_b \notin Sol(I_2, P_2)) \vee (t_b \in Sol(I_2, P_2) \wedge t_a \notin Sol(I_2, P_2)));$

4. *$PE_1$ and $PE_2$ are said to be* disjoint (*written as $PE_1 \lozenge PE_2$*), *if, for all $t_1$, $t_2$ such that* $(t_1 \in Sol(I_1, P_1) \wedge t_2 \in Sol(I_2, P_2)$, *the following condition holds*:

   $$((t_1 \in Sol(I_2, P_2) \rightarrow t_1 \in ESol(I_2, P_2)) \wedge (t_2 \in Sol(I_1, P_1) \rightarrow t_2 \in ESol(I_1, P_1)),$$

   *where $ESol(I, P)$ is the set of end-points of intervals in $(I, P)$ such that if $t \in ESol(I, P)$ then $t \in Sol(I, P)$.*

A set of periodic expressions is said to be *disjoint* if the period expressions are pair-wise disjoint, otherwise it is said to be *non-disjoint*. Similarly, a set of periodic expressions is said to be *equivalent* if all the period expressions are equivalent to each other.

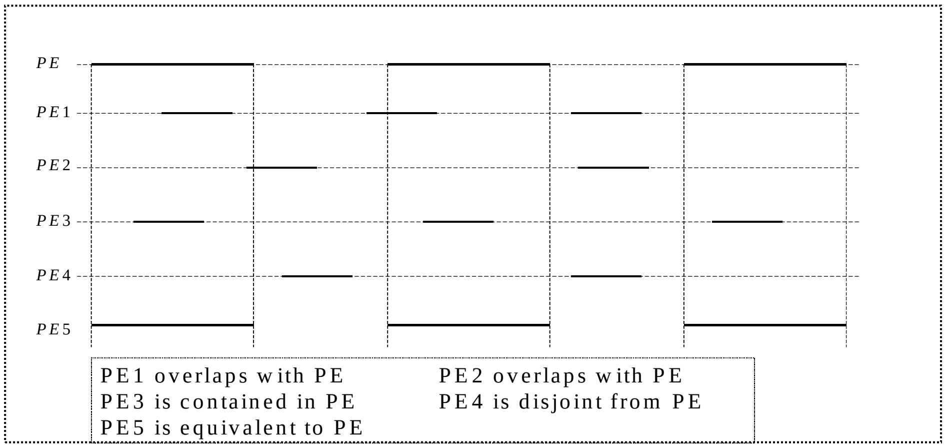


Figure 5. Temporal relations between a pair of periodic expressions

Figure 5 shows some examples of these relations. Note that the fourth part of the definition implies that if only endpoints of intervals of two periodic expressions are common, then they are considered disjoint. Ideally, we want to compute a disjoint set of periodic expressions that is minimal. The next definition expresses the notion of *minimal disjoint set* (MDS) over a set of periodic expressions.

**Definition 4.2.2 (Minimal Disjoint Set)**: *Le $PE = \{PE_1, PE_2, \ldots, PE_n\}$ be a set of arbitrary periodic expressions. The* minimal disjoint set (MDS) over *PE is the least set of* disjoint *periodic expressions, $MDS_{PE}$, defined as*:

$$MDS_{PE} = \min_m\{PE'_i \mid 1 \le i \le m\},$$

*such that the following conditions hold,*

1. $PE'_1 \cup PE'_2 \cup \ldots \cup PE'_m = PE_1 \cup PE_2 \cup \ldots \cup PE_n$,
2. *If there exists a $t_1$, such that* $(t_1 \in Sol(PE_j) \wedge t_1 \in Sol(PE'_i))$ for $1 \le i \le m$, $1 \le j \le n$, then $\forall t_2, (t_2 \notin Sol(PE_j) \rightarrow t_2 \notin Sol(PE'_i)))$

In the definition, the first condition says that the MDS contains periodic expressions containing all time instants that are contained in all the original set of periodic expressions $PE_i$s. The second condition ensures that if a time instant from a $PE_j$ is in $PE'_i$ then the time instants that are not in $PE_j$ will not be in $PE'_i$. In other words, each $PE'_i$ is contained in one or more $PE'_j$. Associated with MDS, we define *minimal subset* (MS) *of a periodic expression over a* MDS as follows.

**Definition 4.2.3 (Minimal subset (MS) for a periodic expression over a MDS)**: *Let $MDS_{PE} = \min_m\{PE'_i \mid 1 \le i \le m\}$ be a minimal disjoint set over periodic expressions $PE = \{PE_1, PE_2, \ldots, PE_n\}$; The* minimal subset (MS) for a periodic expression $PE_j \in PE$ over the $MDS_{PE}$ *is the set* $MS_{PE_j}(MDS_{PE}) = \{PE'_{\pi 1}, PE'_{\pi 2}, \ldots, PE'_{\pi k}\} \subseteq MDS_{PE}$, $1 \le k \le m$ *such that,*

- $\{\pi 1, \pi 2, \ldots, \pi k\} \subseteq \{1, 2, \ldots, m\}$ for $1 \le k \le m$ , and
- *for all t,* $t \in Sol(PE_j)$ *iff* $(t \in Sol(PE'_{\pi 1}) \vee t \in Sol(PE'_{\pi 2}) \vee \ldots \vee t \in Sol(PE'_{\pi k}))$.

We see that MS of a periodic expression $PE_i$ of $PE$ is a subset of $MDS_{PE}$ that collectively contains all the time instants of $PE_i$. Before presenting an example for MDS and MS, we first show some formal properties related to the computation of MDS and MS. We write ${}^{i}MDS_{PE}$ to mean MDS of the first *i* periodic expressions of *PE*.

```
Algorithm PairMDS
Input: $PE_1$, $PE_2$
Output: MDS of $PE_1$, $PE_2$
IF ($PE_1 = PE_2$) THEN return{$PE_1$};
IF ($PE_1 \lozenge PE_2$) THEN return{$PE_1$, $PE_2$};
IF ($PE_1 \subseteq PE_2$) THEN  // as per Lemma 4.2.1(a)
    $PE_x = PE_1$;
    $PE_y = PE_2 - PE_x$;
    return{$PE_x$, $PE_y$};
IF ($PE_2 \subseteq PE_1$) THEN  // as per Lemma 4.2.1(a)
    $PE_x = PE_2$;
    $PE_y = PE_1 - PE_x$;
    return{$PE_y$, $PE_x$};
IF ($PE_1 \otimes PE_2$) THEN // as per Lemma 4.2.1(b)
    $PE_x = PE_1 \cap PE_2$;
    $PE_y = PE_2 - PE_x$;
    $PE_z = PE_1 - PE_x$;
    return{$PE_x$, $PE_y$, $PE_z$}
//end
```

```
Algorithm ComputeMDS
Input: $PE_1$, $PE_2$, …, $PE_n$
Output: MDS of $PE_1$, $PE_2$, …, $PE_n$
// Assume that $PE = \{PE_1, PE_2, …, PE_n\}$
$S = \emptyset$; $MDS = \emptyset$;
IF $|PE| = 1$ THEN return $PE$;
IF $|PE| = 2$ THEN return PairMDS($PE_1$, $PE_2$);
IF $|PE| > 2$ THEN
    $MDS$ = ComputeMDS($PE_1$, $PE_2$, …, $PE_{n-1}$);
    Let $MDS$ computed be ($PE'_1$, $PE'_2$, …, $PE'_{m1}$);
    FOR $i$ = 1 to $m1$ DO
        $PairMDS$ = PairMDS($PE_i$, $PE_n$);
        IF $|PairMDS| = 1$ THEN
            return $MDS$;
        IF $|PairMDS| = 2$ THEN
            Let $PairMDS$ computed be ($PE'_x$, $PE'_y$);
            $S = S \cup \{PE'_x\}$;
        ELSEIF  $|PairMDS| = 3$ THEN
            Let $PairMDS$ be ($PE'_x$, $PE'_y$, $PE'_z$);
            $S = S \cup \{PE'_x, PE'_z\}$;
    // ENDFOR
    Let S computed be ($PE''_1$, $PE''_2$, …, $PE''_{m2}$);
    $PE''_{m2+1} = PE_n - (PE''_1 \cup PE'' \cup … \cup PE''_{m2})$;
    IF ($PE''_{m2+1} = \emptyset$) THEN
        $MDS = (PE''_1, PE''_2, …, PE''_{m2}, PE''_{m2+1})$;
    ELSE
        $MDS = (PE''_1, PE''_2, …, PE''_{m2})$;
    return $MDS$
//END IF
```

Figure 6. Algorithms PairMDS  and computeMDS

**Lemma 4.2.1 (MDS for two expressions)**: *Let ($PE_1$, $PE_2$) be a pair of non-equivalent and non-disjoint periodic expressions; The following holds true*:

a. *if* ($PE_i \subseteq PE_j$) *then, for* $(i, j) \in \{(1, 2), (2, 1)\}$, *there exist periodic expressions* $PE_x$, $PE_y$ *such that* $MDS_{PE} = \{PE_x, PE_y\}$. *Furthermore,* $PE_x = PE_i$ and $PE_y = PE_j - PE_i$.

b. *if* ($PE_i \otimes PE_j$) *then for* $(i, j) \in \{(1, 2), (2, 1)\}$, *there exist periodic expressions* $PE_x$, $PE_y$, $PE_z$ *such that* $MDS_{PE} = \{PE_x, PE_y, PE_z\}$. *Furthermore,* $PE_x = PE_i \cap PE_j$, $PE_y = PE_j - PE_x$ *and* $PE_z = PE_i - PE_x$.

Algorithm PairMDS computes MDS for a pair of periodic expressions. We note that when the two expressions are equivalent, the MDS contains a single periodic expression, which can be either of the original expression. Similarly, when the expressions are disjoint, the MDS contains both the periodic expressions. Algorithm computeMDS repeatedly calls PairMDS and recursively builds the MDS by first finding the MDSs of smaller sizes. It uses the inductive technique used to prove Lemma 4.2.2. The following formal results show that `computeMDS` computes the MDS of a set of periodic expressions.

**Lemma 4.2.2 (MDS for n periodic expressions):** *Given a non-equivalent and non-disjoint set of periodic expressions* $PE = \{PE_1, PE_2, ..., PE_n\}$, *there exist periodic expressions* $PE'_1, PE'_2, ..., PE'_m$ *such that* $MDS_{PE} = \{PE'_1, PE'_2, ..., PE''_m\}$.

**Theorem 4.2 (MDS using `computeMDS`):** *Given an arbitrary set of periodic expressions* $PE = \{PE_1, PE_2, ..., PE_n\}$, *there exist periodic expressions* $PE'_1, PE'_2, ..., PE'_m$, *such that*

a. $MDS_{PE} = \{PE'_1, PE'_2, ..., PE'_m\}$ *and*
b. *For PE as input, algorithm* `computeMDS` *produces* $MDS_{PE}$

Theorem 4.2 shows that we can construct a MDS of an arbitrary set of periodic expressions. As we will show later, this will help us in finding a minimum set of roles corresponding to a set of periodic expressions such that they are minimal and disjoint in terms of their enabling intervals. We also derive the following two corollaries.

**Corollary 4.2.1 (Bounds for size of MDS)**: *Given a set of periodic expressions* $PE = \{PE_1, PE_2, ..., PE_n\}$, *the algorithm* `computeMDS` *produces* $MDS_{PE} = \{PE'_1, PE'_2, ..., PE'_m\}$ *such that if* $s_n = |MDS_{PE}|$ *then* $1 \le s_n \le (2^n - 1)$.

**Corollary 4.2.2 (Bounds for size of MS)**: *Given a set of periodic expressions* $PE = \{PE_1, PE_2, ..., PE_n\}$ *and* $MDS_{PE} = \{PE'_1, PE'_2, ..., PE'_m\}$ *produced by algorithm* `computeMDS`, *if* $p_n = |MS_{PE1}| + |MS_{PE2}| + \ldots + |MS_{PEn}|$, *then* $n \le p_n \le n2^{n-1}$.

We illustrate the notion of MDS and MS, and the computation of MDS by algorithms `computeMDS` and `pairMDS` with the following example.

Thus, we see that one way to replace a periodic constraint on assignments is by the technique used by the algorithms `transform1` and `transform2`. Another technique is to compute the MDS and create a role corresponding to each of its intervals. Such a technique is used in algorithm `TransformMDS`, as shown in Figure 8, which only replaces the *user-role* assignment constraints by new roles and constraints on them. Theorem 4.3 shows that the algorithm correctly produces a configuration without *user-role* periodicity constraints that is *a-equivalent* to the input configuration.

`Example 4.1`: To simplify notation, we consider the *Daytime* of the days listed for a periodic expression. For example, if $PE$ = {Sun}, we mean the interval (9am, 9pm) or daytime of a Sunday. Let $PE_A$ = {Sun, Mon, Tue, Wed, Thu, Fri}, $PE_B$ = {Sun, Tue}, $PE_C$ = {Sun, Tue, Thu, Fri}, $PE_D$ = {Sun, Mon, Tue, Wed, Sat}, $PE_E$ = {Thu, Fri}. The following steps illustrate the computation of $MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}}$ using algorithm `computeMDS`.

1. $MDS_{\{PEA,\ PEB\}}$ = {$PE'_1$, $PE'_2$} = {{Sun, Tue}, {Mon, Wed, Thu, Fri}} (as $PE_B \subseteq PE_A$)

2. $MDS_{\{PEA,\ PEB,\ PEC\}}$ = MDS of { $PE'_1$, $PE'_2$, $PE_C$} = MDS of {{Sun, Tue}, {Mon, Wed, Thu, Fri}, {Sun, Tue, Thu, Fr}}

   Here,

   - MDS of ($PE'_1$, $PE_C$} = {$PE'_{x1}$ ={Sun, Tues}, $PE'_{y1}$ = {Thu, Fr} (as $PE'_1 \subseteq PE_C$)},
   - MDS of {$PE'_2$, $PE_C$}= {$PE'_{x2}$ ={Thu, Fri}, $PE'_{y2}$ ={Sun, Tues}, $PE'_{z2}$= {Mon, Wed}}(as $PE'_2 \otimes PE_C$)
   - $S$ = {$PE'_{x1}$, $PE'_{x2}$, $PE'_{z2}$}
   - $PE'_{x1} \cup PE'_{x2} \cup PE'_{z2}$ = {Sun, Mon, Tues, Wed, Thu, Fri}.
   - $PE''_4 = PE_C$ - ($PE'_{x1} \cup PE'_{x2} \cup PE'_{z2}$) = ∅

   Therefore $MDS_{\{PEA,\ PEB,\ PEC\}}$ = {$PE''_1$, $PE''_2$, $PE''_3$} = {{Sun, Tues}, {Thu, Fri}, {Mon, Wed}}

3. $MDS_{\{PEA,\ PEB,\ PEC,\ PED\}}$ = MDS of {$PE''_1$, $PE''_2$, $PE''_3$, $PE_{\ D}$}

   = MDS of {{Sun, Tues}, {Thu, Fri}, {Mon, Wed}, {Sun, Mon, Tues, Wed, Sat}}
   Here,

   - MDS of {$PE''_1$, $PE_D$} = {$PE'_{x3}$ ={Sun, Tues}, $PE'_{y3}$ = {Mon, Wed, Sat} (as $PE''_1 \subseteq PE_D$ },
   - MDS of {$PE''_2$, $PE_D$} = {$PE'_{x4}$ = {Thu, Fri}, $PE'_{y4}$ = {Sun, Mon, Tues, Wed, Sat}} (as $PE''_2 \lozenge PE_D$ },
   - MDS of {$PE''_3$, $PE_D$} = {$PE'_{x5}$ = {Mon, Wed}, $PE'_{y5}$ = {Sun, Tues, Sat}} (as $PE''_3 \subseteq PE_D$ },
   - S = {$PE'_{x3}$, $PE'_{x4}$, $PE'_{x5}$}
   - $PE'_{x3} \cup PE'_{x4} \cup PE'_{x5}$ = {Sun, Mon, Tues, Wed, Thu, Fri},
   - $PE'''_4 = PE_D$ - ($PE'_{x3} \cup PE'_{x4} \cup PE'_{x5}$) = {Sat};

   Therefore, $MDS_{\{\ PEA,\ PEB,\ PEC,\ PED\}}$ ={$PE'''_1$, $PE'''_2$, $PE'''_3$, $PE'''_4$}
   ={{Sun, Tues}, {Thu, Fri}, {Mon, Wed}, {Sat}}

4. $MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}}$ = MDS of {$PE''_1$, $PE''_2$, $PE''_3$, $PE'''_4$, $PE_E$}

   = MDS of {{Sun, Tues}, {Thu, Fri}, {Mon, Wed}, {Sat}, {Thu, Fri}}

   Since $PE_E = PE''_2$, $MDS_{\{\ PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}}$ = $\text{MDS}_{\{\ PEA,\ PEB,\ PEC,\ PED\ \}}$

   = {$PE'''_1$, $PE'''_2$, $PE'''_3$, $PE'''_4$} = {{Sun, Tues}, {Thu, Fri}, {Mon, Wed}, {Sat}}

Also, we see that,

1. $MS_{PEA}(MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\}})$ = {$PE'''_1$, $PE'''_2$, $PE'''_3$}.
2. $MS_{PEB}(MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\}})$ = {$PE'''_1$}
3. $MS_{PEC}(MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\}})$ = {$PE'''_1$, $PE'''_2$}.
4. $MS_{PED}(MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}})$ = {$PE'''_1$, $PE'''_3$, $PE'''_4$}.
5. $MS_{PEE}(MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}})$ = {$PE'''_2$}.

Figure 7. Example of minimal disjoint set (*MDS*) and minimal subset (*MS*)

```
Algorithm TransformMDS
Input    : $C_{in}$
Output   : $C_{out}$
1.   $C_{out}$ = {T', Roles', RH'} = $C_{in}$ = {T, Roles, RH};
2.   FOR each $r \in$ Roles DO
3.       Let $PE = \{PE_1, PE_2 \ldots, PE_n\}$ and $U = \{u_1, u_2 \ldots, u_n\}$ be such that ($PE_i$, assign $r$ to $u_i$) $\in T'$;
4.       Compute MDS of PE;  Let the computed $MDS = \{PE'_1, PE'_2 \ldots, PE'_n\}$;
5.       FOR $i = 1$ to $n$ DO
6.           Compute $MS_{PEi}$ for $PE_i$
7.       //ENDFOR
8.       FOR each $PE'_i \in MDS$ DO
9.           Create a unique role $r_i$;
10.          FOR all $u_k \in U$ such that $PE'_i \in MS_{PEk}$ for $PE_k$ DO
11.              Add default assignment (assign $r_i$ to $u_k$) in T'.
12.              Add constraint ($PE_i$, enable $r_i$ ) in T'.
13.              Remove constraint ($PE_i$, assign $r$ to $u_i$)  from T';
14.              Roles' = Roles' $\cup \{r_i\}$;
15.          $RH' = RH' \cup \{r \prec r_i\}$;                    // Note: all are A-hierarchy
16           //ENDFOR
17       //ENDFOR
18.  //ENDFOR
```

Figure 8. Algorithm TransformMDS

**Theorem 4.3 (Correctness of TransformMDS)**: *Given an input configuration $C_{in}$ with only periodicity constraint on user-role assignments, algorithm* TransformMDS *produces a configuration $C_{out}$ such that the following holds:*

1. *$C_{in} \approx C_{out}$, and*

2. *$C_{out}$ has no periodicity user-role assignment constraints.*

Here, we considered only the presence of the periodicity constraints on *user-role* assignment. If we allow the presence of *per-role* constraints, algorithm TransformMDS will still work as the original roles are simply retained in their original form (along with any *per-role* constraints on them).

## 4.2 System Complexity and Design considerations

The complexity of a GTRBAC system may have different components. Foremost among them is the number of roles. Typically, we do not want an unmanageable number of roles in a system. Another component is the number of temporal constraints. Then we have the complexity incurred by a hierarchy. Finally, we have the *default assignments* (non-temporal). In *default assignments*, the only check needed is the membership check, for example, to determine whether a particular user is assigned to a role or not. Thus, we can expect temporal assignments to introduce additional complexity compared to an RBAC system without temporal constraints because it involves, besides checking for membership, ensuring the temporal validity of a membership. To simplify our discussion on trade-offs and complexity issues, we first develop a family of GTRBAC models that have equivalent expressive power, based on the results in the previous section, and then investigate the potential benefits of a model at a higher level of family hierarchy over those at the lower level.

The *minimality* result in the previous section shows that the minimal model of GTRBAC system is the one that includes the following temporal constraints: *per-role* activation constraint, *periodicity* and *duration* constraints for role-enabling/disabling, constraint enabling and triggers, as shown in Table 5. Figure 9 shows the *minimal model* as $GTRBAC_0$ at level 0. At level 1, we have three different models, each of which adds a new type of constraint to the constraint set of $GTRBAC_0$. $GTRBAC_{1,A}$ represents the model having all the temporal constraints of $GTRBAC_0$ plus the *per-user-role* activation constraints. Similarly, $GTRBAC_{1,U}$ represents the model having all the temporal constraints of $GTRBAC_0$ plus the *user-role* assignment constraints, whereas, $GTRBAC_{1,P}$ represents the model having all the temporal constraints of $GTRBAC_0$ plus the *role-permission* assignment constraints. At level 2, we have the overall $GTRBAC_2$ model that contains all the temporal constraints. We note that we can have other models between Level 1 and Level 2 that represent models representing the pairs of level 1 models. However, for our analysis, we adopt this simpler hierarchy. We also keep in mind that, according to the results in the previous section, all the models in Figure 9 have the same expressive power, i.e, these models can be used to generate *a-equivalent* configurations.

| Level | Model | Constraint Set |
|---|---|---|
| 2 | $GTRBAC_2$ | $T = T_{1,A} \cup T_{1,U} \cup T_{1,P}$ |
| 1 | $GTRBAC_{1,P}$ | $T_{1,P} = T_0 \cup \{C_{PRp}, C_{PRd}\}$ |
|  | $GTRBAC_{1,U}$ | $T_{1,U} = T_0 \cup \{C_{Urp}, C_{Urd}\}$ |
|  | $GTRBAC_{1,A}$ | $T_{1,A} = T_0 \cup \{C^a_{dur}, C^a_{mur}, C^a_{nur}, C^a_{nmur}\}$ |
| 0 | $GTRBAC_0$ *Minimal* | $T_0 = \{C_d, C_{Rp}, C_{Rd}, C^a_r, C_{tr}, C_c\}$ |

Table 5. GTRBAC Family of models and constraint sets

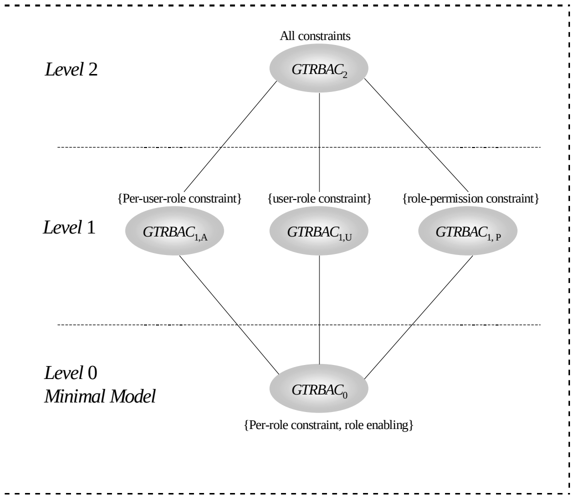


Figure 9. A family of GTRBAC models

Next, we show through analysis that, it is advantageous to use a model at a higher level in terms of *user-convenience* and *complexity of representation*. Our analysis will focus on the advantages and disadvantages of using a Level 1 model compared against that of the Level 0, the *minimal model*.

### 4.2.1 Constraints on Role Enabling and Assignments

We have shown in section 4.1 that all temporal constraints on *user-role* and *role-permission* assignments can be transformed into the temporal constraints on role. However, such a transformation may result in a large number of roles and/or produce inconvenient or complex access control structures. In this section, we look at various design alternatives for choosing constraints on role enablings and assignments. We do this by comparing the complexity of representation using a Level 1 model against those of various representations using the *minimal model* for expressing the same set of access requirements.

As we can see, in `Transform2`, the transformation from temporal constraints on *user-role* assignments to the temporal constraints on roles is the same as that of the transformation from temporal constraints on *role-permission* assignments to the temporal constraints on roles, except for the difference in hierarchy relation. That is, in the first case, the new roles inherit from the old role, whereas in the second case, the old role inherits from the new roles. Because of this similarity, we will mainly focus on the *user-role* assignments, as similar results can be obtained for the *role-permission* assignments. Also, algorithm `Transform2` transforms both the *periodicity* and *duration* constraints in the similar way, i.e., each such constraint is replaced by a new role. Hence, the complexity analysis we apply for *periodicity* constraints will apply for the duration constraints as well. We will, hence, focus on the *periodicity* constraint and point out important considerations related to *duration* constraints whenever they apply.

A temporal constraint on *user-role* assignment states that the user can activate a role in the specified periods or for a specified duration, provided the role is enabled. Instead of using a temporal constraint on *user-role* assignment (the user is still assigned to the role using default assignment), we enforce the desired access control by using temporal constraints on role enabling. Next, we will present the complexity issues related to the representations of a set of access requirements using $GTRBAC_0$ and $GTRBAC_{1,U}$ models. For our purpose, we use the following example

`Example 4.2`: Let us assume that there is a DayDoctor role in a hospital. Five doctors *A, B, C, D*, and *E* are assigned to this role in the periods given by the periodic expressions $PE_A$, $PE_B$, $PE_C$, $PE_D$, and $PE_E$ of `Example 4.1`. We assume that we have the $GTRBAC_{1,\ U}$ representation of these constraints (hence, there are no activation constraints). We will also look at two different representations using $GTRBAC_0$ model, which we will denote as $GTRBAC_0^1$ and $GTRBAC_0^2$ representations.

**$GTRBAC_2$ representation**: For each doctor, a periodicity constraint on his assignment to the DayDoctor role is specified using periodic expressions shown in Figure 10(a). For example, for doctor *A*, $PE_A$ is the periodic expression used – i.e., there is a constraint ($PE_A$, `assign` DayDoctor `to` *A*) in *T*. Similarly, assignment constraints for the remaining doctors with the respective periodic expressions are specified.

**$GTRBAC_0^1$ representation**: In this alternative, we use algorithm `Transform2` with the above $GTRBAC_2$ representation as the input. Accordingly, a role is created for each constraint and a *default* assignment and a periodicity constraint on the new role are added. For instance, for a constraint ($PE_A$, `assign` DayDoctor `to` *A*), a role, say $r_A$, is created and a new constraint ($PE_A$, `enable` $r_A$) is added, whereas the constraint ($PE_A$, `assign` DayDoctor `to` *A*) is replaced by *default* assignment (`assign` DayDoctor `to` $r_A$). Similarly, all other temporal assignments are replaced. This is depicted in Figure 10(b).

**$GTRBAC_0^2$ representation**: This alternative uses the *minimal disjoint* set approach using algorithm `TransformMDS`. The result is as shown in Figure 10 (c). From `Example 4.1`, we know that $MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}}$ = {$PE'''_1$, $PE'''_2$, $PE'''_3$, $PE'''_4$}= {{Sun, Tues}, {Thu, Fri}, {Mon, Wed}, {Sat}}. A role is created for each periodic expression of $MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}}$. As $|MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}}|$ = 4, four new roles are created, and a *periodicity* constraint is added for each new role. The *i*th new role is associated with the *i*th periodic expression of $MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\ \}}$. Each doctor is assigned to a set of new roles that corresponds to the periodic expressions that constitutes *MS* of the periodic expression associated with him, e.g., since $MS_{PEC}(MDS_{\{PEA,\ PEB,\ PEC,\ PED,\ PEE\}})$ = {$PE'''_1$, $PE'''_2$}, doctor *C* is assigned to the new roles that correspond to periodic expressions $PE'''_1$ and $PE'''_2$.

In the discussions that follow, we will use *UR* to refer to temporal *user-role* assignment, *DUR* to refer to default *user-role* assignment, *R* to mean temporal constraint on roles and *H* to represent the overhead associated with hierarchy. In the complexity expressions we will neglect original role and any activation constraints associated with it, as they remain the same in all the representations. We can see that for the $GTRBAC_{1,\,U}$ representation, the complexity is:

$$n.UR\ .$$

The following theorem establishes formally the complexities of the alternative representations using $GTRBAC_0$ model.

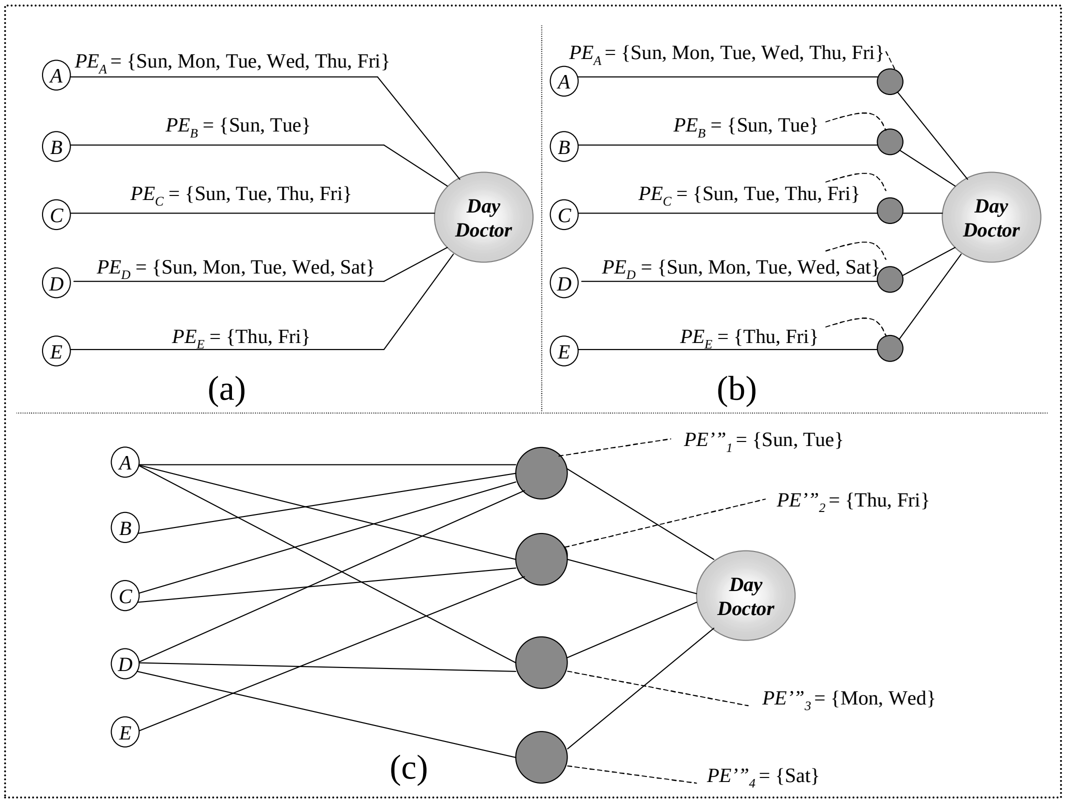


Figure 10. Access requirements of `Example 4.2` using (a) $GTRBAC_{1,\,U}$ representation (b) $GTRBAC_0^{\,1}$ representation and (c) $GTRBAC_0^{\,2}$ representation

**Theorem 4.4 (Complexity expressions for $GTRBAC_0^{\,1}$ and $GTRBAC_0^{\,2}$ representations)**: *Let n be the number of users assigned to a role r, and let PE = {$PE_1$, $PE_2$ …, $PE_n$} be the set of the periodic expressions in the user-role assignment constraints corresponding to n users assigned to r, i.e., there is a ($PE_i$,* `assign` *r* `to` *$u_i$) for each i = 1 to n; Then, the general complexity expressions for the alternative representations $GTRBAC_0^{\,1}$ and $GTRBAC_0^{\,2}$ are as follows*:

1. $GTRBAC_0^{\,1}$ *representation*: *n.DUR + n. R + n. roles + H*,

2. $GTRBAC_0^2$ *representation*: $p_n.DUR + s_n.\ R + s_n.\ roles + H$;

*where* $p_n = |MS_{PE1}(MDS_{PE})| + |MS_{PE2}(MDS_{PE})| + \ldots. + |MS_{PEn}(MDS_{PE})|$, *and* $s_n = |MDS_{PE}|$.

Based on this, we get the following complexities for each representation of `example 4.3`, which is shown in Figure 10.

$GTRBAC_{1,\,U}$ representation: $5.UR$.

$GTRBAC_0^1$ representation: $5.DUR + 5\ R + 5\ roles + H$

$GTRBAC_0^2$ representation: $10.DUR + 4.R + 4.roles + H$
(using algorithm `Transform2`)

We see that, for the above example, the $GTRBAC_{1,\,U}$ representation is the best in terms of complexity – it has the least number of roles, no hierarchy overhead and no default assignments; furthermore, it is simple and intuitive to use and hence very convenient. The main difference between $GTRBAC_0^1$ and $GTRBAC_0^2$ representations is that the latter always produces roles that are temporally disjoint. $GTRBAC_0^1$ representation associates one role for each user for whom there is a temporal assignment constraint. However, $GTRBAC_{1,\,U}$ representations may not be the best for all cases as we show below.

It can be seen that the complexities of $GTRBAC_{1,\,U}$ representations and $GTRBAC_0^1$ representations each remain the same for a given *n*, irrespective of how periodic expressions are pair-wise related. The complexity of $GTRBAC_0^2$ representations, for a given *n*, depends on *MS* and *MDS* of *PE*. The following corollary states the effect of MS and MDS on the complexity of the $GTRBAC_0^2$ representations.

**Corollary 4.4.1(Complexity cases for $GTRBAC_0^2$ representations)**: *Let n be the number of users assigned to a role r, and let PE =* $\{PE_1, PE_2 \ldots, PE_n\}$ *be the set of the periodic expressions in the user-role assignment constraints corresponding to n users, i.e., there is a* ($PE_i$, `assign` *r* `to` $u_i$) *for each i* = 1 *to n; Then:*

1. *if* $PE_i \neq PE_j$, *for all i, j pairs such that* $1 \leq i, j \leq n$ (*i.e., they are pair-wise disjoint*), *then the following holds true*:

   $$complexity\ of\ GTRBAC_0^2 = complexity\ of\ GTRBAC_0^1$$

   *In other words, the complexity* of $GTRBAC_0^2 = n.\ DUR + n.R + n.roles + H$

2. *if* $PE_i = PE_j$, for all *i, j pairs such that* $1 \leq i, j \leq n$ (i.e., *they are pair-wise equivalent*), *then the following holds true*:

$$\textit{the complexity of } GTRBAC_0^2 = n.\ DUR + 1.R + 1.roles + H.$$

3. *the worst case for* $GTRBAC_0^2$ *is:* $n2^n.DUR + 2^n.\ R + 2^n.\ roles + H.$

The first part of the corollary shows that when all the periodic expressions associated with the *user-role* assignments are disjoint, the $GTRBAC_0^2$ representation is the same as the $GTRBAC_0^1$ representation. When $PE_i = PE_j$, for all $i$, $j$ = 1 to $n$ and $n$ is large, $GTRBAC_0^2$ is substantially better than $GTRBAC_{1,\,U}$ representation**,** based on the fact that *temporal constraints* incur more processing cost than *default* assignments. The hierarchy overhead introduced by an extra role can be expected to be negligible to membership check associated with *default* assignment for large $n$. Furthermore, the new role created can be combined with the original role, if that does not introduce extra complicacies, and thus removing the *hierarchy overhead*.

However, the worst case for $GTRBAC_0^2$ representation, as indicated by third part of corollary 4.3 is $O(2^n)$ in the number of new roles created as well as temporal constraints on roles, and $O(n2^n)$ in the number of default assignments.

Based on above observation, we can summarize the following design guidelines.

1. The $GTRBAC_{1,\,U}$ representation is preferable to $GTRBAC_0^1$ representations as its complexity in terms of the number of roles and/or the number of temporal constraints is always better.

2. The $GTRBAC_{1,\,U}$ and $GTRBAC_0^1$ representations may result in using temporal constraints that can be avoided because of some common periodic expressions. For example, there may be a large number of doctors who need to use the role DayDoctor role at *daytime*, making *daytime* a common period for many users. Using the $GTRBAC_0^1$ representations in such cases also results in the same temporal periodicity constraints on different roles, as algorithm `Transform2` does not attempt to reduce constraints based on common periodicity expressions. The $GTRBAC_0^2$ is a good solution in al such cases where some user-role assignments have common periodic expressions. If all the periodic expressions are equivalent then it produces a single role and all users are assigned to that role, as indicated by the results in second part of corollary 4.3. Theorem 4.4 and corollary 4.3 show that $GTRBAC_0^2$ is advantageous when the *MS* set of each periodic expression is very small (the smallest case is when it has one member, as in the 2nd part of corollary, i.e., when all the periodic expressions

are equivalent). Furthermore, we want a small *MCS* set, as it determines the number of new roles created.

Similarly, if all the periodic expressions are pair-wise disjoint, then $GTRBAC_0^2$ representation becomes equivalent to the $GTRBAC_0^1$ representations as shown by the first part of corollary 4.3.

3. The $GTRBAC_{1,\ U}$ representation is very flexible with respect to access specification since it supports temporal constraints on *user-role* assignments, in addition to the constraints on role enabling. For example, we can have the following constraints:

   ([Mon, Wed, Fri], `assign` John `to` DayDoctor)
   ([Tue, Thurs], `assign` John `to` NightDocotor).
   ([10am, 3pm], `assign` Greg `to` DayDoctor).

   By using the above constraints, we can keep the roles that have static temporal enabling times fixed in the system and express individual user requirements using periodicity constraints. Here, DayDoctor and NightDoctor roles are more or less fixed in the system and, as illustrated, users are assigned to it as required.

4. Note that if there are *per-user-role* activation constraints, using the $GTRBAC_0^2$ representations may not be advantageous. For example, in the example above (Figure 10(c)), each user is assigned to multiple new roles. In such a case, if there had been a *per-user-role* constraint for each user, we would have needed to take extra steps during its transformed representation. Here, we note that algorithm `TransformMDS` creates an *activation* hierarchy of type $A_r$ between the new roles and the original role. So if we leave the *per-user-role* untouched, i.e., in the transformed representation, the *per-user-role* is still specified in terms of the original role, then the new representation is still valid, as the users assigned to the new role will have to explicitly activate the new role. However, it is neither intuitive nor convenient to keep track, as the users are only implicitly assigned to the original role. Therefore, in presence of *per-user-role* activation constraints $GTRBAC_0^1$ *and* $GTRBAC_{1,\ U}$ provide more intuitive and convenient representations than $GTRBAC_0^2$.

5. Unlike periodicity constraints, duration constraints are somewhat inflexible in terms of being replaced (for example, replacing *user-role* assignment by role enabling). As duration constraints have non-deterministic start times, such constraints depend on some other events. Such dependencies often have some application semantics and even though it may be

possible to replace a duration constraint on *user-role* assignment, as in the case of periodicity constraints, care must be taken to ensure that the dependency semantics is not hindered. We illustrate this with an example:

`Example 4.3`: Consider Manager and Employee roles in an office and assume that the Employee role is enabled on weekdays from 9am to 5pm, whereas the Manager role is enabled everyday. At other times, the Employee role is enabled only if Mr. Smith, the manager who is also the owner, has activated his Manager role. This can be expressed using the following trigger:

`activate` Manager `for` Smith $\rightarrow$ `enable` Employee ($t_1$)

Suppose Smith wants to allow John, an employee in his office, to work on Saturday and Sunday when he is also working, for at most 4 hours, then he can do that by adding the following constraints:

([Sat], 4 hours, `assign` John `to` Employee) ($c_1$)

`activate` Manager `for` Smith $\rightarrow$ `assign` John `to` Employee ($t_2$)

`de-activate` Manager `for` Smith $\rightarrow$ `disable` Employee ($t_3$)

When Smith activates the Manager role on Saturday, it enables Employee using trigger $t_1$ and assigns John to the Employee role using trigger $t_2$. Because of the constraint $c_1$ active at the time, the assignment gets restricted to 4 hours during which John can work.

In this case, if we try to use the duration constraint on Employee role instead, the implicit dependency between the activation of Manager role and allowing John to work is lost.

6. We note that transformation such as in $GTRBAC_0^2$ is not possible for *user-role* assignment with duration constraints. Although there may be common duration values associated with different *user-role* assignments, there is an inherent dependency semantics associated with each duration constraint that relates it to a trigger or a constraint enabling expression.

7. Except for the discussion presented in 4, all apply to *role-permission* assignments too.

Thus, we can see, except for some cases, where $GTRBAC_0^2$ is better in terms of complexity of representations. $GTRBAC_2$ gives the best representational form, both in terms of complexity and convenience.

### 4.2.2 Activation Constraints

In this section, we compare the use of *GTRBAC*$_0$ and *GTRBAC*$_{1,A}$ models to express the same set of activation constraints. For simplicity, we assume that *GTRBAC*$_{1,A}$ has only *total active duration* constraints in addition to constraints in *GTRBAC*$_0$. Same kinds of analysis apply to other activation constraints. In the complexity expressions, we use *PUR* to mean *per-user-role* activation constraint, *PR* to mean *per-role* activation constraints and *H* to mean the overhead associated with hierarchy. In addition, we will not include the original role and any of its associated *per-role* constraints in the complexity expressions. For discussions that follow, we use the following example:

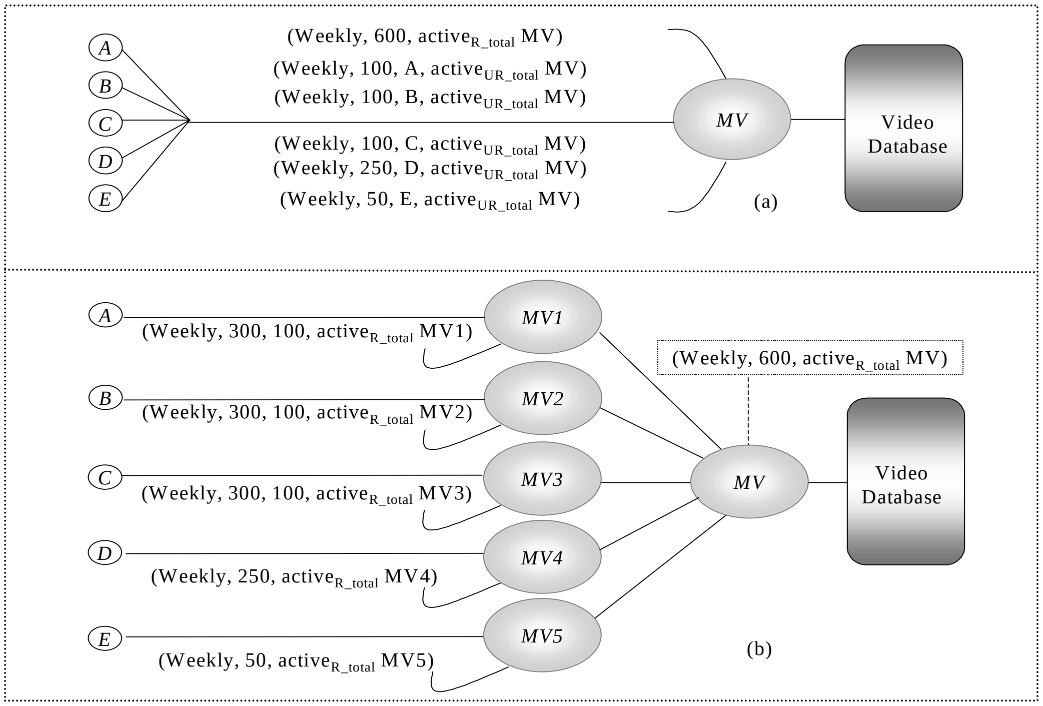
> `Example 4.4`: Let *A*, *B*, *C*, *D* and *E* be the users subscribing 100, 100, 100, 250, 50 hours of active time per week respectively from a Video Library. A straightforward representation of these constraints using *GTRBAC*$_{1,A}$ model is shown in Figure 11(a) (we will refer to this as *GTRBAC*$_{1,A}^s$ representation). To represent these constraints using *GTRBAC*$_0$, we can use the part of algorithm `Transform2` that removes *per-user-role* activation constraints (or we can simply assume that there are no temporal assignment constraints and run the `Transform2` on this configuration). Such a representation, later referred to as *GTRBAC*$_0^s$ representation, is shown in Figure 11(b).

From the example, it is clear that the straightforward representation of a set of *n per-user-role* constraints for *n* users assigned to a role (a *per-role* constraint on the role may or may not be present), using the two models incur the following costs:

*GTRBAC*$_{1,A}^s$ representation: $n\,.PRU$ (*i*)

*GTRBAC*$_0^s$ representation: $n\,.PR + n.role + H$ (*ii*)
(*using algorithm* `Transform2`)

Note that, we did not include the original role and any *per-role* constraints on it, as they will always remain the same. We can see that between the two cases illustrated above, the *GTRBAC*$_{1,A}^s$ model gives better representation in terms of the reduced number of roles. The total number of activation constraints is the same in both. However, we want to know if these give the best representations. We observe that in Figure 11(a), the users *A*, *B* and *C* have same *per-user-role* access requirements and hence can possibly be expressed as one *per-role* constraint. Similarly, we see that in Figure 11(b), *MV1*, *MV2* and *MV3* have the same *per-role* constraint values, which can possibly be combined. The following theorem formally shows that such reduction in complexity can be achieved, when there are duration constraint values that are common.



A
B
C
D
E
(Weekly, 600, active$_{R\_total}$ MV)
(Weekly, 100, A, active$_{UR\_total}$ MV)
(Weekly, 100, B, active$_{UR\_total}$ MV)
(Weekly, 100, C, active$_{UR\_total}$ MV)
(Weekly, 250, D, active$_{UR\_total}$ MV)
(Weekly, 50, E, active$_{UR\_total}$ MV)
MV
Video Database
(a)
A
B
C
D
E
(Weekly, 300, 100, active$_{R\_total}$ MV1)
(Weekly, 300, 100, active$_{R\_total}$ MV2)
(Weekly, 300, 100, active$_{R\_total}$ MV3)
(Weekly, 250, active$_{R\_total}$ MV4)
(Weekly, 50, active$_{R\_total}$ MV5)
MV1
MV2
MV3
MV4
MV5
(Weekly, 600, active$_{R\_total}$ MV)
MV
Video Database
(b)

Figure 11. Access requirements of `Example 4.4` using (a) $GTRBAC_{1,A}{}^{s}$ representation (b) $GTRBAC_{0}{}^{s}$ representation by running algorithm `Transform2` on a $GTRBAC_{1,A}{}^{s}$ configuration

**Theorem 4.5 (Complexity expression for *GTRBAC*$_0$ and *GTRBAC*$_1$ representations)**: *Let n be the number of users assigned to role r, D = {$d_1$, $d_2$,, … $d_n$ | $d_i$ is the total active duration that the i*$^{th}$ *user is allowed over role r}, $D_m$ = {$d'_1$, $d'_2$,, … $d'_m$} $\subseteq$ D be the set of distinct elements of D, and $C_m(d)$ be the number of times d occurs in D; Then the complexities of the following two representations are as follows:*

1. *GTRBAC*$_{1,A}$ *representation*: $(n_x - n_y) .PUR + n_y\, PR + c.( b.n_y + 1)$ *roles* $+ c. H$
2. *GTRBAC*$_0$ *representation*: $n_x .PR + n_x .roles + H$

*where,*

- $n_x = |D_m|$ *and* $n_y = |D'|$ *, such that*
    - o $D' \subseteq D_m$ *,and*
    - o *if* $d \in D'$ *then* $C_m(d) > 1$
- $b = 1$ *if* $(n > n_x)$; $b = 0$ *otherwise,*
- $c = 1$ *if* $(n > n_x > 0)$; $c = 0$ *otherwise.*

The complexities of the previously mentioned representations of the constraints as shown in (*i*) and (*ii*) can be easily derived by forcing each element in $D$ to be considered as unique. In that case,

$$n_x = |D_m| = n,\ n_y = 0,\ b = 0\ \textit{ and } c = 0$$

and hence the complexities are as follows.

$GTRBAC_{1,\mathrm{A}}^{s}$ representation:

$$= (n_x - n_y)\,.PUR + n_y\,PR + c.(\,b.n_y + 1)\ \ roles + c.\,H = n.PUR \qquad (\textit{same as } (i))$$

$GTRBAC_0^{s}$ representation:

$$= n_x\,.PR + n_x\,.roles + H\ \ =\ n\,.PR + n\,.roles + H \qquad (\textit{same as } (ii))$$

Thus, for `Example 4.4`, we have the following complexities, as given by Theorem 4.5 (the constraints are as shown in Figure 11):

| | |
|---|---|
| $GTRBAC_{1,\mathrm{A}}^{s}$ representation: | $5\ PUR$ |
| $GTRBAC_0^{s}$ representation: | $5.PR + 5.roles + H$ |

Here, we see that, in $GTRBAC_0^{s}$ representation, there are 5 temporal constraints for the 5 new roles and one for the old role. In $GTRBAC_{1,\mathrm{A}}^{s}$ representation, there is just one role with on per-role constraint (original role and hence not included) but there are five *per-user-role* and one *per-role* constraints.

Figure 12 illustrates the general constraint design that combines common total active duration constraints as is used in Theorem 4.5. Here, we get $n_x$=3 as $D_m$={50, 100, 250}, $n_y$=1 as $D'$={100}, $b$ = 1 and $c$ = 1. Therefore, the complexities are:

$GTRBAC_0$ representation:

$$= n_x\,.PR + n_x\,.roles + H\ = 3.PR + 3.roles + H$$

$GTRBAC_{1,\mathrm{A}}$ representation:

$$= (n_x - n_y)\,.PUR + n_y\,PR + c.(\,b.n_y + 1)\ \ roles + c.\,H$$

$= 2.PUR + 1.PR + 2.role + H$

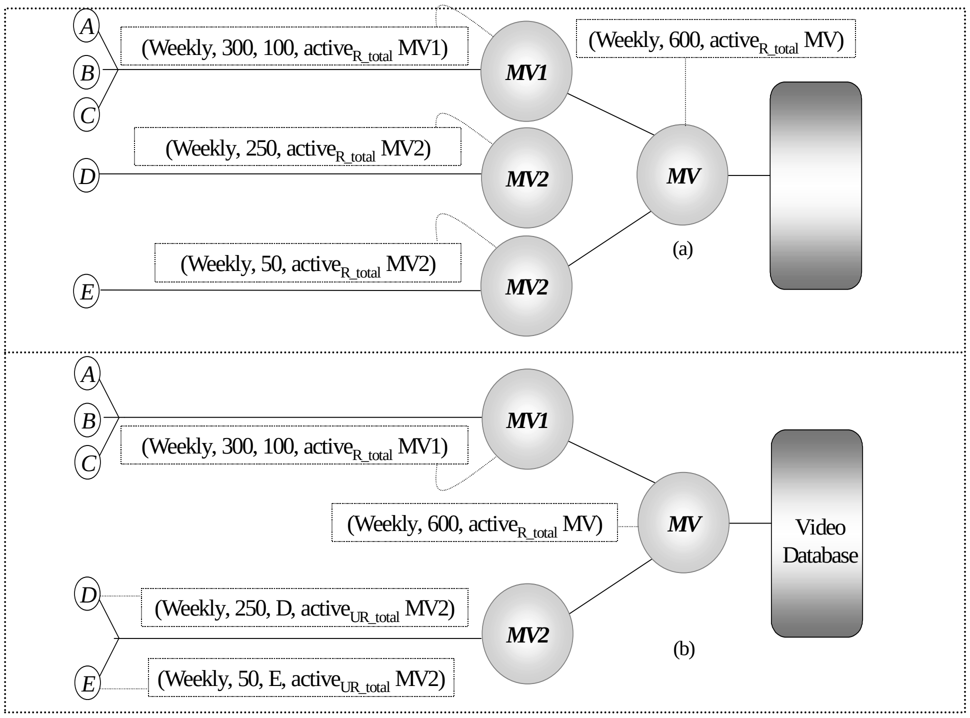

A
B
C
(Weekly, 300, 100, active$_{R\_total}$ MV1)
MV1
(Weekly, 600, active$_{R\_total}$ MV)
(Weekly, 250, active$_{R\_total}$ MV2)
D
MV2
MV
(Weekly, 50, active$_{R\_total}$ MV2)
E
MV2
(a)
A
B
C
MV1
(Weekly, 300, 100, active$_{R\_total}$ MV1)
(Weekly, 600, active$_{R\_total}$ MV)
MV
Video Database
D
(Weekly, 250, D, active$_{UR\_total}$ MV2)
MV2
E
(Weekly, 50, E, active$_{UR\_total}$ MV2)
(b)

Figure 12. Constraints of `Example 4.4` (a) using *GTRBAC*$_0$ representation (b) using *GTRBAC*$_{1,A}$ representation

We can summarize the following guidelines based on the above observation.

1. If there are many users having a common active duration requirement, then using a role and a constraint that specify both the *total* and *default* duration constraint minimizes both the number of roles and the number of temporal constraints, as shown by Theorem 4.5

2. If the expected requirements for active durations for individual users vary substantially from user to user, *GTRBAC*$_{1,A}$ representation is preferable.

3. If flexibility is needed, using *per-user-role* constraints (and hence *GTRBAC*$_{1,A}$ representation) is better. For example, if the users *A, B, C, D* and *E* request different active duration every week, then the use of *per user-role* constraints is more appropriate.

4. In some cases, a hybrid approach utilizing constraints on both *per role* and *per user-role* will give a more efficient representation, as shown by Figure 12(b). This is the *GTRBAC*$_{1,A}$ representation as per Theorem 4.5

Thus, we see that *GTRBAC*$_{1,A}$ representation has distinct advantages over the *GTRBAC*$_0$ representation.

## 5 Related Work

The TRBAC model proposed by Bertino *et. al*. [5], is the first known model that addresses temporal constraints for a an RBAC model. It, however, provides constraints only on role enabling and triggers, considerably limiting its use in a diverse set of practical requirements. The work presented in [11] is a generalization of the TRBAC model and constitutes a substantial extension to it. However, issues such as whether the exhaustive set of GTRBAC constraints has any practical benefit are not addressed in [11]. The effects of temporal constraints on the inheritance semantics of a role hierarchy have not been dealt upon in [12]. This paper has mainly focused on the issue of expressiveness of the GTRBAC model. Another approach dealing with the time based control of access can be found in [3] by Bertino *et. al.*, that supports temporal authorization and derivation rules. Formalism for periodic time used in this paper as well as in [11] has been borrowed from [14, 5].

Many researchers have addressed the need for supporting constraints in an RBAC model. The attention has been particularly in supporting SOD constraints [1, 13, 17, 18, 20]. SOD constraints are mainly aimed at reducing the risk of a fraud by not allowing any individual to have sufficient rights to perpetrate such frauds. Ferraiolo *et. al.* [6] propose an RBAC model that supports the cardinality constraints. In [1], Ahn *et. al.* propose *RCL2000* – a role based constraint specification language. However, they do not address time based access restrictions. Bertino *et. al.* have proposed a logic based constraint specification language that can be used to specify constraint on roles and users and their assignments to workflow tasks [4]. However, it also doe not include temporal constraints in their specification models.

## 6 Conclusions

In this paper, we have addressed the issue of expressiveness of the GTRBAC model. As our major contribution, we showed through exhaustive analysis of minimality of the GTRBAC model that a comprehensive set of GTRBAC constraints can provide distinct advantages over minimal GTRBAC model in terms of user convenience and the complexity of constraint representation. This is practically a significant result as it shows that although the GTRBAC model is not minimal, its constraints set provides constraint designers with flexibility and intuitive choices over various constraint expressions as well as a much better and less complex representations in certain cases. Based on these results, we outlined some design guidelines that can assist constraint designers in choosing more convenient and less complex constraint expressions.

We plan to extend the present work in various directions. The first direction is an extensive investigation on significant design issues when arbitrary mixed hierarchies are present. We note that we presented our theoretical analysis by assuming the use of a monotype *A*-hierarchy. As we have mentioned, *A*-type hierarchy is theoretically the best representative among all hierarchies that are *AC*-equivalent to it, which excludes the cases where *I*-hierarchy precedes an *A*-hierarchy within the same hierarchical chain. We also plan to develop an SQL-like language for specifying temporal properties for roles and the various types of constraints and inheritance relations.

We furthermore plan to develop a prototype of such language by extending the implementation of TRBAC to support all the constraint types of GTRBAC. Finally, we plan to develop a tool that, based on the results presented in this paper, supports the security policy designer in establishing the proper role configuration for a given set of policy requirements.

## References

[1] G.J. Ahn and R. Sandhu The RSL99 Language for Role-Based Separation of Duty Constraints. In *Proc. of the Fourth ACM Workshop on Role-Based Access Control*, Fairfax (VA), 1999.

[2] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, and D.R. Kuhn. Role Based Access Control for the World Wide Web. In *20th National Information System Security Conference*, NIST/NSA, 1997.

[3] E. Bertino, C. Bettini, E. Ferrari, P. Samarati. An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning. *ACM Transactions on Database Systems*, 23(3):231-285, September 1998.

[4] E. Bertino, E. Ferrari, and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM Transactions on Information and System Security*, 2(1):65-104, 1999.

[5] E. Bertino, P. A. Bonatti, E. Ferrari. TRBAC: A Temporal Role-based Access Control Model. *ACM Transactions on Information and System Security*, 4(4), 2001 (in print).

[6] D. F. Ferraiolo, D. M. Gilbert, and N Lynch. An examination of Federal and commercial access control policy needs. In Proceedings of NISTNCSC National Computer Security Conference, pages 107--116, Baltimore, MD, September 20-23 1993.

[7] L. Giuri. A new model for role-based access control. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 249-255, New Orleans, LA, December 11-15 1995.

[8] L. Giuri. Role-based access control: A natural approach. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[9] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford. Security models for web-based applications. *Communications of the ACM*, 44, 2 (Feb. 2001), pages 38-72.

[10] J. B. D. Joshi, A. Ghafoor, W. Aref, E. H. Spafford. Digital Government Security Infrastructure Design Challenges. *IEEE Computer*, Vol. 34, No. 2, February 2001, pages 66-72.

[11] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor. Generalized Temporal Role Based Access Control Model (GTRBAC) (Part I)– Specification and Modeling. CERIAS TR 2001-47, Purdue University, USA, 2001 (Submitted as Part I to TKDE for review along with this paper).

[12] J. B. D. Joshi, E. Bertino, A. Ghafoor. Temporal Hierarchies and Inheritance Semantics for GTRBAC. CERIAS TR 2001-52, Purdue University, USA, 2001.

[13] S. Kandala and R. Sandhu. Extending the BFA Workflow Authorization Model to Express Weighted Voting. In *Research Advances in Database and Information Systems Security*, pages 145-159, Kluwer Academic Publishers, 1999

[14] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proc.First International Conference on Information and Knowledge Management*, 1992.

[15] M. Nyanchama and S. Osborn. The Role Graph Model and Conflict of Interest. *ACM Transactions on Information and System Security*, 2(1):3-33, 1999.

[16] J. S. Park, R. Sandhu, G. J. Ahn. Role-based access control on the web. *ACM Transactions on Information and System Security* (TISSEC) Volume 4, Issue 1 (February 2001) Pages: 37 – 71.

[17] R. Sandhu. Separation of Duties in Computerized Information Systems. In *Database Security IV: Status and Prospects*, pages 179-189. North Holland, 1991.

[18] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman. Role-Based Access Control Models", *IEEE Computer* 29(2): 38-47, IEEE Press, 1996.

[19] R. Sandhu. Role Activation Hierarchies. *Proceedings of 2rd ACM Workshop on Role-based Access Control*, Fairfax, Virginea, October 22-23, 1998

[20] R. Simon and M.E. Zurko. Separation of Duty in Role-based Environments. In *Proc. 10th IEEE Computer Security Foundations Workshop*, June 1997.

# Appendix

**Proof Lemma 4.1.1**: (**Correctness of** `Transform1`)

Let us consider an arbitrary user $u$ such that $(u \overset{C_{in}}{\underset{t}{\Rightarrow}} p)$. We need to show that $(u \overset{C_{out}}{\underset{t}{\Rightarrow}} p)$. Since $(u \overset{C_{in}}{\underset{t}{\Rightarrow}} p)$, the following must be true at time $t$ for $C_{\text{in}}$:

(1) there is a constraint $\{X, pr$:`assign/deassign` $p$ `to` $r\} \in C_{\text{in}}$ because of which $p$ is assigned to role $r$,

(2) role $r$ is enabled,

(3) user $u$ is assigned to role $r$,

(4) there is no activation constraint that prevents the user from activating the role.

We note that algorithm `Transform1` only replaces the constraints of types $\{X, pr$:`assign/deassign` $p$ `to` $r\}$ to produce $C_{\text{out}}$ and temporal constraints on original roles are not changed. Hence, (2), (3) and (4) are still valid in $C_{\text{out}}$ at time $t$. The FOR loop in line 2 repeats for every constraint of type $\{X, pr$:`assign/deassign` $p$ `to` $r\}$. Each constraint $\{X, pr$:`assign/deassign` $p$ `to` $r\}$ is replaced by temporal constraint on role enabling/disabling in line 4. Thus a constraint of type $\{X, pr$:`assign/deassign` $p$ `to` $r\}$ is not in $C_{\text{out}}$. We need to show that $C_{\text{in}} \approx C_{\text{out}}$ for the following two cases:

**Case 1**: Let $X = (I, P)$, i.e. $\{X, pr$:`assign/deassign` $p$ `to` $r\}$ *in* (1) *is a periodicity constraint*: We note that following replacements take place in $T'$ (initially $T' = T$) according to lines 4, 5, and 7:

(i) the replacement of all temporal role-permission assignment expressions by

*a.* temporal constraint on the corresponding new role in line 4, and

*b.* default assignments, as is shown added in line 5.

(ii) the replacement of all occurrences of temporal role-permission assignment expressions and role-permission assignment status expressions in triggers by constraint and status expressions on the new roles as shown in line 7-12..

Because of (i) and (ii), for all triggers or constraint enabling events that cause "`assign/deassign` $p$ `to` $r$" event in $C_{\text{in}}$, the algorithm produces triggers and constraints that cause "`enable/disable` $r_i$" event in $C_{\text{out}}$ and vice versa. Hence, if because of $\{(I, P),$

*pr*:`assign/deassign` *p* `to` *r*} in $C_{\text{in}}$, permission *p* is assigned to *r* at time *t*, then because of {(*I*, *P*), *pr*:`enable/disable` $r_i$} and default assignment {*pr*:`assign/deassign` *p* `to` $r_i$} in $C_{\text{out}}$, *p* is assigned to $r_i$ at time *t* and vice versa. Hence, a user *u* who is assigned to role *r* that is enabled at time *t* and as (2), (3) and (4) remain same at *t*, can inherit *p* through $r_i$, using *restricted inheritance* $I_r$ in $C_{\text{out}}$. This inheritance allows *u* to acquire exactly those permissions that he can acquire in $C_{\text{in}}$. Hence, $(u \overset{C_{out}}{\underset{t}{\Rightarrow}} p)$ and therefore $C_{\text{in}} \approx C_{\text{out}}$.

**Case 2**: ( $X$ = ([(*I*, *P*)| *D*], $D_x$), i.e. *c* = {*X*, *pr*:`assign/deassign` *p* `to` *r*} *in* (1) *is a duration constraint*): The transformation indicated by line 4 also replaces all duration constraints of this form by the same duration constraint on the new role's enabling/disabling times. Thus enabling/disabling of the assignment constraint in $C_{\text{in}}$ done by any "`enable/disable` *c*" expression (independent constraint enabling expressions or in triggers) now enables the duration constraint on the new role and vice versa. Thus, since (2), (3) and (4) still remain valid, user *u*, who is assigned to role *r*, which is enabled at time *t* can inherit *p* through $r_i$, using *restricted inheritance* $I_r$ in $C_{\text{out}}$. Thus, $(u \overset{C_{out}}{\underset{t}{\Rightarrow}} p)$. Hence, $C_{\text{in}} \approx C_{\text{out}}$

**Proof of Lemma 4.1.2** (**Correctness of** `Transform2`):

We prove this by considering following cases:

**Case 1**: *There are no per-user-role activation constraints in T*: In this case lines 16-37 do not apply. We also note that except for the hierarchy relations added to *RH*' with respect to the new roles, everything else is same as that of algorithm `transform1` if the assignment of permissions is replaced by assignment of users. So, by arguments similar to one used to prove lemma 4.1.1, we can show that the transformation of temporal constraints on user-assignments done by `transform2` produces an *a-equivalent* configuration.

**Case 2**: *There are no temporal constraints on user assignments*: In this case, only lines 16-37 apply. Since *S* is empty, new roles will be created for all *per-user-role* activation in line 30. Each set of *per-user-role* constraint associated with user role pair (*u, r*) is replaced by a new role and a correspond set of *per-role* constraint on it so that all activation constraints associated with a user-role pair applies to the corresponding new role. Since each new role is assigned to only one user, in $C_{\text{out}}$, the *per-role* constraint on it has the same effect as the *per-user-role* with the matching constraint value (total active duration, cardinality etc.). Since an old role retains the *per-role* activation constraint in $C_{\text{out}}$, any effect it has on users assigned to it will be the same as the effect it has when a user attempts to activate it through new roles that inherit form it. Thus, as $(u \overset{C_{in}}{\underset{t}{\Rightarrow}} p)$, it follows that $(u \overset{C_{out}}{\underset{t}{\Rightarrow}} p)$.

**Case 3**: *Both temporal user assignments and per-user-role activation constraints are present*: This case is similar to case 1, in that a new role is created for each user assignment. In addition, all *per-user-role* activation constraints are transformed into *per-role* constraint for the new role created, as indicated by line 2 and 30 (use of $getSu_i$ allows creation of one new role for a $(u, r)$ pair). As the new roles still have only one user assigned to it, the *per-role* constraints applied to them has the same effect as the original *per-user-role* constraints. Hence, as $(u \overset{C_{in}}{\underset{t}{\Rightarrow}} p)$, it follows that $(u \overset{C_{out}}{\underset{t}{\Rightarrow}} p)$

**Case 4**: *There are no user-role assignment and no per-user-role activation constraints*: In this case the algorithm simply returns $C_{\text{in}}$ as $C_{\text{out}}$ as both the FOR loops at lines 2 and 22 are not entered.

Hence, it follows that for a given input $C_{\text{in}}$, if $C_{\text{out}}$ is the output produced by algorithm `Transform2`, then $C_{\text{out}}$ contains no temporal user assignments and *per-user-role* activation constraints, and $C_{\text{in}} \approx C_{\text{out}}$.

**Proof of Theorem 4.1 (Minimality of GTRBAC)**

**<u>Proof for (a) and (b)</u>:** To prove (a) and (b), we do the following:

**Step 1**: Let $C_{12}$= `transform1`($C_1$), i.e., configuration $C_1$ is input to algorithm `transform1` , and $C_{12}$ is the new *a-equivalent* configuration returned by it.

**Step 2**: Let $C_2$= `transform2`($C_{12}$), i.e., configuration $C_{12}$ is input to algorithm `transform2`, and $C_2$ is the new *a-equivalent* configuration returned by it.

Since $C_1 \approx C_{12}$ by lemma 4.1.1, and $C_{12} \approx C_2$ by lemma 4.1.2, it implies that $C_1 \approx C_2$ as per definition 4.1.1. As `transform1` removes all temporal role-permissions assignments, $C_{12}$ does not have any temporal constraints on role-permission assignment. Similarly, since `transform2` removes all temporal user-role assignments and *per-user-role* activation constraints, $C_{12}$ does not have any temporal constraints on role-permission assignment and *per-user-role* activation constraint. Hence, $MCS\ (T_2) \subseteq \{C_d, C_{Rp}, C_{Rd}, C^a{}_r, C_{tr}, C_c\}$.

<u>Proof for (c)</u>: From (b), we have $MCS(T_2) = \{C_d, C_{Rp}, C_{Rd}, C^a{}_r, C_{tr}, C_c\}$. We need to prove that $MCS(T_2)$ is minimal. We show that a constraint type from $MCS(T_2)$ can not be replaced by another constraint type of $MCS(T_2)$ to produce an *a-equivalent* configuration. We show this case-wise.

**Case 1**: (*Periodicity*($C_{Rp}$) *vs. Duration constraints*($C_{Rd}$) *on role*): Periodicity constraint specifies each time instant at which a role is enabled/disabled, whereas, duration constraint does not specify the starting/ending time at which a role is enabled/disabled. Furthermore, an event associated with a duration constraint needs to be triggered or caused by a runtime request. A *periodicity* constraint can be represented by a *duration* constraint if there is a way to enable it (the duration constraint) at a specific time instant that corresponds to the start time of the *periodic* expression. But GTRBAC does not support such specific constraint enabling unless we use a trigger in which clock timer is allowed to trigger an "`enable` *c*" event that enables the duration constraint, which then becomes equivalent to the original periodicity constraint. However, even if we allow that, the duration constraint that is generated to enforce the periodicity constraint will allow any other trigger or run-time event to enable the role, which is not what the periodicity constraint is intended to do.

Similarly, a duration constraint cannot be specified using a periodicity constraint as it does not have deterministic start times.

**Case 2**: (*Duration constraint vs. Trigger*): Assume we have the following set of triggers:

$B \rightarrow \texttt{enable}\ r$ ................................................................................................................(1)

$\texttt{enable}\ r \rightarrow\ \texttt{disable}\ r\ \texttt{after}\ \Delta t.$ ..........................................................................(2)

If (1) triggers the non-blocked event `enable` *r* then (2) will allow role *r* to be enabled for a duration $\Delta t$. In effect, this is similar to the duration constraint ($D = \Delta t$, `enable` *r*). However, if we also have a periodicity constraint (*I*, *P*, `enable` *r*) in *T* of $C_{\text{in}}$, then whenever, for an instant $t \in Sol(I, P)$, the non-blocked event `enable` *r* is caused, trigger (2) will enable the duration constraint. This is semantically different from a duration constraint ($D = \Delta t$, `enable` *r*), in which only a trigger or a run-time event can cause the duration restriction for event "`enable` *r*" as specified by ($D$ = $\Delta t$, `enable` $r$). Thus, representing the semantically same duration constraint by triggers is not possible in the GTRBAC framework.

**Case 3**: (*Activation vs. Non-activation constraint*): Replacement of one by the other is not possible because they refer to the different states of a role. In addition, for an enabling/disabling of a role, no user needs to be assigned to the role. An activation constraint needs to be enforced only when a user is actually using the associated role.

Hence, $MCS$ ($T_2$) $\subseteq\{C_d, C_{Rp}, C_{Rd}, C^a{}_r, C_{tr}, C_c\}$ is minimal.

**Proof of Lemma 4.2.1 (MDS of two periodic expressions)**

a. Here we have $PE_i \subseteq PE_j$. Hence for all $t \in Sol(PE_i)$, it is also true that $t \in Sol(PE_j)$. But since $PE_i \neq PE_j$ (non-equivalent), there exists some $t \in Sol(PE_j)$ such that $t \notin Sol(PE_i)$. Therefore, there are two groups of time instants of which one group belongs to both $PE_i$ and $PE_j$, and the other group belongs to only $PE_j$. This implies that atleast two groups of periodic expressions are needed to represent the time instants of both the periodic expressions. This is because if there is a single group for both $PE_i$ and $PE_j$, then we need $PE_x = PE_i \cup PE_j$ in order to satisfy the first condition of a MDS. But then, if we consider $t_1$ and $t_2$ such that $t_{1,} t_2 \in Sol(PE_i)$, $t_1 \in Sol(PE_j)$ and $t_2 \notin Sol(PE_j)$, then the second condition required for a MDS is not satisfied.

As the first group contains time instants that belong to both $PE_i$ and $PE_j$, we can write the first expression to denote this group as $PE_x = PE_i \cap PE_j$, but $PE_i = PE_i \cap PE_j$ , hence, $PE_x = PE_i$ as all time instants that are in $Sol(PE_i)$ are also in $Sol(PE_j)$. The second group of time instants belong to only $PE_j$ , hence we can denote the second group as $PE_y = PE_j - PE_i = PE_j - PE_x$. We can see that $PE_y$ do not contain time instants in $PE_x$, hence, $PE_x$ and $PE_y$ are disjoint.

From the construction of $PE_x$ and $PE_y$, we can see that $PE_x \cup PE_y = PE_i \cup PE_j$ , which is the first condition for a MDS (definition 4.2.2 (a)) Furthermore, Since $PE_x = PE_i$, only those time instants in $PE_i$ belongs to $PE_x$; any time instant $t$ not in $PE_i$ also is not in $PE_x$. Similalry, since $PE_x$ is contained in $PE_j$, only a proper subset of time instants in $PE_j$ is in $PE_x$, and no time instants that is not in $PE_j$ is in $PE_x$. Similarly, by construction, only a proper subset of time instants in $PE_j$ is in $PE_y$, and no time instants that is not in $PE_j$ is in $PE_{x.}$. Thus, $PE_x$ and $PE_y$ satisfy the condition (b) of definition 4.2.2 too. Hence $MDS_{PE} = \{PE_x, PE_y\}$.

b. Here, we have $PE_i \otimes PE_j$. Hence, as the definition of $PE_i \otimes PE_j$ implies, there are three groups of time instants. The first group belongs to both $PE_i$ and $PE_j$. The second group belongs only to $PE_i$, whereas the last group belongs only to $PE_j$. As there exist some common time instants in the two periodic expressions, based on the argument presented in (a) above, its MDS must contain more that one periodic expressions.

   Assume that we can create a MDS that contains two disjoint periodic expressions. Since, there is a group of periodic instants that belong to both $PE_i$ and $PE_j$, they must be represented by a single periodic expression otherwise we cannot get a disjoint pair as required for a MDS. So assume that $PE_x = PE_i \cap PE_j$. Now, we have two remaining groups of time instants, one that belongs only to $PE_i$ and the other that belongs only to $PE_j$. If we combine the two groups to get $PE_y$, then $\{PE_x, PE_y\}$ can not be a MDS, because it will not satisfy the second condition (just take time instants $t_{1,}\ t_2$ such that $t_1$ belong to $PE_i$, and $t_2$ belongs to $PE_j$ but not to $PE_i$, then such $t_{1,}\ t_2$ do not satisfy the second condition).

   Thus, the problem is that one group of time instants belongs to only $PE_i$ and the other belongs to only $PE_i$. Now if we construct $PE_y = PE_i - PE_x$ and $PE_z = PE_j - PE_x$ , we get the disjoint set of periodic expressions $\{PE_x, PE_y, PE_z\}$. As in (a), it is easy to see that $\{PE_x, PE_y, PE_z\}$ satisfies the two conditions of a MDS. Hence, it follows that $\{PE_x, PE_y, PE_z\}$ is a MDS of $\{PE_i, PE_j\}$.

**Proof of Lemma 4.2.2 (MDS of n periodic expressions)**

We show this by induction on the number of periodic expressions *n.* Note that ${}^{i}MDS_{PE}$ represents the MDS of the first *i* periodic expressions of *PE*.

*Basis* : $n = 2$: That is, $PE = \{PE_1 , PE_2\}$. Then by lemma 1, we have the following:

- if $PE_1 \subseteq PE_2$ then $MDS_{PE} = \{PE_x, PE_y\}$ and
- if $PE_1 \otimes PE_2$ then $MDS_{PE} = \{PE_x, PE_y, PE_z\}$.

*Hypothesis*: Assume that it is true for *n*-1, i.e. there exists ${}^{n-1}MDS_{PE} = \{PE'_1, PE'_2, \ldots, PE'_{m1}\}$ for $PE = \{PE_1, PE_2, \ldots, PE_{n-1}\}$

We need to show that $MDS_{PE} = \{PE''_1, PE''_2, \ldots, PE''_{m2}\}$ for $PE = \{PE_1, PE_2, \ldots, PE_n\}$.

We start by writing $MDS_{PE}$ ({$PE_1$, $PE_2$, …, $PE_n$}) = $MDS_{PE}$ ({$^{n-1}MDS_{PE}$, $PE_n$}) = $MDS_{PE}$ ({$PE'_1$, $PE'_2$, …, $PE'_{m1}$, $PE_n$}) (This is true because $PE'_1 \cup PE'_2 \cup ... \cup PE'_{m1} = PE_1 \cup PE_2 \cup ... \cup PE_{n-1}$). Now, we look at pair-wise relations between $PE_n$ and $PE'_i$, for $1 \le i \le m1$. First, we note that it is possible that $PE_n$ is equivalent to some $PE'_i$. A simple example is when $PE_n = PE_i \cap PE_k$ and $PE'_i$ represents $PE_i \cap PE_k$ in $^{n-1}MDS_{PE}$. However, as $PE$ is not a disjoint set, $PE_n$ cannot be disjoint from all $PE'_i$ , $1 \le i \le m1$. We look at each of the possible relations that $PE_n$ may have with each $PE_i$s.

**Case 1**: $PE_n = PE'_i$ for some $i$, such that $1 \le i \le m1$ : Then, $MDS_{PE}$ ({$PE_1$, $PE_2$, …, $PE_n$}) = $^{n-1}MDS_{PE}$ and we are done.

**Case 2**: $PE'_i \subseteq PE_n$ for some $i$, such that $1 \le i \le m1$: Then, by lemma 4.2.1(a), $MDS_{\{PE'i,\ PEn\}}$ = {$PE''_{xi}$, $PE''_{yi}$}, where $PE''_{xi} = PE'_i$, and $PE''_{yi} = PE_n - PE''_{xi}$ .

**Case 3**: $PE'_i \otimes PE_n$ for some $i$, such that $1 \le i \le m1$: Then by lemma 4.2.1(b), $MDS_{\{PE'i,\ PEn\}}$ = {$PE''_{xi}$, $PE''_{yi}$, $PE''_{zi}$}, where $PE''_{xi} = PE'_i \cap PE_n$, and $PE''_{yi} = PE_n - PE''_{xi}$ , *and* $PE''_{zi} = PE'_i - PE''_{xi}$ .

We can see that $PE_n$ may be related to each of the $PE'_i$s, $1 \le i \le m1$ by either case 2 or case 3 (As shown above, we need not worry about case 1; the case of $PE_n \subseteq PE'_i$ can be handled easily by reversing the periodic expressions of MDS in case 2).

Now, consider that $PE'_i \subseteq PE_n$ and $PE'_j \subseteq PE_n$ for $i \neq j$. (i.e. case 2 applies to both $i$ and $j$). Thus, we have $MDS_{\{PE'i,\ PEn\}}$ = {$PE''_{xi}$, $PE''_{yi}$} and $MDS_{\{PE'j,\ PEn\}}$ = {$PE''_{xj}$, $PE''_{yj}$}. We see that $PE''_{xi}$ and $PE''_{xj}$ are disjoint as $PE''_{xi} = PE'_i$ and $PE''_{xj} = PE'_j$, and $PE'_i$ and $PE'_j$ belong to $^{n-1}MDS_{PE}$. However, we do not know how $PE''_{yi}$ and $PE''_{yj}$ are related; but we do know that each of them is a proper subset of $PE_n$.

Now consider that $PE'_i \otimes PE_n$ and $PE'_j \otimes PE_n$ for $i \neq j$. (i.e. case 3 applies to both $i$ and $j$). As shown in case 3, we get: $MDS_{\{PE'i,\ PEn\}}$ = {$PE''_{xi}$, $PE''_{yi}$, $PE''_{zi}$}and $MDS_{\{PE'j,\ PEn\}}$ = {$PE''_{xj}$, $PE''_{yj}$, $PE''_{zj}$}. Now we know that, $PE''_{xi}$ is a subset of $PE'_i$ and $PE''_{xj}$ is a subset of $PE'_j$. Hence, $PE''_{xi}$ and $PE''_{xj}$ are disjoint (as $PE'_i$ and $PE'_j$ are disjoint). Similarly, $PE''_{zi}$ is a subset of $PE'_i$ and $PE''_{zj}$ is a subset of $PE'_j$ and hence, $PE''_{zi}$ and $PE''_{zj}$ are disjoint. Again, we are left with

$PE''_{yi}$ and $PE''_{yj}$ but we do not know how they are related. However, again, we know that each of them is a subset of $PE_n$.

And lastly consider that $PE'_i \subseteq PE_n$ and $PE'_j \otimes PE_n$ for $i \neq j$. (i.e. case 2 applies to the first and case 3 applies to the second; we ignore the situation in which case 3 applies to the first and case 3 applies to second, as it is a simple case of exchanging the index values). Thus, we get $MDS_{\{PE'i, PEn\}} = \{PE''_{xi}, PE''_{yi}\}$ and $MDS_{\{PE'j, PEn\}} = \{PE''_{xj}, PE''_{yj}, PE''_{zj}\}$. Similar to the reasons given above, $PE''_{xi}$ and $PE''_{xj}$ , and $PE''_{xi}$ and $PE''_{zj}$ are disjoint. Again, we are left with $PE''_{yi}$ and $PE''_{yj}$, and we do not know how they are related. However, here too, we do know that they are each subset of $PE_n$.

Hence, we have, $\{MDS_{\{PE'1, PEn\}}, MDS_{\{PE'2, PEn\}} \ldots, MDS_{\{PE'm1, PEn\}}\} = \{PE''_1, PE''_2, \ldots, PE''_{m2}, PE''_{y1}, PE''_{y2}, PE''_{ym1}\}$, where $\{PE''_1, PE''_2, \ldots, PE''_{m2}\} =$

$\{PE''_{xi}, PE''_{xj} |$ case 2 applies both to $MDS_{\{PE'i, PEn\}}$ and $MDS_{\{PE'j, PEn\}}$ and $i \neq j\} \cup$
$\{PE''_{xi}, PE''_{xj}, PE''_{zj} |$ case 2 applies to $MDS_{\{PE'i, PEn\}}$, case 3 to $MDS_{\{PE'j, PEn\}}$ and $i \neq j\}$.

This implies that $(PE''_i \lozenge PE''_j)$, for all $i, j$ pairs such that $i \neq j$, $1 \leq i, j \leq m2$. However, we cannot guarantee that $(PE''_i \lozenge PE''_{yj})$ for all $i, j$ pairs such that $i, j$, $1 \leq i \leq m2$ and $1 \leq j \leq m1$. This is because each $PE''_{yj}$ is a proper subset of $PE_n$ and there are some $PE''_i$ such that $(PE''_i \otimes PE_n)$. However, since the construction of each $PE''_i$ involves breaking down time instants contained in $PE_n$, we can construct a periodic expression for the group of time instants in $PE_n$ that were not contained or overlapped with any other $PE'_i$. Hence we have

Now let $PE''_{m2+1} = PE_n - (PE''_1 \cup PE'' \cup \ldots \cup PE''_{m2})$. Then if $PE''_{m2+1}$ is not empty then $MDS_{PE}(\{PE_1, PE_2, \ldots, PE_n\}) = \{PE''_1, PE''_2, \ldots, PE''_{m2}, PE''_{m2+1}\}$ otherwise $MDS_{PE}(\{PE_1, PE_2, \ldots, PE_n\}) = \{PE''_1, PE''_2, \ldots, PE''_{m2}\}$.

We need to show that $MDS_{PE}$ constructed in this way is minimal. Assume that it is not minimal. Then there is at least one periodic expression $PE''_i$, $1 \leq i \leq m2$ such that all time instants in $PE''_i$ are contained in one or more of $PE''_j$ for $i \neq j$ and $1 \leq i, j \leq m2+1$. But it can only be possible if the periodic expressions in ${}^{n-1}MDS_{PE}$ are not disjoint, as the construction above does not introduce such a non-disjoint set. Hence it contradicts with our assumption. Therefore, $MDS_{PE}$ constructed above is the MDS of $PE$.

**Proof of theorem 4.2 (MDS using `computeMDS`)**

(a) We prove this by taking all the possible cases:

**Case 1 -** ***All the n periodic expressions are equivalent***: In this case anyone of the periodic expression can constitute the MDS, as each periodic expression satisfies the conditions of a MDS.

**Case 2 -** ***The n periodic expressions are pair-wise disjoint***: In this case we can simply consider $MDS = PE$ (and thus $MS_{PEj} = \{PE_j\}$). This satisfies the conditions of a MDS.

**Case 3:** ***The set of n periodic expressions are non-equivalent and non-disjoint***: In this case, according to lemma 4.2.2, there exists a MDS.

Therefore, there exists a MDS for an arbitrary set of periodic expressions.

(b) Again, we prove this by taking all the possible cases used above:

**Case 1 -** ***All the n periodic expressions are equivalent***: In this case, for each $n$>2, `computeMDS` recusively computes MDS of smaller size at line 6. When the recursive call reaches $n = 2$, the algorithm calls `pairMDS` to compute the MDS of $\{PE_1, PE_2\}$. As the periodic expressions are equivalent, `pairMDS` returns $\{PE_1\}$ from line 1. This is returned by `computeMDS` in line 4 for $n = 2$. This is also the value of $MDS$ computed by `computeMDS` at line 6 for $n = 3$. So for $n = 3$, `computeMDS` will compute the MDS of $\{PE_1, PE_3\}$ at line 9 by using the algorithm `pairMDS`. But since $PE_1$ and $PE_3$ are equivalent, again $\{PE_1\}$ is returned. And thus, from line 11, $\{PE_1\}$ will be again returned. We can see, for all $n$ >2, the MDS is the same periodic expression that was returned by the invocation of the algorithm for $n = 2$. Hence, the algorithm correctly returns the MDS for a set of periodic expressions that are equivalent.

**Case 2*****: The n periodic expressions are pair-wise disjoint***: Since all are pair-wise disjoint, for each pair, `pairMDS` returns the original pair of periodic expressions. Now if ${}^{n-1}MDS_{PE} = \{PE_1, PE_2,\ldots, PE_{n-1}\}$, then after the FOR loop in line 8, $S$ will be $\{PE_1, PE_2,\ldots, PE_{n-1}\}$ (i.e., $m2 = n$-1 in the algorithm). Hence, $PE''_{n= m2+1} = PE_n$ at line 20. Therefore, $MDS_{PE} = \{PE_1, PE_2,\ldots, PE_n\}$. Hence, it follows that the algorithm correctly returns the MDS for a set of periodic expressions that are equivalent.

**Case 3:** ***The set of n periodic expressions are non-equivalent and non-disjoint***: We can see that lines 5-25 implement the inductive method used to prove lemma 4.2.2. When $n > 2$, *MDS* of lower values are recursively computed. The FOR loop computes the pair-wise *MDS* of the new periodic expression $PE_n$ with each of the periodic expressions $PE'_j$ computed for the earlier value of $n$. Line 11 returns the earlier *MDS* if $PE_n$ is equivalent to any one of $PE'_j$. In lines 14 and 17, those periodic expressions returned by `pairMDS` are collected in $S$, which constitutes time instants that belong to the periodic expressions of ${}^{n-1}MDS_{PE}$, some of which may also belong to $PE_n$ (when $PE_n$ is contained in or overlaps with some $PE'_j$). In line 20, a periodic expression is created for any time instants that do not fall in the periodic expressions of ${}^{n-1}MDS_{PE}$ but only in $PE_n$. The IF-ELSE statement ensures that this periodic expression is not empty. Hence the algorithm correctly computes the *MDS* of *PE*.

**Proof of Corollary 4.2.1 (Bounds for size of MDS)**: We prove this by induction on $n$.

*Basis*: Let $n = 1$, then trivially $1 \le s_1 \le (2^1 - 1)$, as implied by the first IF statement of line 3 of algorithm `computeMDS`.

For $n = 2$, the second IF statement of algorithm `computeMDS` is executed and the returned set is the set returned by algorithm `pairMDS` for $\{PE_1, PE_2\}$. But algorithm `pairMDS` returns a set whose cardinality is 1, 2, or 3. Hence, $1 \le s_2 \le 3 = 2^2 - 1$.

*Hypothesis*: We assume that it is true for $n$-1. That is, $1 \le s_{n-1} = |{}^{n-1}MDS_{PE}| \le 2^{n-1} - 1$. We need to show that $1 \le s_n = |MDS_{PE}| \le 2^n - 1$.

We observe that a pair-wise MDS is computed for each pair $(PE'_j, PE_n)$, $1 \le j \le s_{n-1} = m1$, where $PE'_j \in {}^{n-1}MDS_{PE}$. For each such pair $(PE'_j , PE_n)$, algorithm `pairMDS` returns at most three disjoint periodic expressions $\{PE_x , PE_y , PE_z\}$. In such a case $MS_{PE'j} = \{PE_x , PE_z\}$. Thus, we see that each of the periodic expression of ${}^{n-1}MDS_{PE}$ is split into at the most two disjoint sets. Furthermore, a new set is created for the remaining time instants of $PE_n$. Hence, we get the following expression,

$$s_n \le 2\, s_{n-1} + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$$

Furthermore, when all periodic expressions are equivalent we get $s_n = 1$.Therefore, $1 \le s_n \le (2^n - 1)$.

**Proof of Corollary 4.2.2 (Bounds for size of MS)**

*Basis*: Let $n = 1$. Then, trivially, $p_1 = 1$ and $n \leq p_1 \leq n2^{n-1}$. Let $n = 2$. Then, when the periodic expressions are equivalent; we get $|MDS_{PE}| = 1$ and $|MS_{PEj}| = 1$ for both $j = 1$ and 2, hence, $p_2 = 2$. However, if $MDS_{PE} = \{PE_x, PE_y, PE_y\}$, then $|MDS_{PE}| = 3$ and $|MS_{PE}| = 2$ for both $j = 1$ and 2, and hence $p_2 = 4$. Thus, $2 \leq p_2 \leq 4 = 2.2^{2-1}$.

*Hypothesis*: Assume that for $n$-1, it is true, i.e., $(n-1) \leq p_{n-1} \leq (n-1)2^{n-2}$. We need to show that $n \leq p_n \leq n2^{n-1}$.

*Induction step*: If all $n$ periodic expressions are equivalent, then $|MDS_{PE}| = 1$ and $|MS_{PEj}| = 1$ for each $1 \leq j \leq s_{n-1} = m1$. Thus, $p_n = n$. Since, this creates the minimum number of expressions in $|MDS_{PE}|$, we have $n \leq p_n$.

When we add the $n^{\text{th}}$ periodic expression $PE_n$, each of $PE'_j$ of ${}^{n-1}MDS_{PE}$ is split into two periodic expressions at the most. Thus the maximum increase in $p_n$ occurs when $PE_n$ overlaps with each $PE'_j$ for $1 \leq j \leq s_{n-1}$. Thus, each of the $MS_{PEj}$ will be split into two. Furthermore, $|MS_{PEn}| = s_{n-1} + 1$, as in the worst case, $PE_n$ overlaps with each of the periodic expressions $PE'_j$ of ${}^{n-1}MDS_{PE}$, and there is a periodic expression that represents those instants of $PE_n$ that are not contained in ${}^{n-1}MDS_{PE}$. Hence, we have,

$$\begin{aligned}
p_n &\leq 2p_{n-1} + (s_{n-1} + 1) \\
&= 2(2p_{n-2} + (s_{n-2} + 1)) + (s_{n-1} + 1) = 2^2 p_{n-2} + 2^1(s_{n-2} + 1) + (s_{n-1} + 1) \\
&= \ldots \\
&= 2^{n-1}p_1 + 2^{n-2}(s_1 + 1) + \ldots + 2^1(s_{n-2} + 1) + 2^0(s_{n-1} + 1) \\
&= 2^{n-1} + 2^{n-2}(s_1 + 1) + \ldots + 2^1(s_{n-2} + 1) + 2^0(s_{n-1} + 1) \\
&\leq 2^{n-1} + 2^{n-2}(2^1 - 1 + 1) + \ldots + 2^1(2^{n-2} - 1 + 1) + 2^0(2^{n-1} - 1 + 1) \\
&= 2^{n-1} + (n-1)2^{n-1} \\
&= n2^{n-1}
\end{aligned}$$

Therefore, $n \leq p_n \leq n2^{n-1}$.

**Proof of Theorem 4.3 (Correctness of `TransformMDS`)**:

We have $PE = \{PE_1, PE_2 \ldots, PE_n\}$ and $MDS = \{PE'_1, PE'_2 \ldots, PE'_m\}$. Furthermore, in line 6 all the required $MS_{PEi}$ are computed. Line 9 creates a unique role for each of the expressions $PE'_i$ which is made senior to the original role using $A_n$. Line 10 inside the FOR loop of line 9 ensures that each user $u_k$ which corresponds to $PE_k$ in the *user-role* assignment of $C_{\text{in}}$ is assigned to this new role associated with $PE'_i \in MS_{PEk}$. This ensures that the following hold

For each $t \in PE_i$, we have $t \in MS_{PEi}$ (by **definition 4.2.3** of MS) .......................................... (a)

For each $u_i \in U$, we see that $u_i$ is (*default*) assigned to each new role that corresponds to expressions in $MS_{PEi}$ by lines 11 and **FOR** loops at lines 8 and 10, and ............................ (b)

For each $PE'_i$, ($PE'_i$, `enable` $r_i$ ) is added to $T$'by line 12.................................................... (c)

Thus from (a), it follows that a $u_i$ can activate the original role $r$ through one of the new roles that corresponds to expressions in $MS_{PEi}$ at $t\in MS_{PEi}$. Furthermore, the periodic expressions {$PE_1$, $PE_2$…, $PE_n$} and {$PE'_1$, $PE'_2$…, $PE'_m$} exactly cover the same time instants. The main loop in line 2 ensures that such transformation is done for each $r$. Hence, it follows from (a), (b) and (c) that $C_{\text{in}} \approx C_{\text{out}}$. We note that the repetition of line 13 removes all the user-role assignment. Hence, $C_{\text{out}}$ is free of user-role assignments.

**Proof of Theorem 4.4** (**Generic complexity expressions $GTRBAC_0^1$ and $GTRBAC_0^2$**)

1. $GTRBAC_0^1$ representation is $n.DUR + n.\ R + n.\ roles + H$ :  Here, $GTRBAC_0^1$ refers to the use of algorithm `Transform2`. For each *user-role* assignment, algorithm `Transform2` creates a new role (therefore total of *n roles*), adds a constraint for each new role (total is *n.R*), and a adds a default assignment (hence, total is *n.DUR*). Furthermore, as new roles are made senior to the original role, we introduce hierarchy overhead too. Hence, for $n$ user-role assignments the complexity incurred is: $n.DUR + n.\ R + n.\ roles + H$,.

2. $GTRBAC_0^2$ representation is $p_n.DUR + s_n.\ R + s_n.\ roles + H$: It follows immediately from corollaries 4.2.1 and 4.2.2 and Theorem 4.3.

**Proof of Corollary 4.4.1** (Complexity cases for $GTRBAC_0^2$ representations)

__**Proof of Part 1**__

From Theorem 4.4, the complexities for $GTRBAC_0^1$ and $GTRBAC_0^2$ are ($n.DUR + n.\ R + n.\ roles + H$) and ($p_n.DUR + s_n.\ R + s_n.\ roles + H$) respectively.

When $PE_i \neq PE_j$, *for all i, j pairs such that* $1 \leq i, j \leq n$, we obtain $p_n = n$ *and* $s_n = n$ as per corollaries 4.2.1 and 4.2.2. Furthermore, hierarchy overhead is also incurred. Hence, the complexity for $GTRBAC_0^2$ representation, using `TransformMDS` (Theorem 4.3) and by Theorem 4.4, is ($p_n.DUR + s_n.\ R + s_n.\ roles + H = n.DUR + n.\ R + n.\ roles + H$) which is also the complexity of the $GTRBAC_0^1$ representations.

__**Proof of Part 2**__

When $PE_i = PE_j$, *for all* $i, j$ *pairs such that* $1 \le i, j \le n$, we obtain $p_n = n$ *and* $s_n = 1$ as per corollaries 4.2.1 and 4.2.2. Furthermore, hierarchy overhead is also incurred. Hence, the complexity for $GTRBAC_0^2$ representation is:

$$= p_n.DUR + s_n. R + s_n. roles + H = n.DUR + 1. R + 1. roles + H.$$

**Proof of Part 3**

As shown by corollaries 4.2.1 and 4.2.2, the worst cases for $p_n$ *and* $s_n$ are $n2^n$ and $2^n$ respectively.

Thus using it in the complexity expression given by Theorem 4.4, we get

$$= p_n.DUR + s_n. R + s_n. roles + H = n2^n.DUR + 2^n. R + 2^n. roles + H$$

**Proof of Theorem 4.5** (**Generic complexity expression** $GTRBAC_0$ **and** $GTRBAC_{1,A}$)

**Proof of 1** (**$GTRBAC_{1,A}$ representation) :** We prove this by cases.

**Case 1**: $d_i \ne d_j$, for all $i, j$ pairs such that $1 \le i, j \le n$, i.e., durations are pair-wise disjoint.

Since durations are distinct from each other, we need a *per-user-role* activation constraint for each. Hence we have the term $n.PUR$ to represent $n$ such constraints. Other than that we do not require other constraints as they fully express the required access constraints. However, the original role is still there and if there is a *per-role* constraint on the original role, it will still be there. Thus the complexity of representation without including the original role and *per-role* constraints on it simply is: $n.PUR$..

Now we show that the expression provided by the theorem gives this same expression. Since, the durations are all pair-wise disjoint, we get $D_m = D$ and therefore $n_x = n$, and $n_y = 0$. Similarly, we see that $b = 0$ and $c = 0$. Therefore the complexity is:

$$= (n_x - n_y) .PUR + n_y\, PR + c.( b.n_y + 1)\ \ roles + c.\, H = n_x.PUR = n.PUR .$$

Thus, the expression holds for this case.

**Case 1**: $d_i = d_j$, for all $i, j$ pairs such that $1 \le i, j \le n$, i.e., durations are all equal.

When all the durations are same, then all *per-user-role* constraints can be expressed as a *per-role* constraint on a role such that the *default* value of the *per-role* constraint is that duration. Thus, we create a new role that is senior to the *original* role and specify the new *per-role* constraint. This obviously incurs some hierarchy overhead. The complexity is, thus, $1.\ PR + 1.\ role + H.$

Now, lets look at the constraint expression given by the theorem. Since, all the durations are equal, there is only one distinct duration element. Hence, $n_x = 1$. Similarly, $n_y = 1$, as the same element occurs more than once in $D$. Values for $b = 0$ and $c = 1$. Therefore, we get :

$$= (n_x - n_y)\,.PUR + n_y.\,PR + c.(\,b.n_y + 1)\ \ roles + c.\,H = 0\,.PUR + 1.\,PR + \ 1.\ \ roles + c.\,H$$

$$= 1.\,PR + 1.\,role + 1.\,H$$

Thus, the expression holds for this case also.

**Case 3**: There is at least one $i, j$ pair, $1 \leq i, j \leq n$ such that $d_i = \ d_j$ and there is at least one $i, j$ pair, $1 \leq i, j \leq n$ such that $d_i \neq \ d_j$.

In this case, $D_m \subset D$. Let $D_m = \{\ d'_1, d'_2,, \ldots\ d'_{nx}\}$, i.e., $n_x = |D_m| < n$. We know that $D' = \{\ d'_{\pi\_1}, d'_{\pi\_2},, \ldots\ d'_{\pi\_ny}\} \subseteq D_m$, where $n_y\ = |D'| \geq 0$. Since each of duration $d'_{\pi\_i}$ is common to at least two users, we create a role $r_{\pi\_i}$ corresponding to each $d'_{\pi\_i}$ and, and specify a *per-role* constraint with $d'_{\pi\_i}$ as the default value and $(C_m(d'_{\pi\_i}) \times d'_{\pi\_i})$ as the total *per-role* active duration value, i.e, the new *per-role* constraint is $(C_m(d'_{\pi\_i}) \times d'_{\pi\_i}, [d'_{\pi\_i}],\ \texttt{active}_{\texttt{R\_total}}\ \ r_{\pi\_i})$. Complexity incurred by this is: $n_y.\,PR + n_y\,.\,roles$.......................................................................................................... (a)

Since $n_x = |D_m|$, $(n_x - n_y)$ is the number of durations that occurs only once in $D$ and hence we can create a role $r_{nx}$ and assign all the users associated with these durations to it and specify an associated *per-user-role* activation constraints for each user. This will incur cost as follows:

$(n_x - n_y).\ PUR + 1.\ roles$ ....................................................................................(b)

Furthermore, the roles $r_{\pi_i}$s and $r_{nx}$ need to be made senior to the original role and thus incurring $H$. Hence adding (a) and (b) and $H$, we get the following complexity:

$(n_x - n_y).\ PUR\ + n_y.\ PR + (n_y + 1).\ Roles + H$ ............................................................($i$)

Now we show that this is exactly the complexity given by the theorem. According to the theorem, in this case, $b = 1$, as $n > n_x$ holds true because of the strict subset relation $D_m \subset D$. Similarly, since there is at least one $i, j$ pair, $1 \leq i, j \leq n$ such that $d_i \neq\ d_j$, therefore $(n > n_x > 0)$ holds true; hence $c = 1$. Therefore, the complexity expression, according to the theorem is:

$$(n_x - n_y)\,.PUR + n_y\,PR + c.(\,b.n_y + 1)\ \ roles + c.\,H = (n_x - n_y)\,.PUR + n_y\,PR + (n_y + 1)\ \ roles + H$$

Which is the same as ($i$) Thus, complexity expression holds good for case 3 too. Since the three cases cover all the possible relation among the duration values, it follows that the complexity expression for $GTRBAC_{1,\mathrm{A}}$ representation is true.

**<u>Proof of 2</u> (*$GTRBAC_0$* representation)**

As $n_x$ is the number of distinct durations $D$, we simply create $n_x$ roles and add to each role a *per-role* constraint. Such a constraint will use the duration value in $D$. For all durations which occurs only once in $D$, they are used as *per-role* duration value in the corresponding new *per-role* constraint, i.e, the new constraint is (X, $d$, $\texttt{active}_{R\_total}$, $r$), where $d$ occurs only once in $D$. For all $d$'s that occur more than once in $D$, the new *per-role* constraint is (X, $C_m(d) \times d$ , $d$, $\texttt{active}_{R\_total}$, $r$). The new roles are senior to the original roles, thus incurring $H$. Hence, the complexity becomes: $n_x.PR + n_x.roles + H$.