Formal Foundations for Hybrid Hierarchies in GTRBAC

JAMES B. D. JOSHI
University of Pittsburgh
ELISA BERTINO
Purdue University
ARIF GHAFOOR
Purdue University
and
YUE ZHANG
University of Pittsburgh

A role hierarchy defines permission acquisition and role-activation semantics through role–role relationships. It can be utilized for efficiently and effectively structuring functional roles of an organization having related access-control needs. The focus of this paper is the analysis of hybrid role hierarchies in the context of the *generalized temporal role-based access control* (GTRBAC) model that allows specification of a comprehensive set of temporal constraints on role, user-role, and role-permission assignments. We introduce the notion of *uniquely activable set* (UAS) associated with a role hierarchy that indicates the access capabilities of a user resulting from his membership to a role in the hierarchy. Identifying such a role set is essential, while making an authorization decision about whether or not a user should be allowed to activate a particular combination of roles in a single session. We formally show how UAS can be determined for a hybrid hierarchy. Furthermore, within a hybrid hierarchy, various hierarchical relations may be derived between an arbitrary pair of roles. We present a set of inference rules that can be used to generate all the possible derived relations that can be inferred from a specified set of hierarchical relations and show that it is *sound* and *complete*. We also present an analysis of hierarchy transformations with respect to role addition, deletion, and partitioning, and show how various cases of these transformations allow the original permission acquisition and role-activation semantics to be managed. The formal results presented here provide a basis for developing efficient security administration and management tools.

James Joshi's work has been supported by the U.S. National Science Foundation award IIS 0545912.
Authors' addresses: James B. D. Joshi, School of Information Sciences, University of Pittsburgh, Pennsylvania 15260; Elisa Bertino, Research Director of CERIAS, Department of Computer Science and School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907; Arif Ghafoor, School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907; Yue Zhang, Department of Computer Science, University of Pittsburgh, Pennsylvania 15260.

[AQ1] Categories and Subject Descriptors: ... [...]: ...
General Terms: Derived Hierarchy, Hierarchy Transformation
Additional Key Words and Phrases: Role-based access control, hybrid hierarchy, GTRBAC

ACM Reference Format:
Joshi, J. B. D., Bertino, E., Ghafoor, A., and Zhang, Y. 2007. Formal foundations for hybrid hierarchies in GTRBAC. ACM Trans. Inform. Syst. Secur. 10, 4, Article 14 (Nov. 2007), 39 pages. DOI = 10.1145/1284680.1284682 http://doi.acm.org/ 10.1145/1284680.1284682

1. INTRODUCTION

Role-based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access-control (DAC and MAC) models, [AQ2] which have inherent limitations [Giuri 1995, 1996; Joshi et al. 2001a; Nyanchama and Osborn 1999; Osborn et al. 2000; Sandhu et al. 1996; Koch et al. 2002]. Several beneficial features, such as policy neutrality, support for least privilege and efficient access, control management are associated with RBAC models [Ferraiolo et al. 1993; Joshi et al. 2001b; Sandhu et al. 1996]. Such features make RBAC better suited for handling access-control requirements of diverse organizations. RBAC models have also been found suitable for addressing security issues in the Internet environment [Barkley et al. 1997; Joshi et al. [AQ2] 2001a; Park et al. 2001] and show promise for newer heterogeneous multido- [AQ2] main environments that raise serious concerns related to access control across multiple domains [Biskup et al. 1998; Joshi et al. 2001b].

An essential part of an RBAC model is the notion of a role hierarchy. Role hierarchies play a crucial role in authorization management and administration [Moffett 1998; Sandhu et al. 1996, 1998; Jaeger and Tidswell 2001] and in the succinct RBAC representations of DAC and MAC policies [Osborn et al. 2000]. When two roles are hierarchically related, one is called the senior and the other the junior. In the most commonly accepted RBAC96 family of models [Sandhu et al. 1996], a senior and its junior roles are related by an inheritance relation that has two semantic parts: *permission-inheritance* (also called permission-usage [Sandhu 1998]) and *role-activation* semantics. *Permission-inheritance* semantics allows a senior role to inherit all the permissions assigned to its junior roles, whereas the *role-activation* semantics allows all the users assigned to a senior role to activate its junior roles. The RBAC96 models use the combined hierarchy semantics that allows both the *permission-inheritance* and the *role-activation* semantics. This significantly reduces assignment overhead, as the permissions need only be assigned to junior roles [Sandhu 1998; Moffett 1998]. Sandhu showed that, under the combined hierarchy semantics, certain *separation-of-duty* (SoD) constraints cannot be defined on hierarchically related roles, thus, restricting its effectiveness in supporting a broader set of fine-grained constraints and, in particular, in representing MAC policies [Sandhu 1998]. To address such shortcomings of RBAC96, Sandhu has proposed the ER-RBAC96 model that incorporates a distinction between a *usage* hierarchy that applies only the *permission-inheritance* semantics and an *activation* hierarchy that uses the combined hierarchy semantics [Sandhu 1998]. Later,

Joshi et al. have established a clear distinction between the three role hierarchies: *permission-inheritance*-only hierarchy (*I* hierarchy), *activation-only* hierarchy (*A* hierarchy), and the combined *permission-inheritance* and *activation* hierarchy (*IA* hierarchy) [Joshi et al. 2002]. The need for different semantics for hierarchical relations has also been recognized by Moffet et al. in Moffett [1998] and Moffett and Lupu [1999]. In particular, they have identified the need for three types of organizational hierarchies: *is a*, *activity*, and *supervision*, in order to address the needs of control principles in an organization, such as SoD, *decentralization of control* and *supervision* and *review*. Use of a combined hierarchy semantics has been found to limit a hierarchy in achieving these organizational control goals and, hence, to address such control requirements, it is desirable to configure a *hybrid* role hierarchy that allows different hierarchical relations among roles [Joshi et al. 2002]. Such a hybrid hierarchy is provided as part of the recently proposed *generalized temporal RBAC* (GTRBAC) model and it is able to support a variety of combinations of inheritance and activation semantics [Joshi et al. 2005b].

Another relevant functionality in access control is that of time-constraining accesses to resources for controlling time-sensitive activities in an application, for instance, in a *workflow management system* (WFMS) [Bertino and Ferrari 1999], where various workflow tasks, each having some timing constraints, need to be executed in some order. Bertino et al.'s temporal RBAC model (TRBAC) provided the first framework for modeling *time-constrained access policies* [Bertino et al. 2001]. The GTRBAC model extends the TRBAC model and incorporates a set of language constructs for specifying a large set of periodicity and duration constraints, including those on role enabling, user-role and role-permission assignments, and role activations. An important issue in the GTRBAC model is the interplay between the temporal constraints and role hierarchies, which has been first addressed in Joshi et al. [2002]. Accordingly, Joshi et al. identify various subtypes of the I, A, and IA hierarchies that capture temporal semantics of a hierarchy in presence of temporal constraints on roles. In the presence of a *hybrid* hierarchy containing multiple hierarchy types, a user may be able to activate different sets of junior roles in a session. Sets of roles that can be activated or permissions that can be acquired by a user at a particular time indicate the overall access capabilities of the user. From the perspective of the principle of least privilege, it may be necessary to ensure that such activable sets of roles do not result in granting users unnecessary access capabilities. Determining such sets can become very complex in the presence of a hybrid hierarchy. Furthermore, it is essential to know what indirect relations may exist between roles that are not directly related so that when modifications are made to the hierarchy, original relations can be maintained, if at all possible. For example, consider the relatively simple hybrid hierarchy of Figure 1. Here, determining the sets of roles that can be activated in a single session by a user assigned only to role r_3 is not straightforward. Similarly, when we delete the role s_1, we need to make sure that the original relations between r_3 and t_1, r_3 and s_2, or r_3 and x_1 are retained.

Flexible models, like GTRBAC, need formal tools for an efficient security administration and management. In this paper, we present a formal basis for

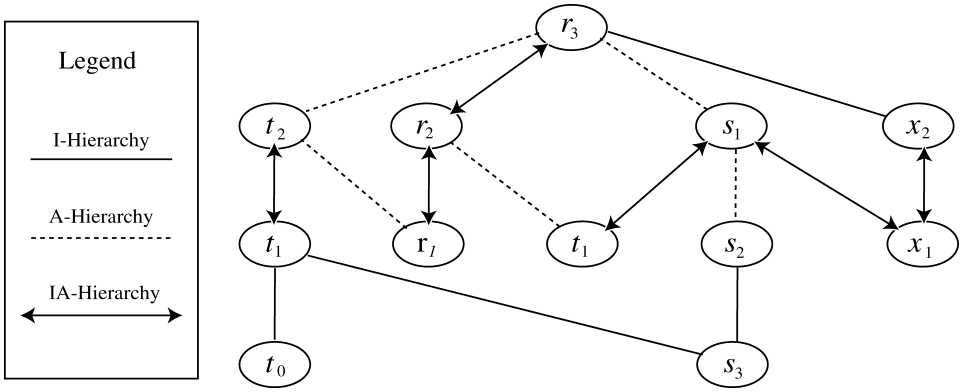

Fig. 1. An example hybrid hierarchy.

analyzing hybrid hierarchies in GTRBAC. The contributions of this paper include the following:

- We define the notion of uniquely activable set (UAS) of a hierarchy that can be used by security administrators for determining access capabilities that a user can obtain from a role hierarchy in a single session. We show formally how such a set can be determined in a hybrid role hierarchy.
- We introduce a set of inference rules that allows inferring the hierarchical relationships between an arbitrary pair of roles that are not directly related and show that it is *sound* and *complete*.
- We develop a set of hierarchy transformation algorithms to assist in administering role hierarchies when the roles are added, deleted, or modified.

The paper is organized as follows. In Section 2, we present an overview the GTRBAC model. In Section 3, we introduce the three basic hierarchical relations that can exist on a set of roles followed by their subtypes. In Section 4, we introduce the notion of UAS and present a formal technique for characterizing it. In Section 5, we introduce a set of inference rules for inferring derived relations between an arbitrary pair of roles. In Section 6, we introduce hierarchy transformation algorithms. Related work is discussed in Section 7. Conclusions and future work are presented in Section 8. The proofs for the theorems presented in this paper are available in the techical report version of the paper at https://www.cerias.purdue.edu/ with the same title.

2. GENERALIZED TEMPORAL ACCESS CONTROL MODEL (GTRBAC)

The GTRBAC 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. The GTRBAC model allows the specification of the following set of constraints:

1. *Temporal constraints on role enabling/disabling:* These constraints allow the specification of intervals and durations in which a role is enabled. When

Table I. Constraint Expressions

Constraint Categories	Constraints		Expression
Periodicity constraint	User-role assignment		$(I, P, pr$: assign$_U$/deassign$_U$ r *to* $u)$
	Role enabling		$(I, P, pr$: enable/disable $r)$
	Role-permission assignment		$(I, P, pr$: assign$_P$/deassign$_P$ p *to* $r)$
Duration constraints	User-role assignment		$([(I, P)\|D], D_U, pr$: assign$_U$/deassign$_U$ r *to* $u)$
	Role enabling		$([(I, P)\|D], D_R, pr$: enable/disable $r)$
	Role-permission assignment		$([(I, P)\|D], D_P, pr$: assign$_P$/deassign$_P$ p *to* $r)$
Duration constraints on Role activation	Total active role duration	Per-role	$([(I, P)\|D], D_{active}, [D_{default}], pr$: active$_{R_total}$ $r)$
		Per-user role	$([(I, P)\|D], D_{uactive}, u, pr$: active$_{UR_total}$ $r)$
	Max role duration per activation	Per-role	$([(I, P)\|D], D_{max}, pr$ active$_{R_max}$ $r)$
		Per-user role	$([(I, P)\|D], D_{umax}, u, pr$ active$_{UR_max}$ $r)$
Cardinality constraint on role activation	Total no. of activations	Per Role	$([(I, P)\|D], N_{active}, [N_{default}], pr$: active$_{R_n}$ $r)$
		Per-user role	$([(I, P)\|D], N_{uactive}, u, pr$: active$_{UR_n}$ $r)$
	Max. no. of concurrent activations	Per-role	$([(I, P)\|D], N_{max}, [N_{default}], pr$: active$_{R_con}$ $r)$
		Per-user role	$([(I, P)\|D], N_{umax}, u, pr$: active$_{UR_con}$ $r)$
Trigger	$E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow pr : E$ after Δt		
Constraint enabling	pr : enable/disable c where $c \in \{(D, D_x, pr : E), (C), (D, C)\}$		
Run-time requests	Users' activation request		$(s : (de)activate\ r$ for u after $\Delta t))$
	Administrator's run-time request		$(pr$: assign$_U$/de-assign$_U$ r to u after $\Delta t)$
			$(pr$: enable/disable r after $\Delta t)$
			$(pr$: assign$_P$/de-assign$_P$ p to r after $\Delta t)$
			$(pr$: enable/disable c after $\Delta t)$

a role is enabled, the permissions assigned to it can be acquired by a user by activating it. When a duration constraint is specified, the enabling/disabling of a role is initiated by a constraint-enabling event that results from the firing of a trigger or through an administrator-initiated run-time event.

2. *Temporal constraints on user-role and role-permission assignments:* These constraints allow specifying intervals and durations in which a user or a permission is assigned to a role.
3. *Activation constraints:* These constraints allow specification of restrictions on the activation of a role. These include, for example, specifying the total duration for which a user may activate a role or the number of concurrent activations of a role at a particular time.
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 request activation or deactivation of a role.
5. *Constraint-enabling expressions:* The GTRBAC model includes events that enable or disable duration and role-activation constraints mentioned earlier.
6. *Triggers:* The GTRBAC triggers allow expressing dependencies among events.

Table I summarizes the constraint types and expressions of the GTRBAC model. The periodic expression used in the constraint expressions is of the form

Table II. An Example GTRBAC Access Policy for a Medical Information System

1	a.	(*DayTime*, `enable` **DayDoctor**), (*NightTime*, `enable` **NightDoctor**), (*AllTime*, pr_1: `disable` **NurseInTraining**), $pr_2 > pr_1$
	b.	((M, W, F), `assign`$_U$ *Adams* `to` **DayDoctor**)
		((T, Th, S, Su), `assign`$_U$ *Bill* `to` **DayDoctor**);
		(M, W, F), `assign`$_U$ *Alice* `to` **NightDoctor**)
		((T, Th, S, Su), `assign`$_U$ *Ben* `to` **NightDoctor**)
	c.	([10 AM, 3 PM], `assign`$_U$ *Carol* `to` **DayDoctor**)
2	a.	(`assign`$_U$ *Ami* `to` **NurseInTraining**); (`assign`$_U$ *Elizabeth to* **DayNurse**)
	b.	$c_1 = (6\ hr, 2\ hr, pr_2$: `enable` **NurseInTraining**)
3	a.	(`enable` **DayNurse** → `enable` c_1)
	b.	(`activate` **DayNurse** `for` *Elizabeth* → `enable` **NurseInTraining** `after` 10 *min*)
	c.	(`enable` **NightDoctor** → `enable` **NightNurse** `after` 10 *min*);
		(`disable` **NightDoctor** → `disable` **NightNurse** `after` 10 *min*)
	d.	(`enable` **DayDoctor** → `enable` **DayNurse** `after` 10 *min*);
		(`disable` **DayDoctor** → `disable` **DayNurse** `after` 10 *min*)

(I, P), where P is an *expression* denoting an infinite set of periodic intervals and $I =$ [`begin`, `end`] is a time interval denoting the lower and upper bounds that are imposed on instants in P. The function $Sol(I, P)$ is used to denote all the time instants in (I, P). D expresses the duration specified for a constraint. In the duration and role-activation constraint expressions, D_x and N_x indicate the duration and cardinality values. If the subscript x starts with u, then it is a *per-user-role* constraint otherwise it is a *per-role* constraint. For instance, D_{active} indicates the duration for which the specified role can be active, whereas, $D_{uactive}$ indicates the duration for which the specified user may activate the specified role. The following example illustrates the specification of a GTRBAC policy. For more details on the GTRBAC model, we refer the readers to Joshi et al. [2005b].

Example 1. Table II contains the GTRBAC policy for a hospital. The periodicity constraint 1*a* specifies the enabling times of **DayDoctor** and **NightDoctor** roles. For simplicity, we use *DayTime* and *NightTime* instead of their (I, P) forms. The periodicity constraint 1b allows the **DayDoctor** role to be assigned to *Adams* on *Mondays*, *Wednesdays*, and *Fridays*, and to *Bill* on *Tuesdays*, *Thursdays*, *Saturdays*, and *Sundays*. Similarly, *Alice* and *Ben* are assigned to the **NightDoctor** role on the different days of the week. Furthermore, the assignment in 1c allows *Carol* to assume the **DayDoctor** role everyday between 10 AM and 3 PM. In 2a, *Ami* and *Elizabeth* are assigned to roles **NurseInTraining** and **DayNurse** respectively, with no temporal restriction, i.e., the assignment is valid at all times. 2b specifies a duration constraint of 2 *hr* on the enabling time of the **NurseInTraining** role, but this constraint is valid for only 6 *hr* after the constraint c_1 has been enabled. Because of this, *Ami* will be able to activate the **NurseInTraining** role, at most, for 2 *hr* whenever constraint c_1 is enabled. In row 3, we have a set of triggers. Trigger 3a indicates that constraint c_1 is enabled when the **DayNurse** is enabled, which means, now, the **NurseInTraining** role, can be enabled for, at most, 2 *hr* within the next 6 *hr* (Note that after the next 6 *hr*, the **NurseInTraining** role can be enabled for any duration of time). Trigger 3b indicates that 10 *min* after *Elizabeth* activates

Table III. Various Status Predicates

Predicate	Meaning
$enabled(r, t)$	Role r is enabled at time t
$u\_assigned(u, r, t)$	User u is assigned to role r at time t
$p\_assigned(p, r, t)$	Permission p is assigned to role r at time t
$can\_activate(u, r, t)$	User u can activate role r at time t
$can\_acquire(u, p, t)$	User u can acquire permission p at time t
$can\_be\_acquired(p, r, t)$	Permission p can be acquired through role r at time t
$active(u, r, s, t)$	Role r is active in user u's session s at time t
$acquires(u, p, s, t)$	User u acquires permission p in session s at time t

the **DayNurse** role, the **NurseInTraining** role is enabled. This shows that a nurse in training will have access to the system only if *Elizabeth* is present in the system, that is, she may be acting as a training supervisor. The remaining triggers in row 3 show that the **DayNurse** and **NightNurse** roles are enabled (disabled) 10 *min* after the **DayDoctor** and **NightDoctor** roles are enabled (disabled).

3. HYBRID ROLE HIERARCHIES

In an earlier work, we have introduced the following three hierarchy types, mentioned earlier: *permission-inheritance-only* hierarchy (*I* hierarchy), *role-activation-only* hierarchy (*A* hierarchy), and the combined *permission-inheritance-activation* hierarchy (*IA* hierarchy) [Joshi et al. 2002]. Table III shows the notation for various predicates used in the definitions of these hierarchies. Predicates $enabled(r, t)$, $assigned(u, r, t)$ and $assigned(p, r, t)$ refer to the status of roles, user-role and role-permission assignments at time t. Predicate $can\_activate(u, r, t)$ indicates that user u can activate role r at time t. This implies that user u is implicitly or explicitly assigned to role r. $active(u, r, s, t)$ indicates that role r is active in user u's session s at time t whereas, $acquires(u, p, s, t)$ implies that u acquires permission p at time t in session s. The axioms below capture the key relationships among these predicates and precisely identify the permission-acquisition and role-activation semantics allowed in the GTRBAC model [Joshi et al. 2002].

AXIOMS. *If* $r \in$ `Roles`, $u \in$ `Users`, $p \in$ `Permissions`, $s \in$ `Sessions`, *and time instant* $t \geq 0$, *the following implications hold*:

1. $p\_assigned(p, r, t) \rightarrow can\_be\_acquired(p, r, t)$
2. $u\_assigned(u, r, t) \rightarrow can\_activate(u, r, t)$
3. $can\_activate(u, r, t) \wedge can\_be\_acquired(p, r, t) \rightarrow can\_acquire(u, p, t)$
4. $active(u, r, s, t) \wedge can\_be\_acquired(p, r, t) \rightarrow acquires(u, p, s, t)$

Axiom (1) states that if a permission is assigned to a role, then it *can be acquired* through that role. Axiom (2) states that all users assigned to a role *can activate* that role. Axiom (3) states that if a user u can activate a role r, then all the permissions that *can be acquired* through r *can be acquired* by u. Similarly, axiom (4) states that if there is a user session in which a user u has activated a role r then u *acquires* all the permissions that *can be acquired*

through role r. We note that axioms (1) and (2) indicate that the semantics for the *permission-acquisition* and *role-activation* is governed by explicit user-role and role-permission assignments.

3.1 Formal Definitions of Temporal Role Hierarchies

Semantically, the use of a role hierarchy is to extend the possibility of permission-acquisition and role-activation semantics beyond the explicit assignments as indicated by the definitions below [Joshi et al. 2002]. The GTRBAC model's constraint enabling/disabling expressions can be used to specify when a hierarchical relation can be enabled/disabled. Hence, if h is a hierarchical relation, we write "*enable*/*disable h*" to enable/disable the relation. This allows administrators to dynamically change, if needed, the hierarchical relationships on a set of roles through periodicity or duration constraints, run-time requests and triggers. The following definitions do not consider the enabling times of the hierarchically related roles and, hence, the hierarchies are termed *unrestricted*.

Definition 3.1 (*Unrestricted I Hierarchy* [*Joshi et al.* 2002]). Let x and y be roles such that $(x \geq_{\mathrm{i}} y)$, that is, x has a permission inheritance-only relation over y at time t. Then the following holds:

$$\forall p,\ (x \geq_{\mathrm{i}} y) \wedge can\_be\_acquired(p, y, t) \rightarrow can\_be\_acquired(p, x, t) \quad (c_1)$$

Definition 3.2 (*Unrestricted A Hierarchy* [*Joshi et al.* 2002]). Let x and y be roles such that $(x \geq_a y)$, that is, x has an activation-only relation over y at time t. Then the following holds:

$$\forall u,\ (x \geq_{\mathrm{a}} y) \wedge can\_activate(u, x, t) \rightarrow can\_activate(u, y, t) \quad (c_2)$$

Definition 3.3 (*Unrestricted IA Hierarchy*). Let x and y be roles such that $(x \geq y)$, that is, x has a general inheritance relation over y at time t. Then the following holds:

$$\begin{aligned} &\forall p, \forall u, ((x \geq y) \wedge can\_be\_acquired(p, y, t) \rightarrow can\_be\_acquired(p, x, t)) \\ &\quad \wedge ((x \geq y) \wedge can\_activate(u, x, t) \rightarrow can\_activate(u, y, t)) \end{aligned} \quad (c_3)$$

In the definitions above, x is said to be a senior of y and, conversely, y is said to be a junior of x. Thus, if $(x \geq_i y)$, the permissions that can be acquired through x, include all the permissions assigned to x (by axiom (1)) and all the permissions that *can be acquired* through role y (by c_1). Condition c_2 states that if user u can activate role x, and x has A-relation over y, then u can also activate role y, even if u is not explicitly assigned to y. The *IA* is the most common form of hierarchy. On a given set of roles, various inheritance relations may be defined. Therefore, we require that the following *consistency* property be satisfied in a role hierarchy in order to ensure that the senior–junior relationship between two roles in one type of hierarchy is not reversed in another. Note that all three hierarchies are transitive.

PROPERTY 1 (CONSISTENCY OF HIERARCHIES [JOSHI ET AL. 2005A]). *Let* $\langle f_1 \rangle$, $\langle f_2 \rangle \in \{\geq_i, \geq_a, \geq\}$. *Let x and y be two distinct roles such that* $(x \langle f_1 \rangle y)$; then the condition $\neg(y \langle f_2 \rangle x)$ must hold.

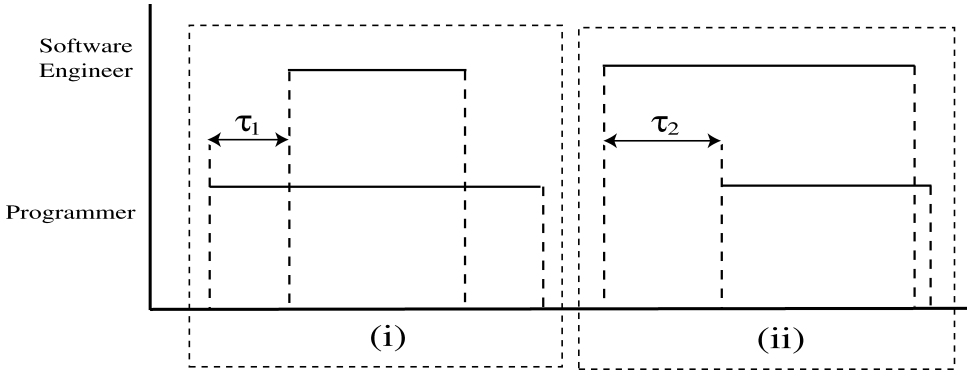

Fig. 2. Enabling intervals of software engineer and programmer roles.

Table IV. Inheritance Semantics for the *Weakly* and *Strongly Restricted* Hierarchies[a]

Interval $\tau \rightarrow$ $\downarrow$ Hierarchy Type		$\tau = \tau_1$ r_1 disabled, r_2 enabled	$\tau = \tau_2$ r_1 enabled, r_2 disabled
I hierarchy	I_w	No inheritance in τ	Permission-inheritance in τ
	I_s	No inheritance in τ	No inheritance in τ
A hierarchy	A_w	Activation-inheritance in τ	No inheritance in τ
	A_s	No inheritance in τ	No inheritance in τ
$I\,A$ hierarchy	IA_w	Activation-inheritance in τ	Permission-inheritance in τ
	IA_s	No inheritance in τ	No inheritance in τ

[a]Note: w means *weakly-restricted* and s means *strongly-restricted*.

In what follows, we will always assume that hierarchies are consistent. When we consider the enabling times of hierarchically related roles, we obtain *weakly restricted* and *strongly restricted* forms of the hierarchies. Their meaning is exemplified in Figure 2, in which roles—`Software Engineer` and `Programmer`—are hierarchically related. Of those two roles, only one is enabled in intervals τ_1 and τ_2. In a *strongly restricted* hierarchy, inheritance is not allowed in these intervals. This is because, in this type of hierarchy, both roles must be enabled for inheritance to take place. By contrast, in a *weakly restricted* hierarchy, inheritance may be allowed in these intervals. Table IV shows the inheritance properties of *weakly restricted* and *strongly restricted* hierarchies in τ_1 and τ_2, when r_1 is senior of r_2.

When activation-time restrictions are to be enforced in GTRBAC, different hierarchy types may need to be considered depending upon whether the constraint is *user-* or *permission-centric* [Joshi et al. 2002]. An activation constraint is *user-centric* if it is designed to control different aspects of users in the system through role activations, for example, to control the number of users activating a role. An activation constraint is *permission-centric* if it is aimed at controlling distribution of the permissions through role activations. [Joshi et al. 2002]. show that an I or IA hierarchy is appropriate when an activation constraint is *user-centric*, whereas an A hierarchy is appropriate when the activation constraint is *permission centric*.

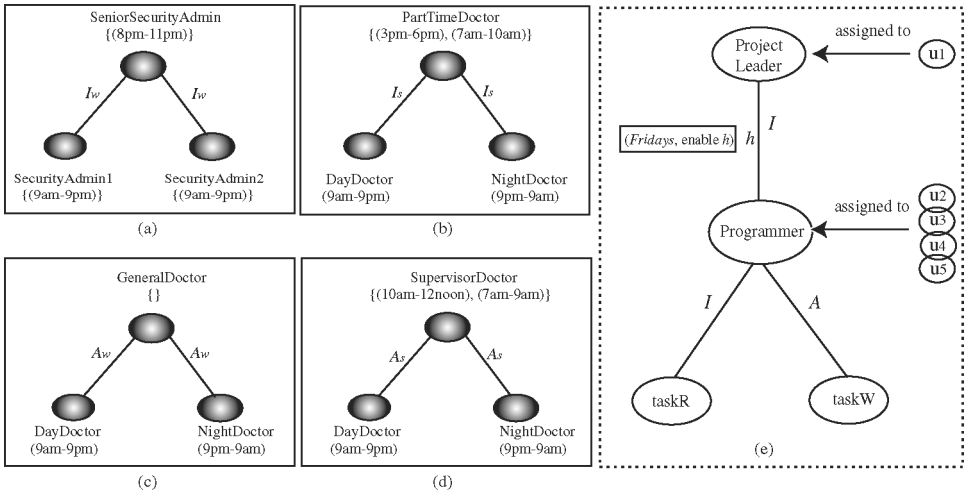

Fig. 3. Hierarchy examples.

3.2 Examples of Temporal Role Hierarchies

We illustrate with the examples 3.1 and 3.2 that refers to Figure 3 the practical uses of the various kinds of hierarchies.

Example 3.1. Consider the hierarchy in Figure 3(a). Here, we see that the `SeniorSecurityAdmin` role is enabled only in interval (8 PM, 11 PM). Neither of its junior roles is enabled in the entire interval (8 PM, 11 PM). The I_w relation allows a user who activates the `SeniorSecurityAdmin` role to acquire all the permissions of its junior roles. This may be desirable if `SeniorSecurityAdmin` role is designed to perform special security operations for checking and maintenance. In such a case, it is reasonable to think that the user assigned to the `SeniorSecurityAdmin` role will need all the administrative privileges of the junior roles. The temporal restrictions on `SecurityAdmin1` and `SecurityAdmin2` constrain the users assigned to them to carry out corresponding system administration activities only in the specified intervals. However, here, the user assigned to `SeniorSecurityAdmin` cannot assume the role of the junior roles `SecurityAdmin1` and `SecurityAdmin2`. To remove this limitation, we can use the IA_w hierarchy instead. The hierarchy in Figure 3b, on the other hand, is of type I. The senior role is the `PartTimeDoctor` role, which has two intervals in which it can be enabled, (3 PM, 6 PM) and (7 AM, 10 AM). If a user activates the `PartTimeDoctor` role in the first interval, according to the I_s relation, he essentially gets all the privileges of the `DayDoctor` role, as the `NightDoctor` role is disabled at that time. Now, consider the second interval. We see that it overlaps with the enabling times of the two junior roles. Hence, if the user activates the `PartTimeDoctor` role in the second interval, he acquires the privileges of only the `NightDoctor` role in the subinterval (7 AM, 9 AM) and that of only the `DayDoctor` role in the subinterval (9 AM, 10 AM). Thus, we see that the two different semantics of an inheritance hierarchy can be used to achieve different

needs. Again, a part-time doctor cannot work as a DayDoctor or a NightDoctor, although he can acquire the permissions assigned to them. If a user is also to be allowed to use the junior roles, we can use IA_s hierarchy instead. Now, consider Figure 3c. Here we see that there is no interval in which the GeneralDoctor role can be enabled. However, since the activation hierarchy is of type A_w, any user assigned to the GeneralDoctor role can activate either of the junior roles when they are enabled. In effect, any user assigned to the GeneralDoctor role can activate both the DayDoctor and the NightDoctor roles whenever they are enabled. Figure 3d illustrates the use of an activation hierarchy of type A_s. Here, a supervising doctor can assume the SupervisorDoctor role in intervals (10 AM, 12 *noon*) and (7 AM, 9 AM). In the first interval, the supervisor will be able to acquire all the privileges of the DayDoctor role by activating it in the second interval, he will be able to acquire all the privileges of the NightDoctor role by activating it along with the SupervisorDoctor role. The SupervisorDoctor role may simply contain some extra privileges that are required for the supervision task during daytime or nighttime. [Moffett 1998]. has identified such a *supervision-review* capability as an important organizational control principle.

Example 3.2. Consider the following requirements for a programming project. A software tool is used for the programming task. The project leader mainly supervises the programming tasks. Only the programmers do the coding. The project leader can only look at the tasks the programmers have carried out on a weekly basis, say on *Fridays*. Figure 3e depicts the hierarchy that can be generated for achieving the goal. Role TaskR contains the read-only permissions whereas role TaskW contains all the write/modify permissions related to the programming task. The Project Leader role becomes the senior of Programmer role only on *Fridays*. Note that the users assigned to the Project Leader can acquire permissions of TaskR but not of TaskW.

4. UNIQUELY ACTIVABLE SET OF A HIERARCHY

In this section, we introduce the notion of UAS and present a formal approach for characterizing it for a hierarchy. The UAS associated with a role in a hierarchy is essentially the set of *role sets* that can be activated by a user assigned only to that role. In a hierarchy that allows coexistence of the multiple hierarchy types, the *permission-inheritance* and *role-activation* semantics can be complex, thus making administration and management of large hierarchies difficult. The UAS gives the role combinations that can be activated by a user in a single session and thus helps in determining the granularity of permission sets that can be acquired by users through a role in the hierarchy. Thus, UAS is mainly relevant from the perspective of the *principle of least privilege*. Here, we first determine the UAS characteristics of a *monotype* hierarchy with only one type of hierarchical relation over the roles, followed by that of a *hybrid* linear path and then formalize results for the more general role hierarchy. The approach to determining the UAS presented in this section is algorithmic in nature. A mathematical (declarative) way of establishing the UAS is presented in Appendix C. We then introduce the notion of *acquisition equivalence* to characterize equivalent hierarchies in order to address the usefulness

of a hybrid hierarchy. Here onward we will only use the *unrestricted* forms of hierarchies.

4.1 Computing Uniquely Activable Set of a Hierarchy

We represent by $\bigsqcup(H)$ the UAS associated with a user assigned to the senior-most role of a hierarchy H. For a given role set $X = \{x_1, x_2, \ldots, x_n\}$ and a set of hierarchy relations $[f] \subseteq \{\geq_i, \geq_a, \geq\}$, we represent a general hierarchy H over X as $(X, [f])$. If $[f] = \{\langle f \rangle\}$ is a singleton set with hierarchy relation $\langle f \rangle$, then we call H a *monotype hierarchy* and write $(X, \langle f \rangle)$, else we call H a *hybrid hierarchy*. Furthermore, H is a linear path over X if $(X, [f])$ is an ordered sequence of relations $x_1\langle f_{12}\rangle x_2\langle f_{23}\rangle x_3 \ldots x_{n-1}\langle f_{(n-1)n}\rangle x_n$, where $\langle f_{ij}\rangle \in [f]$. We represent a monotype linear path as $L = (X, \langle f \rangle)$ and a hybrid linear path as $Lh = (X, [f])$. We use LH to represent either L or Lh. H represents any hierarchy. For $H = \langle X, [f]\rangle$, $Role(H) = X$. In this paper, we assume that (a) the set of permissions assigned to each role in $\texttt{Roles}(H)$ is distinct, and (b) hierarchy H has only one senior-most role, indicated by S_H.

The results can be easily extended to deal with a general hierarchy. We use J_H to denote the set of junior-most roles of H. We use notation $P(r)$ to refer to the set of permissions that are available through $r(P(r) = \{p | can\_be\_acquired(r, p)\})$. Similarly, given a set X of roles, we use $P(X)$ to denote $\bigcup_{r \in X} P(r)$. We formally define the UAS of a hierarchy as follows.

Definition 4.1 (*Uniquely Activable Set of a Hierarchy H*). Let H = (X, [f]) be a hierarchy. Then, the UAS for a user u assigned only to role S_H, $\bigsqcup$(H), is the maximal set of role sets $Y_1, Y_2, \ldots, Y_m$, such that

1. for each $i \in \{1, 2, \ldots, m\}$, $\emptyset \subset Y_i \subseteq X$, and all roles in each Y_i can be activated in a single session of u,
2. for all pairs $i, j \in \{1, 2, \ldots, m\}$ and $i \neq j$, $P(Y_i) \neq P(Y_j)$, and,
3. for each $Z \subseteq X$ such that $Z \notin \bigsqcup(H)$, if $P(Y_i) = P(Z)$ for some i, then $(|Y_i| < |Z|)$ (where $|A|$ denotes the cardinality of set A).

Note that each element Y_i is a subset of X. Condition (2) indicates that each role set of $\bigsqcup(H)$ is unique in terms of the permissions that are available through its roles. Condition (3) considers the possibility of different role sets associated with the same set of permissions. In such a case, $\bigsqcup(H)$ contains the role set that has the least number of roles. Conditions (2) and (3) prevent a pair of senior and junior roles, e.g., of an *IA* hierarchy, to be in a role set of $\bigsqcup(H)$. For instance, if relation $(x \geq y)$ is in H, then the set $\{x\}$ and not $\{x, y\}$ will be in $\bigsqcup(H)$, as $P(x) = P(\{x, y\})$. The $\bigsqcup(H)$ values for I, A, and IA hierarchy can differ significantly because of the difference in *permission-inheritance* and *role-activation* semantics associated with them.

As a *hybrid* linear path may have different types of hierarchical relations, it can be decomposed into a set of *monotype* linear paths. The following definition formalizes the notion of *monotype decomposition of a hybrid linear path* (MDHP).

We denote the senior- and the junior-most roles of a *hybrid linear path Lh* as S_{Lh} and J_{Lh}, respectively.

Definition 4.2 (*Monotype Decomposition of Hybrid Path—MDHP*). Let $Lh = (X, [f])$ be a hybrid linear path over role set X. Then Lh can be decomposed into an ordered set $Lh = (L_1, L_2, \ldots, L_n)$ with $X = X_1 \bigcup X_2 \bigcup .. \bigcup X_n$, such that $L_i = (X_i, \langle f_i \rangle)$ is a monotype linear path, and the following conditions hold:

(1) for all $i \in \{1, \ldots, n-1\}$, (i) $\langle f_i \rangle \neq \langle f_{i+1} \rangle$, *and* (ii) $X_i \bigcap X_{i+1} = \{J_{L_i}\} = \{S_{L_{(i+1)}}\}$, and
(2) for all $i \in \{1, \ldots, n\}$ and $(i+1 < j \leq n)$ or $(1 \leq j < i-1)$, $X_i \bigcap X_j = \phi$,

Here, $(L_1, L_2, \ldots, L_n)$ is minimal MDHP. Lh can also be written as (L_1, Lh'), (Lh'', L_n), or (Lh_x, Lh_y), each of which is a decomposition of Lh.

It is easy to see that $S_{Lh} = S_{L_1}$, and $J_{Lh} = J_{L_n}$. As indicated by definition 4.2, we can break a *hybrid* linear path into an ordered set of *monotype* linear paths. Such an MDHP of a *hybrid* path allows us to use the UAS of the *monotype* linear paths to determine the UAS of a *hybrid* linear path. Note that the *minimal* MDHP consists of *monotype* linear paths that are maximal in the sense that combining any consecutive pair of component linear paths will give a component *hybrid* linear path, as indicated by part (1) of the definition. Example 4.1 illustrates the decomposition of a *hybrid* linear path into its monotype components.

Example 4.1. Consider the role hierarchy of Figure 4a. The complete MDHP of the *hybrid* linear path is $(L_1, L_2, L_3, L_4, L_5, L_6)$, as shown in Figure 4c. We note that if L_4 is split into $L_{4,1} = (\{4, 5\}, IA\text{-type})$ and $L_{4,2} = (\{5, 6\}, IA \text{ type})$, then $L_1, L_2, L_3, L_{4,1}, L_{4,2}, L_5, L_6$ is not a MDHP, as $L_{4,1}$ and $L_{4,2}$ do not satisfy part (1) of definition 4.2.

In this paper, we also use functions $sub_L(LH)$ and $sub_U(LH)$ that return the lower and upper parts of a linear path LH. That is, if LH has $x_1\langle f_{12}\rangle x_2\langle f_{23}\rangle x_3 \cdots x_{n-1}\langle f_{(n-1)n}\rangle x_n$, *where* $\langle f_{i(i+1)}\rangle \in \{\geq_i, \geq_a, \geq\}$ and $(L_1, L_2, \ldots, L_n)$ *is its MDHP*, then,

- $sub_L(LH) = x_2\langle f_{23}\rangle x_3 \cdots x_{n-1}\langle f_{(n-1)n}\rangle x_n = (sub_L(L_1), L_2, \ldots, L_n)$
- $sub_U(LH) = x_1\langle f_{12}\rangle x_2\langle f_{23}\rangle x_3 \cdots \langle f_{n-2}\rangle x_{n-1} = (L_1, L_2, \ldots, sub_U(L_n))$
- For $L = (x\langle f \rangle y)$, $sub_L(L) = sub_U(L) = \emptyset$

Here, $sub_L(LH)$ and $sub_U(LH)$ return the lower and the upper subpaths of LH. $sub_L(x\langle f \rangle y) = sub_U(x\langle f \rangle y) = \emptyset$ indicates that path $(x\langle f \rangle y)$ has no subpaths.

Because of the different activation semantics associated with each hierarchy type, UAS associated with each type is also different. The following theorem formally characterizes the UAS of a *monotype* hierarchy, H:

THEOREM 4.4. *Let* $H = (X, \langle f \rangle)$ *be a monotype linear hierarchy defined over role set* $X = \{x_1, x_2, \ldots, x_n\}$ *with* $\langle f \rangle \in \{\geq_i, \geq_a, \geq\}$, *then*

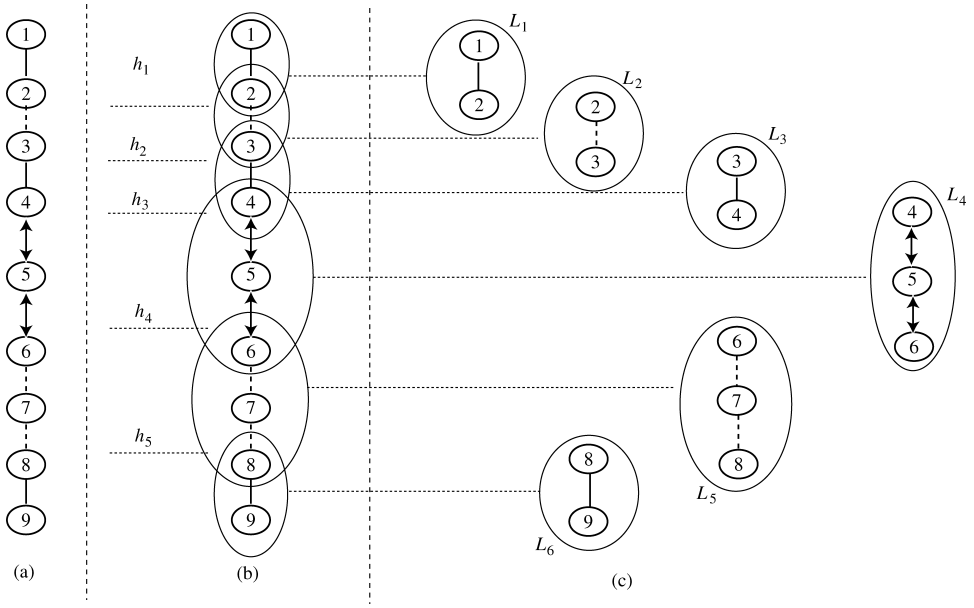

Fig. 4. Complete horizontal partition of a *hybrid* linear path.

$$\bigsqcup(\mathrm{H}) = \begin{cases} \{S_H\} & if(\langle \mathrm{f} \rangle = \geq_i) \\ \{2^X \backslash \emptyset\} & if(\langle \mathrm{f} \rangle = \geq_a) \\ \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\} & if(\langle \mathrm{f} \rangle = \geq) \end{cases}$$

The theorem states that if H is an I hierarchy, $\bigsqcup(H)$ contains the senior-most role only. If H is an A hierarchy, $\bigsqcup(H)$ contains the power set of the role set X without the empty element, i.e., the senior-most role can activate every combination of the roles in the hierarchy. Similarly, if H is an IA hierarchy, $\bigsqcup(H)$ contains set elements containing individual roles of the hierarchy. The proof for the theorem follows directly from the transitive properties of the hierarchical relations and the *permission-inheritance-only* and/or *role-activation-only* semantics of the three hierarchies. The following example illustrates the use of the results of Theorem 4.1.

Example 4.2. Consider the monotype hierarchies of Figure 3. For each of the monotype hierarchies in Figures 3a and 3b, the corresponding UAS only contain the set with the senior-most role of the hierarchy, as each of them has the senior-most role related to its junior(s) by I-relation(s). For hierarchies in Figures 3c and 3d, assuming *unrestricted* forms in both the cases, instead of the *restricted* forms indicated in the figures, the UASs are as follows:

Hierarchy of Figure 3c: Here,

$\bigsqcup(H)$= {{GeneralDoctor}, {DayDoctor}, {NightDoctor}, {GeneralDoctor, DayDoctor}, {GeneralDoctor,NightDoctor}, {GeneralDoctor, DayDoctor,NightDoctor}, {DayDoctor,NightDoctor}}.

However, `GeneralDoctor` is never enabled.
If we do not consider temporal constraints, the UAS of hierarchy in Figure 3d is:

$\bigsqcup(H)$= {{`SupervisorDoctor`}, {`DayDoctor`}, {`NightDoctor`},
{`SupervisorDoctor,DayDoctor`}, {`SupervisorDoctor,NightDoctor`},
{`SupervisorDoctor,DayDoctor,NightDoctor`}, {`DayDoctor,`}.
{`NightDoctor`}}.

Next, we present a formal basis for characterizing $\bigsqcup(H)$ for a *hybrid* linear path. We first present the results for a *hybrid* linear path consisting of only two *monotype* linear components in the following lemma and then use it to characterize arbitrary *hybrid* linear paths.

LEMMA 4.1. *Let* $Lh = (L_1, L_2)$ *be a hybrid linear path such that* $L_1 = (X_1, \langle f \rangle)$ *and* $L_2 = (X_2, \langle f_2 \rangle)$*, where* $X = \{x_1, x_2, \ldots, x_n\} = X_1 \bigcup X_2$*, and* $\langle f_1 \rangle \neq \langle f_2 \rangle$*. Then for a user u assigned only to* S_{L_1}*, we have:*

$$\bigsqcup(Lh) = \begin{cases} \bigsqcup(L_1) & \text{if } \geq_i \in \{\langle f_1 \rangle, \langle f_2 \rangle\} \\ \bigsqcup(L_1^U) \bigcup \bigsqcup(L_2) \bigcup (\bigsqcup(L_1^U) \bigotimes \bigsqcup(L_2)) & \text{if } (\langle f_1 \rangle, \langle f_2 \rangle) = (\geq_a, \geq) \\ \bigsqcup(L_1) \bigcup \bigsqcup(L_2^L) \bigcup (\bigsqcup(L_1) \bigotimes \bigsqcup(L_2^L)) & \text{if } (\langle f_1 \rangle, \langle f_2 \rangle) = (\geq, \geq_a) \end{cases}$$

where, $L_2^L = sub_L(L_2), L_2^U = sub_U(L_2)$ *and* $A \bigotimes B = \{\{x \bigcup y\} \mid x \in A \text{ and } y \in B\}$.

Note that in the computation involving $\bigsqcup(Lh)$, the components on the right side are disjoint with respect to each other and, hence, $|\bigsqcup(Lh)|$ is simply the sum of the cardinalities of the components on the right side. Theorem 4.2 determines the $\bigsqcup(H)$ for an arbitrary *hybrid* linear path.

THEOREM 4.2. *Let* $Lh = (L_1, LH_2)$ *be a hybrid linear path such that* $L_1 = (X_1, \langle f_1 \rangle)$*,* LH_2 *is a linear path over* X_2*, where* X_1 *and* X_2 *are role sets, and* $X = X_1 \bigcup X_2$*. Furthermore, let* $LH_2 = (L_x, LH')$*, where* $L_x = (X_x, \langle f_x \rangle)$ *over role set* X_x *such that* $\langle f_x \rangle \neq \langle f_1 \rangle$ *and* LH' *is a linear path, possibly empty. Then, we have the following:*

1. *if* $\langle f_1 \rangle = \geq_i$ *then* $\bigsqcup(Lh) = \bigsqcup(L_1)$
2. *if* $\langle f_1 \rangle = \geq_a$ *then*
$$\bigsqcup(Lh) = \begin{cases} \bigsqcup(L_1) & \text{if } (\langle f_x \rangle = \geq_i) \\ \bigsqcup(L_1^U) \bigcup \bigsqcup(LH_2) \bigcup (\bigsqcup(L_1^U) \bigotimes \bigsqcup(LH_2)) & \text{if } (\langle f_x \rangle = \geq) \end{cases}$$
3. *if* $\langle f_1 \rangle = \geq$ *then*
$$\bigsqcup(Lh) = \begin{cases} \bigsqcup(L_1) & \text{if } (\langle f_x \rangle = \geq_i) \\ \bigsqcup(L_1) \bigcup \bigsqcup(LH_2^L) \bigcup (\bigsqcup(L_1) \bigotimes \bigsqcup(LH_2^L)) & \text{if } (\langle f_x \rangle = \geq_a) \end{cases}$$

The next example illustrates the use of the above theorem and refers to Figure 5. We note that to compute $\bigsqcup(H)$ for the hierarchy in case (c), we need to first compute for cases (a) and (b).

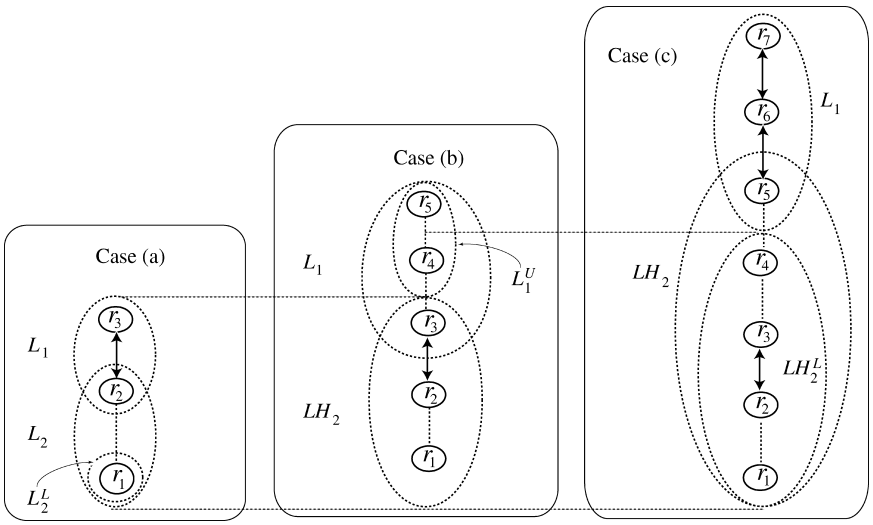

Fig. 5. Computing UAS for a hybrid linear hierarchy

Case a. Here L_1 has $r_3 \geq r_2$, and L_2 has $r_2 \geq_a r_1$, i.e. $(\langle f_1 \rangle, \langle f_2 \rangle) = (\geq, \geq_a)$. Therefore, by lemma 4.1, we have,

$$\begin{aligned}\bigsqcup(Lh) &= \bigsqcup(L_1) \bigcup \bigsqcup(L_2^L) \bigcup \left(\bigsqcup(L_1) \bigotimes \bigsqcup(L_2^L)\right) \\ &= \{\{r_2\}, \{r_3\}\} \bigcup \{\{r_1\}\} \bigcup (\{\{r_2\}, \{r_3\}\} \bigotimes \{\{r_1\}\}) \\ &= \{\{r_1\}, \{r_2\}, \{r_3\}, \{\{r_1\}, \{r_2\}\}, \{r_1, r_3\}\}\end{aligned}$$

Case b. Here L_1 has $r_5 \geq_a r_4 \geq_a r_3$, and LH_2 is the hierarchy in (a). Now, we apply Theorem 4.2. As $\langle f_1 \rangle = \geq_a$, case (2) of the theorem applies. Thus,

$$\begin{aligned}\bigsqcup(Lh) &= \bigsqcup(L_1^U) \bigcup \bigsqcup(LH_2) \bigcup \bigsqcup(L_1^U) \bigotimes \bigsqcup(LH_2) \\ &= \{\{r_4\}, \{r_5\}, \{r_4, r_5\}\} \bigcup \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}\} \bigcup (\{\{r_4\}, \{r_5\}, \\ &\quad \{r_4, r_5\}\} \bigotimes \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}\}) \\ &= \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}, \{r_4\}, \{r_5\}, \{r_4, r_5\}, \{r_1, r_4\}, \{r_1, r_5\}, \\ &\quad \{r_1, r_4, r_5\}, \{r_2, r_4\}, \{r_2, r_5\}, \{r_2, r_4, r_5\}, \{r_3, r_4\}, \{r_3, r_5\}, \{r_3, r_4, r_5\}, \\ &\quad \{r_1, r_2, r_4\}, \{r_1, r_2, r_5\}, \{r_1, r_2, r_4, r_5\}, \{r_1, r_3, r_4\}, \{r_1, r_3, r_5\}, \{r_1, r_3, r_4, r_5\}\}\end{aligned}$$

Case c. Here $L_1 = r_7 \geq r_6 \geq r_5$, and LH_2 is the hierarchy in (a). Again, we apply Theorem 4.2. Computation can be carried out similarly using:

$$\begin{aligned}\bigsqcup(Lh) &= \bigsqcup(L_1) \bigcup \bigsqcup(LH_2^L) \bigcup \bigsqcup(L_1) \bigotimes \bigsqcup(LH_2^L) \\ &= \{\{r_5\}, \{r_6\}, \{r_7\}\} \bigcup \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}, \{r_4\}, \{r_1, r_4\}, \\ &\quad \{r_2, r_4\}, \{r_3, r_4\}, \{r_1, r_2, r_4\}, \{r_1, r_3, r_4\}\} \bigcup (\{\{r_5\}, \{r_6\}, \{r_7\}\} \bigotimes \{\{r_1\}, \{r_2\}, \\ &\quad \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}, \{r_4\}, \{r_1, r_4\}, \{r_2, r_4\}, \{r_3, r_4\}, \{r_1, r_2, r_4\}, \{r_1, r_3, r_4\}\}) \\ &= \{\{r_5\}, \{r_6\}, \{r_7\}, \{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}, \{r_4\}, \{r_1, r_4\}, \{r_2, r_4\}, \\ &\quad \{r_3, r_4\}, \{r_1, r_2, r_4\}, \{r_1, r_3, r_4\}, \{r_1, r_5\}, \{r_2, r_5\}, \{r_3, r_5\}, \{r_1, r_2, r_5\}, \\ &\quad \{r_1, r_3, r_5\}, \{r_4, r_5\}, \{r_1, r_4, r_5\}, \{r_2, r_4, r_5\}, \{r_3, r_4, r_5\}, \{r_1, r_2, r_4, r_5\},\end{aligned}$$

$\{r_1, r_3, r_4, r_5\}, \{r_1, r_6\}, \{r_2, r_6\}, \{r_3, r_6\}, \{r_1, r_2, r_6\}, \{r_1, r_3, r_6\}, \{r_4, r_6\}, \{r_1, r_4, r_6\}, \{r_2, r_4, r_6\}, \{r_3, r_4, r_6\}, \{r_1, r_2, r_4, r_6\}, \{r_1, r_3, r_4, r_6\}, \{r_1, r_7\}, \{r_2, r_7\}, \{r_3, r_7\}, \{r_1, r_2, r_7\}, \{r_1, r_3, r_7\}, \{r_4, r_7\}, \{r_1, r_4, r_7\}, \{r_2, r_4, r_7\}, \{r_3, r_4, r_7\}, \{r_1, r_2, r_4, r_7\}, \{r_1, r_3, r_4, r_7\}\}$

We note that each hierarchical structure can be broken down into a list of linear paths. We refer to such a decomposition of hierarchy as *linear path decomposition of hybrid hierarchy* (LPDHH). In the following, we consider a general hierarchy rooted at a role and represent it using a set of linear components in LPDHH.

Definition 4.3 (*Linear Path Decomposition of Hybrid Hierarchy —LPDHH*). Let $H = (X, [f])$ be a hierarchy over role set X rooted at role S_H with relation set $[f] \subseteq \{\geq_i, \geq_a, \geq\}$. We say that H is an ordered set of linear paths *(hybrid* or *monotype)*, that is, $H = (LH_1, LH_2, \ldots, LH_m)$, where LH_i is a linear path over X_i, if, for $i, j \in \{1, 2, \ldots, m\}$, $i \neq j$ and LH_i is a linear path over X_i, and the following conditions hold

1. $S_{LH_i} = S_H$; $J_{LH_i} \in J_H$, (note J_{LH_i} is a role, J_H is a role set)
2. $X_i \neq X_j$; $X = \bigcup_{i=1}^{m} X_i$
3. for all $J \in J_H$, there exists no linear path $LH = (\{S_H, x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_i}, J\}, [f'])$, where $[f'] \subseteq [f]$ and $\{x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_i}\} \subseteq X \backslash \{S_H, J\}$, such that $LH \notin \{LH_1, LH_2, \ldots, LH_m\}$.

We say that $LH_1, LH_2, \ldots, LH_m$ is the complete LPDHH of H. H can also be written as (LH_1, H'), (H'', LH_m), (H_x, H_y), etc., each of which is a decomposition of H.

Based on the notion of LPDHH of a general *hybrid* hierarchy, the following theorem shows how we can formally determine $\bigsqcup(H)$ of a general *hybrid* hierarchy that is not a simple linear path.

THEOREM 4.3. *Let $H = (X, [f]) = (LH_1, H_1)$ be a nonempty and nonlinear hierarchy. Then, $\bigsqcup(H) = I \backslash C$, where*

- $I = \bigsqcup(LH_1) \bigcup \bigsqcup(H_1) \bigcup \bigsqcup(LH_1) \backslash B \bigotimes \bigsqcup(H_1) \backslash B$
- $B = (\bigsqcup(LH_1) \sqcap \bigsqcup(H_1))$*, where* $A \sqcap B = \{S,\ T | S \in A, T \in B$ *and* $S \bigcap T \neq \phi\}$
- $C = \{Z | Z \in I,$ *and* $\exists\, x, y \in Z, x \geq y\}$.

The theorem determines $\bigsqcup(H)$ of a general hierarchy that has at least one role having multiple juniors, hence, making it different from the linear paths. The computation is based on the partitioning of the hierarchy into two components, in which one is a linear component and the other is the remaining part of the hierarchy. This allows us to compute $\bigsqcup(H)$ recursively once we have the linear components. The following example illustrates the working of Theorem 4.3.

Example 4.3. Consider the hierarchy in Figure 6. The linear components of the hierarchy are shown in a–d. Each component's $\bigsqcup(LH)$ computed using Theorem 4.2 is shown below, based on which we compute the UAS of the entire

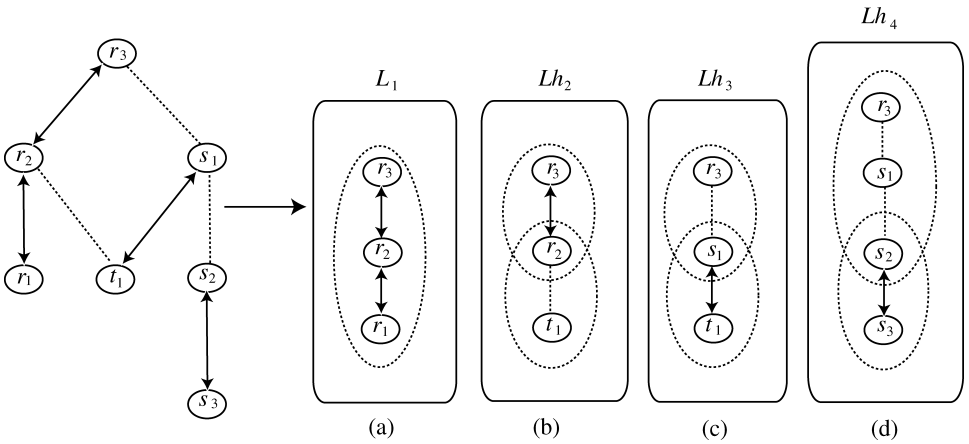

Fig. 6. Computing UAS for a general hierarchy.

hierarchy. We will write H_{12} to mean the hierarchy formed by components L_1 and Lh_2, H_{13} to mean the hierarchy formed by components L_1, Lh_2 and Lh_3, and H_{14} to mean the overall hierarchy. First, we get

$$\begin{aligned}
\bigsqcup(L_1) &= \{\{r_3\}, \{r_2\}, \{r_1\}\} \\
\bigsqcup(Lh_2) &= \{\{t_1\}, \{r_2\}, \{r_3\}, \{t_1, r_2\}, \{t_1, r_3\}\} \\
\bigsqcup(Lh_3) &= \{\{r_3\}, \{s_1\}, \{t_1\}, \{s_1, r_3\},\{r_3, t_1\}\} \\
\bigsqcup(Lh_4) &= \{\{r_3\}, \{s_1\}, \{s_2\}, \{s_3\} , \{r_3, s_1\},\{r_3, s_2\}, \{r_3, s_3\}, \{s_1, s_2\}, \\
&\quad \{s_1, s_3\}, \{r_3, s_1, s_2\}, \{r_3, s_1, s_3\}\}
\end{aligned}$$

Step 1: Consider components L_1 and Lh_2.
Here, $B = \bigsqcup(L_1) \sqcap \bigsqcup(Lh_2) = \{\{r_3\}, \{r_2\},\{t_1, r_2\}, \{t_1, r_3\}\}$.
Therefore, $\bigsqcup(L_1)\backslash B \otimes \bigsqcup(Lh_2)\backslash B = \{\{r_1\}\} \otimes \{\{t_1\}\} = \{\{r_1, t_1\}\}$.
Note that C is empty. Thus, $\bigsqcup(H_{12}) = I \setminus \mathrm{C} = I$
$= \{\{r_3\},\{r_2\}, \{r_1\}, \{t_1\}, \{r_1, t_1\}, \{r_3, t_1\},\{t_1, r_2\}\}$.

Step 2: Consider component H_{12} (result from *Step* 1) and Lh_3.
Here, $B = \bigsqcup(H_{12}) \sqcap \bigsqcup(Lh_3) = \{\{r_3\}, \{t_1\}, \{r_1, t_1\}, \{r_2, t_1\}, \{r_3, t_1\}, \{r_3, s_1\}\}$.
Therefore, $\bigsqcup(H_{12})\backslash B \otimes \bigsqcup(Lh_3)\backslash B$=$\{\{r_2\}, \{r_1\}\} \otimes \{\{s_1\}\}$
$= \{\{r_1, s_1\}, \{r_2, s_1\}\}$.
Hence, $I = \{\{r_3\}, \{r_2\}, \{r_1\}, \{t_1\}, \{r_1, t_1\}, \{r_3, t_1\}, \{t_1, r_2\}, \{s_1\}, \{r_2, s_1\}, \{r_1, s_1\}, \{r_3, s_1\}\}$
Thus, $\bigsqcup(H_{13}) = I\backslash C = \{\{r_3\}, \{r_2\}, \{r_1\}, \{t_1\}, \{r_1, t_1\}, \{t_1, r_2\}, \{s_1\}, \{r_2, s_1\}, \{r_1, s_1\}, \{r_3, s_1\}\}$ (C is empty)

Step 3: Consider component H_{13} (result from *Step 2*) and Lh_4.
Here, $B = \bigsqcup(H_{13}) \sqcap \bigsqcup(Lh_4) = \{\{r_3\}, \{s_1\}, \{r_1, s_1\}, \{r_2, s_1\}, \{r_3,s_2\}, \{r_3, s_3\}, \{s_1,s_2\}, \{r_3, s_1, s_2\}, \{r_3, s_1, s_3\}, \{r_3, s_1\}, \{s_1, s_3\}\}$.
Therefore, $\bigsqcup(H_{13})\backslash B \otimes \bigsqcup(Lh_4)\backslash B$
$= \{\{r_2\}, \{r_1\}, \{t_1\}, \{r_1, t_1\}, \{t_1, r_2\}\} \otimes (\{\{s_2, s_3\}\})$

$= \{\{r_2, s_2\}\ \{r_1, s_2\}, \{t_1, s_2\}, \{r_1, t_1, s_2\}, \{t_1, r_2, s_2\}, \{r_2, s_3\}\ \{r_1, s_3\}, \{t_1, s_3\}, \{r_1, t_1, s_3\}, \{t_1, r_2, s_3\}\}$.

Hence, $\bigsqcup(H_{14}) = I \backslash C = \{\{r_3\}, \{s_1\}, \{s_2\}, \{s_3\}, \{r_3, s_1\}, \{r_3, s_2\}, \{r_3, s_3\}, \{s_1, s_2\}, \{s_1, s_3\}, \{r_3, s_1, s_2\}, \{r_3, s_1, s_3\}\} \bigcup \{\{r_3\}, \{r_2\}, \{r_1\}, \{t_1\}, \{r_1, t_1\}, \{t_1, r_2\}, \{s_1\}, \{r_2, s_1\}, \{r_1, s_1\}, \{r_3, s_1\}\} \bigcup \{\{r_2, s_2\}\ \{r_1, s_2\}, \{t_1, s_2\}, \{r_1, t_1, s_2\}, \{t_1, r_2, s_2\}, \{r_2, s_3\}\ \{r_1, s_3\}, \{t_1, s_3\}, \{r_1, t_1, s_3\}, \{t_1, r_2, s_3\}\}$

$= \{\{r_3\}, \{s_1\}, \{s_2\}, \{s_3\}, \{r_3, s_1\}, \{r_3, s_2\}, \{r_3, s_3\}, \{s_1, s_2\}, \{s_1, s_3\}, \{r_3, s_1, s_2\}, \{r_3, s_1, s_3\}, \{r_2\}, \{r_1\}, \{t_1\}, \{r_1, t_1\}, \{t_1, r_2\}, \{r_2, s_1\}, \{r_1, s_1\}, \{r_2, s_2\}\ \{r_1, s_2\}, \{t_1, s_2\}, \{t_1, s_3\}, \{t_2, r_2\}, \{r_1, t_1, s_2\}, \{t_1, r_2, s_2\}, \{t_1, r_1, s_3\}, \{t_1, r_2, s_3\}, \{s_3, r_3\}, \{s_3, r_1\}\}$.

Based on the theorems, a recursive algorithm can be easily constructed.

4.2 Acquisition Equivalent Hierarchies

An important issue is whether or not we can use a hierarchy of one type to achieve what a hierarchy of another type allows. To address such an issue, we need an appropriate notion of equivalence between different hierarchies, as they may be structurally and semantically different. We note that, central to the use of hierarchies in a GTRBAC system is the efficient and fine-grained management of permissions acquired by users assigned to the various roles in the hierarchy. A notion of the equivalence between two hierarchies can be established if we show that the maximum set of permissions that can be acquired by a user in the two hierarchies is the same. The significance of using the maximum set of permissions is that within the equivalent hierarchies, the users can have the same set of accesses, even though, within each hierarchy, the users may have to activate a different set of roles. Here, we introduce the notion of *acquisition-equivalence* between two hierarchies. We say that two hierarchies are *acquisition-equivalent* if they allow the same maximum set of permissions to be acquired by a user assigned to the senior-most role. We use $P_{max}(H)$ to refer to the maximum set of permissions that a user can acquire through the senior-most role of the hierarchy H in a session. The notion of *acquisition-equivalence* is formally defined as follows:

Definition 4.4 (*Acquisition Equivalence or AC-Equivalence of Two Hierarchies*). Let H_1 and H_2 be two hierarchies over role set `Roles`. We then say that H_1 and H_2 are acquisition-equivalent or AC-equivalent (written as $H_1 =_{AC} H_2$), if $P_{max}(H_1) = P_{max}(H_2)$. Furthermore, $H_1 =_{AC} H_2$ and $H_2 =_{AC} H_3$, then $H_1 =_{AC} H_3$.

The following theorem provides the formal characteristics of an *AC-equivalent* set of hierarchies.

THEOREM 4.11 (AC-EQUIVALENT HIERARCHIES). *Let* $H_1 = (X, [f_1]) = (LH_1, LH_2, \ldots, LH_n)$ *and* $H_2 = (X, \langle f_2 \rangle)$ *be two hierarchies over the role set X. If, for roles* $x, y \in X$ *and a relation* $\langle f \rangle \in [f_1]$, *the condition* $(x \langle f \rangle y) \in H_1 \longleftrightarrow (x \langle f_2 \rangle y) \in H_2$) *holds, then* $H_1 =_{AC} H_2$ (*i.e.* H_1 *and* H_2 *are AC-equivalent*) *provided the following holds*

- *for all* $i \in \{1, 2, \ldots, n\}$*, and for hierarchies LH', LH_{mid}, LH", each possibly empty, the following is satisfied for* $\langle f_x \rangle = \geq_i$ *and* $\langle f_y \rangle = \geq_a$

$$\neg\exists L_x, L_y \text{ such that } LH_i = (LH', L_x, LH_{mid}, L_y, LH''),$$

The condition $LH_i = (LH', L_x, LH_{mid}, L_y, L'')$ implies that in the linear component LH_i, there is an I relation that precedes (not necessarily immediately, as LH_{mid} may not be empty) an A relation. All hierarchies that do not have such a component are *AC-equivalent* to a *monotype* hierarchy of the same structure. As a consequence, first, the theorem implies that any two *monotype* hierarchies are *AC-equivalent*, as the condition $LH_{mid} = (LH', L_x, LH_{mid}, L_y, L'')$ cannot occur in a *monotype* hierarchy. For example, consider a *monotype I* hierarchy H_1. Now construct an A hierarchy H_2 such that it contains the same roles that are in H_1, and for each I relation between a pair of roles in H_1, H_2 has an A relation. The theorem indicates that $H_1 =_{AC} H_2$. This is because all the permissions that can be used by a user assigned to the senior-most role of H_1, can also be used by a user assigned to the senior-most role of H_2. The only difference between the two is that the users may have to activate a different set of roles from the $\bigsqcup(H)$s of the two hierarchies to do that.

Furthermore, the theorem indicates that every hierarchy that does not contain a linear component shown above is *AC-equivalent* to a *monotype* hierarchy of the same structure and, hence, to each other. This is because if an I relation precedes an A *relation* in the hierarchy then the permissions associated with the roles below the A *hierarchy* cannot be acquired by any user assigned to the senior-most role, hence, reducing the permissions that can be acquired. The significance of this result is that in systems where the principle of least privilege is not of much concern, any *monotype* hierarchy can be used instead of a more complex *hybrid* hierarchy.

5. DERIVED RELATIONS IN A HIERARCHY

In a hierarchy where all the three types of hierarchies can coexist, a hierarchical relation between a pair of roles that are not directly related may be derived. From the axioms and the hierarchy definitions presented in Section 3, it is easy to see that the three hierarchy types are transitive. For instance, if $(x \geq y)$ and $(y \geq z)$ then it implies $(x \geq z)$. However, in a hybrid hierarchy, the derived relation between an arbitrary pair of roles can have partial transitivity or special hierarchical semantics. For instance, if $(x \geq_i y)$ and $(y \geq z)$ then it implies $(x \geq_i z)$ i.e., transitivity exists only with respect to the permission-inheritance semantics. Similarly, assume $(x \geq_a y)$ and $(y \geq_i z)$. Here it appears that x and z are not hierarchically related because (1) if a user u assigned to x activates x, he does not acquire z's permissions, and (2) u cannot activate z to acquire its permissions. Note, however, that u can activate y and acquire all the permissions that can be acquired through z. We call this special derived type a *conditioned derived relation*, written as $(x[A](B)\langle f \rangle y)$, and, defined as follows:

Definition 5.1 (*Conditioned-Derived Relation*). Let H be a role hierarchy, x, y $\in$ `Roles (H)` and $A, B \subseteq$ `Roles (H)`. Then $(x[A](B)\langle f \rangle y)$ is called a *conditioned Derived Relation (*also read as the derived relation $(x\langle f \rangle y)$ is conditioned on

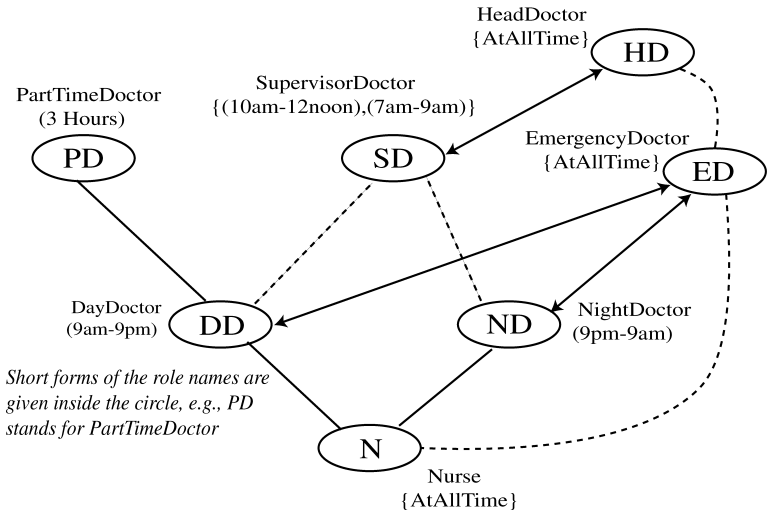

Fig. 7. A hierarchy for a medical department.

roles in A and B″*)*, if, for all $a \in A$ and $b \in B$, the following holds:

$$(x \geq_a a) \wedge (a \langle f \rangle y) \wedge ((x = b) \vee (x \geq_a b)) \wedge (b \geq_a y)$$

where $\langle f \rangle \in \{\geq_i, \geq\}$, $|A| > 0$, $|B| \geq 0$, *and* $(b \geq_a y)$ is a direct relation.

Here, the condition indicates that x is related to each $a \in A$ directly or through a derived A relation, whereas each a is related to y by the $\langle f \rangle$ relation. This implies that a permission that can be acquired through role y can be acquired by a user assigned to role x without activating y, but by activating any of the roles in A. We note that B may be empty, in which case, the *conditioned derived relation* is simply written as $(x[A]\langle f \rangle y)$. If B is not empty, then for each $b \in B$, there is an A path from x to y through b. If $C = A \cap B$ then, for all $c \in C$, both $(c \geq_i y)$ and $(c \geq_a y)$ hold and, hence, $(c \geq y)$. It is possible that $(x[A](\{x\})\langle f \rangle y)$, which means $(x[A]\langle f \rangle y)$ holds and $(x \geq_a y)$ is a direct relation. As we shall see, it is not necessary that each hierarchical path from x to each $a \in A$ contain only A relations; it is only required that a user who is assigned to, or can activate, x can also activate a. This, however, implies that derived/direct relation between x and a is not an I-relation. Furthermore, we note that in $(x[A](B)\langle f \rangle y)$, $\langle f \rangle$ is either $\geq_i$ or $\geq$.

Example 5.1. Consider the hierarchy of Figure 7, representing a medical department. PD can be enabled for 3 hr only. SD's relation to DD and ND are as discussed in Figure 7. N can be I-inherited by DD and ND. ED is enabled at all times. The A relation between ED and N allows a user assigned to ED to explicitly act as a nurse besides inheriting N's permissions through DD or ND. Assume that the HD role represents the head doctor of the medical department, which is enabled at all times. HD can act as the supervisor role of doctors. Two of the *conditioned-derived relations* are as follows.

1. (SD[{DD, ND}] $\geq_i$ N): This is because users assigned to SD can acquire permissions of N only by activating DD or ND.
2. (HD[{DD, ND, ED}]({ED}) $\geq_i$ N): This is because users assigned to HD can acquire permissions of N by activating SD, ND, or ED. Furthermore, the users can directly activate N (because of the A path through ED).

5.1 The Inference Rules

We now introduce the inference rules that allow derivation of indirect relations between roles from a set of explicitly specified hierarchical relations. Such derived relations can be used to determine the permissions that can be acquired through the activation of a role in a hierarchy by a user. We use $\mathbf{ISen}(y) = \{x | (x \geq_a y)$ *is a direct relation*$\}$ to denote the set of the immediate A seniors of role y. The inference rules are as follows.

Definition 5.2 (*Inference Rules*). Let H be a role hierarchy, $x, y, z \in$ Roles(H), and A, A_1, A_2, B_1, $B_2 \subseteq$ Roles(H). Then the inference rules for deriving indirect relations are as shown in Table V.

R_1 is a trivial case of transitivity using a single hierarchy type. Thus, if $\langle f \rangle$ is $\geq_a$, then from the two relations $(x \geq_a y)$ and $(y \geq_a z)$, relation $(x \geq_a z)$ is inferred. R_2 applies to all the pairs with direct or derived relations. This can result in a *conditioned*-derived relation of the form $(x[A]\langle f \rangle z)$. R_3 deals with each of the cases in which an *unconditioned* relation follows a *conditioned*-derived relation.

[AQ3]

In a hierarchy, there may be more than one hierarchical path between roles that are not directly related. Such a situation arises when there are multiple hierarchical paths between a given pair of roles. R_4 deals with such cases. Rule $R_{4.1}$ is a trivial case in which both the hierarchical paths are the same *unconditioned* relation (derived or direct). Rule $R_{4.2}$ captures all the possible combinations of two different *unconditioned* relations between the same pair. Similarly, rule $R_{4.3}$ captures all the possible combinations of two different hierarchical relations between the same pair of roles in which one is an *unconditioned*-derived relation. Last, $R_{4.4}$ captures all the possible combinations of two different hierarchical *conditioned*-derived relations between a pair of roles. Table VI illustrates the application of these rules to determine the derived relations for the hierarchy in Figure 7.

5.2 Soundness and Completeness of the Inference Rules

In this section, we show that the set of inference rules introduced above is *sound* and *complete*, using the notion of *authorization consistent hierarchies*, which is defined below. In the definition, we use predicate *can_activate*(*u, r, H*) to mean that u can activate role r in role hierarchy H. Similarly, we use predicate *can_be_acquired*(*p, r, H*) to mean that permission p can be acquired through role r using permission-inheritance semantics in hierarchy H. Let UAH(H) and PAH(H) be sets of all the user-role and role-permission assignments related to the roles in Roles(H).

Table V. The Inference Rules

Rule	Case		Inference rule
R_1	(Monotype hierarchy)		
	$(x\langle f\rangle y)\wedge(y\langle f\rangle z)\rightarrow(x\langle f\rangle z)$ *for all* $\langle f\rangle\in\{\geq_i,\geq_a,\geq\}$		
R_2	(Hybrid hierarchy with unconditioned relations)		
	1		$(x\langle f_1\rangle y)\wedge(y\langle f_2\rangle z)\rightarrow(x\geq_i z)$ *for all* $\langle f_1\rangle,\langle f_2\rangle\in\{\geq_i,\geq\}$ *such that* $\langle f_1\rangle\neq\langle f_2\rangle$
	2		$(x\geq y)\wedge(y\geq_a z)\rightarrow(x\geq_a z)$
	3		$(x\geq_a y)\wedge(x\langle f\rangle y)\rightarrow(x[\{y\}]\langle f\rangle z)$ for $\langle f\rangle\in\{\geq_i,\geq\}$
R_3	(Hybrid hierarchy with one unconditioned-derived relation)		
	1		*for* $\langle f\rangle\in\{\geq_i,\geq_a,\geq\}$ *such that* $\langle f_1\rangle\neq\langle f_2\rangle$
		a.	$(x[A](B)\geq_i y)\wedge(y\geq_i z)\rightarrow(x[A\cup C]\geq_i z)$, where $C=\{y\}$ *if* $\lvert B\rvert>0$, *else* $C=\phi$
		b.	$(x[A](B)\geq_i y)\wedge(y\geq z)\rightarrow(x[A\cup C](C)\geq_i z)$, where $C=\{y\}$ *if* $\lvert B\rvert>0$, *else* $C=\phi$
	2		*for* $\langle f\rangle\in\{\geq_i,\geq_a,\geq\}$ *such that* $\langle f_1\rangle\neq\langle f_2\rangle$
		a.	$(x[A](B)\geq y)\wedge(y\geq_i z)\rightarrow(x[A\cup C]\geq_i z)$, where $C=\{y\}$ *if* $\lvert B\rvert>0$, *else* $C=\phi$
		b.	$(x[A](B)\geq y)\wedge(y\geq z)\rightarrow(x[A\cup C](C)\geq z)$ where $C=\{y\}$ *if* $\lvert B\rvert>0$, *else* $C=\phi$
	3		$(x[A](B)\geq y)\wedge(y\geq_a z)\rightarrow(x\geq_a z)$
R_4	(Hierarchy with multiple paths between two roles; subscripts indicate the path number)		
	1		$(x\langle f\rangle y)_1\wedge(x\langle f\rangle y)_2\rightarrow(x\langle f\rangle y)$ *for all* $\langle f\rangle\in\{\geq_i,\geq_a,\geq\}$
	2		$(x\langle f_1\rangle y)_1\wedge(x\langle f_2\rangle y)_2\rightarrow(x\geq y)$ *for all* $\langle f_1\rangle,\langle f_2\rangle\in\{\geq_i,\geq_a,\geq\}$ *such that* $\langle f_1\rangle\neq\langle f_2\rangle$
	3		*for all* $\langle f\rangle,\langle f_1\rangle,\langle f_2\rangle\in\{\geq_i,\geq\}$ *such that* $\langle f_1\rangle\neq\langle f_2\rangle$
		a.	$(x[A](B)\geq_i y)_1\wedge(x\geq_i y)_2\rightarrow(x\geq_i y)$ *if* $B=\phi$ $(x[A](B)\geq_i y)_1\wedge(x\geq_i y)_2\rightarrow(x\geq y)$ *if* $\lvert B\rvert>0$ $(x[A](B)\geq y)_1\wedge(x\geq y)_2\rightarrow(x\geq y)$
		b.	$(x[A](B)\langle f\rangle y)_1\wedge(x\geq_a y)_2\rightarrow(x[A](\mathbf{ISen}(y))\langle f\rangle y)$
		c.	$(x[A](B)\langle f_1\rangle y)_1\wedge(x\langle f_2\rangle y)_2\rightarrow(x\geq y)$
	4		*for all* $\langle f\rangle,\langle f_1\rangle,\langle f_2\in\{\geq_i,\geq\}$ *such that* $\langle f_1\rangle\neq\langle f_2\rangle$
		a.	$(x[A_1](B_1)\langle f\rangle y)_1\wedge(x[A_2](B_2)\langle f\rangle y)_2\rightarrow(x[A_1\cup A_2](B_1\cup B_2)\langle f\rangle y)$
		b.	$(x[A_1](B_1)\langle f_1\rangle y)_1\wedge(x[A_2](B_2)\langle f_2\rangle y)_2\rightarrow(x[A_1\cup A_2](A\cup B_1\cup B_2)\geq_i y)$ *such that* $A=A_1$ *if* $\langle f_1\rangle=\geq$ *else* $A=A_2$

Table VI. Application of Inference Rules over the Hierarchy of Figure 7

Rule Applied		Derived Relations
R_1		(PD $\geq_i$ N), (HD $\geq_a$ N)
R_2	1	(ED $\geq$ ND) $\wedge$ (ND $\geq_i$ N) implies (ED $\geq_i$ N)
	2	(HD $\geq$ SD) $\wedge$ (SD $\geq_a$ DD) implies (HD $\geq_a$ DD)
	3	(SD $\geq_a$ DD) $\wedge$ (DD $\geq_i$ N) implies (SD[{DD}] $\geq_i$ N)
		(HD $\geq_a$ ED) $\wedge$ (ED $\geq$ ND) implies (HD[{ED}] $\geq$ ND)
		(HD $\geq_a$ DD) $\wedge$ (DD $\geq$ N) implies (HD[{DD}] $\geq$ ND)
R_3	2a	(HD[{ED}] $\geq$ DD) $\wedge$ (DD $\geq_i$ N) implies (HD[{ED}] $\geq_i$ N)
		(HD[{ED}] $\geq$ DD) $\wedge$ (ND $\geq_i$ N) implies (HD[{ED}] $\geq_i$ N)
R_4	1	(ED $\geq_i$ N) (one through DD, another through ND)
	2	(ED $\geq_i$ N)$\wedge$(ED $\geq_a$ N) implies (ED $\geq$ N)
	3b	(HD[{ED}] $\geq_i$ N) $\wedge$ (ED $\geq_a$ N) implies HD[{ED}]({ED}) $\geq_i$ N) which is the same as (HD[{ED}] $\geq$ N)
	4a	(HD[{DD}] $\geq_i$ N) $\wedge$ (HD[{ND}] $\geq_i$ N) implies HD[{DD, ND}] $\geq_i$ N)
	4b	(HD[{DD,ND}] $\geq_i$ N) $\wedge$ (HD[{ED}] $\geq$ N) implies HD[{DD, ND, ED}] $\geq_i$ N)

Definition 5.3 (*Authorization-Consistent Hierarchies*). Let H_1 and H_2 be two hierarchies such that Roles(H_1) = Roles(H_2), *UAH*(H_1) = *UAH*(H_2) and *PAH*(H_1) = *PAH*(H_2). Then, we say that H_1 and H_2 are authorization consistent (written as $H_1 \approx H_2$) if, for all r $\in$ Roles (H_1), the following conditions hold:

1. $\forall u \in$ Users, can_activate *(u, r, H_1)* $\longleftrightarrow$ can_activate *(*u, r, H_2*)*,
2. $\forall p \in$ Permissions, can_be_acquired *(p, r, H_1)* $\longleftrightarrow$ can_be_acquired *(p, r, H_2)*.

Here, we note that the two hierarchies considered have the same set of roles, user-role assignments and role-permission assignments. Condition (1) implies that if a user u can activate a role r in Roles(H_1), then he can activate it even if H_1 is replaced by H_2 (and vice versa). Similarly, the second condition says that the set of permissions that can be acquired through a role under H_1 is also the same set of permissions that can be acquired through that role in H_2 for any given user. This signifies that if two hierarchies are *authorization consistent* then a user assigned to a role can activate exactly the same set of roles and acquire the same set of permissions under the two hierarchies. This means the permission-inheritance and role-activation semantics in the two hierarchies are the same, even if the sets of hierarchical relations in the two hierarchies are different. Figure 8 depicts an example of the notion of *authorization consistency*. Here, the hierarchy relation h_1 in H_2 can be inferred from the hierarchical relations ($r_1 \geq_i r_3$) and ($r_3 \geq r_5$), whereas, h_2 can be inferred from the two hierarchical paths from role r_1 to r_4. Hence, H_2 adds no new access capability compared to H_1. However, h_3 in H_3 cannot be inferred from the hierarchical relations ($r_1 \geq_i r_3$) and ($r_3 \geq r_5$). In H_3, a user assigned to r_1 can also activate r_5, which is not possible in H_1 or H_2. Hence, ($H_1 \not\approx H_3$), and ($H_2 \not\approx H_3$). We use this notion of authorization consistency between two hierarchies to show that the set of rules presented above is *sound*, i.e., each new derived relation that can be deduced from a given set of hierarchical relations using the rules

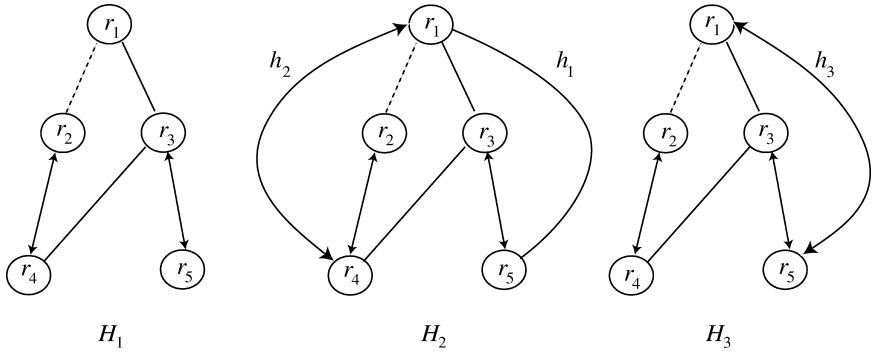

Fig. 8. Example of AC-equivalence; $H_1 \approx H_2, H_1 \nsim H_3$ *and* $H_2 \nsim H_3$.

produces the same inheritance and activation semantics that is already present in the original hierarchy. Within a hierarchy H, we use h_{xz} to represent $(x\langle f\rangle z)$ for $\langle f\rangle \in \{\geq_i, \geq_a, \geq\}$ or $(x[A](B)\langle f\rangle z)$ for $\langle f\rangle \in \{\geq_i, \geq_a\}$, where $x, z \in \texttt{Roles}(H)$ and $A,B \subseteq \texttt{Roles}(H)$. The following theorem formally states this result.

THEOREM 5.5 (SOUNDNESS OF RULES R_1–R_4). *Given a role hierarchy H, if a new hierarchical relation h_{xz} is derived from hierarchical relations in H as per rules R_1–R_4, and $H' = H \cup \{h_{xz}\}$, then H and H′ are authorization consistent, i.e., $H \approx H'$.*

The theorem implies that the new relations derived using the rules do not allow a role to inherit more (or less) permissions than was allowed to it before the derived relation is added. Similarly, the new derived relation does not allow a user to be able to activate more (or less) number of roles than that was allowed before the derived relation is introduced. Next, we present the *completeness* theorem for the rules R_1–R_4. We write $H[R_1\text{–}R_4] \vDash h_{x,z}$ to indicate that the relations in H can logically derive relation $h_{x,z}$ using rules R_1–R_4.

THEOREM 5.2 (COMPLETENESS OF RULES R_1–R_4). *Given a role hierarchy H, rules R_1–R_4 are complete; That is, if $\neg H[R_1\text{–}R_4] \vDash h_{x,z}$, for any pair of roles $x, z \in \texttt{Roles}(H)$, then $H \nsim H \cup \{h_{x,z}\}$,* i.e.*, the hierarchies H and $H' = H \cup \{h_{x,z}\}$ are not authorization consistent.*

The theorem indicates that if a relation, say $\langle f\rangle$, between any two roles, say x and z, of $\texttt{Roles}(H)$ cannot be derived from the hierarchical relations in H, then any role hierarchy containing such a relation is not *authorization consistent* with H. In other words, we can take every pair of roles (x, z) of $\texttt{Roles}(H)$ and every possible hierarchical relations between them, including *conditioned* derived relations and extend H by adding it to get H'. If we get $H \approx H''$, the theorem implies that the rules R_1–R_4 will derive it. Hence, this shows that the rules are *complete*. Using the transitivity of the hierarchical relations and considering all the cases of the rules, we can easily construct the proofs. The proofs for both the theorems are provided in Appendix B.

Table VII. Criteria for Hierarchy Transformations

Criteria		
1	C1	$\forall u \in Users, r \in \texttt{Roles}(H_o)$ $can\_activate\ (u,r,t,H_o) \leftrightarrow can\_activate(u,r,t,H_n)$
2	C2	$\forall p \in P(\texttt{Roles}(H_o)), r \in \texttt{Roles}(H_o)$, $can\_be\_acquired\ (p,r,t,H_o) \leftrightarrow can\_be\_acquired\ (p,r,t,H_n)$
	$C2^r$	C2 is not Satisfied and $(s \geq_i r \geq_a j) \notin H_n$

6. HIERARCHY-TRANSFORMATION ALGORITHMS

In an organization, policies evolve with time, affecting the existing role hierarchies. New roles may need to be added and old ones may need to be deleted or modified. Permission sets of existing roles or their temporal properties may need to be altered. Making such changes may require restructuring the hierarchies to avoid undesirable situations. In this section, we analyze transformations of a role hierarchy when a role is added, modified, or deleted that best maintain the permission-inheritance and role-activation semantics of the original hierarchy.

6.1 Role Addition

Typically, a new role is added to an existing hierarchy to distribute a set of new permissions among the already existing roles in the hierarchy. Before we add a new role to a hierarchy, we need to properly identify the existing sets of roles that can be its seniors and juniors based on the permission-distribution requirements. Furthermore, we need to consider the existing constraints on and/or among roles in the hierarchy to determine possible new relations between the existing roles and the new role. While preexisting hierarchical semantics may need to be maintained, the permission-acquisition and role-activation semantics of the original hierarchy may need to be relaxed to allow some desirable changes.

Let r be the new role to be added in the original hierarchy H_o. Suppose r is to be added between roles s and j, and $(s\langle f \rangle j) \in H_o$. By adding the new role, assume we obtain the new hierarchy H_n. That is, $H_n = (H_o \cup \{(s\langle f_1 \rangle r), (r\langle f_2 \rangle j)\}) \backslash (s\langle f \rangle j)$ for some hierarchy relations $\langle f_1 \rangle$ and $\langle f_2 \rangle$.

In general, when a new role is added, we require that the original permission-acquisition and/or role-activation semantics of the hierarchy is maintained. These requirements can be represented as the conditional criteria, shown in Table VII, that should be valid after the transformation has been made.

When a new role r is added between two hierarchically related roles s and j, a crucial issue is the effect it has on the original relationship between s and j. Based on the hierarchical relations introduced between s and r and between r and j, there could be various derived relations between s and j. We introduce three criteria that capture all the possible changes in the semantics of the relationship between s and j in the new hierarchy H_n compared to the relation they had in the original hierarchy H_o, as shown in Table IX (see later). Here, criterion C1 indicates that the *activation-inheritance* semantics is maintained between s and j after adding the new role r. In other words, C1 is said to be

[AQ4]

Table VIII. Scenarios for Hierarchy Transformations

Scenarios for role addition (r is newly added role)	
S1	No extra constraint is added with respect to the new role r;
S2	A *permission-centric* activation constraint is added for the new role r
S3	A *user-centric* activation constraint is added for the new role r;
S4	(s, r) is considered to be in DSoD
S5	(r, j) is considered to be in DSoD

satisfied if there is *activation-inheritance* semantics applied between s and j in both H_o and H_n (e.g., row a1 in Table IX), or in neither H_o nor H_n (e.g., row b4 in Table IX). If only one of H_o and H_n, has the *activation-inheritance* semantics between s and j (e.g., row a4 and b1 in Table IX), then C1 is not satisfied. Similarly, C2 guarantees that the *permission-inheritance* semantics is maintained between s and j after adding the new role. That is, C2 is satisfied if there is *permission-inheritance* semantics between s and j in both H_o and H_n (e.g., row b4 in Table IX), or in neither H_o nor H_n (e.g., row a1 in Table IX). If, however, only one of H_o and H_n has the *permission-inheritance* semantics between the two roles s and j (e.g., row a4 and b1 in Table IX), then C2 is not satisfied. If C2 is not satisfied, it doesn't necessarily mean that the *permission-inheritance* semantics is completely lost. For instance, a *conditioned-derived relation* could exist between s and j after the new role has been added. Criterion C2^r captures such cases. Note that because one of the three hierarchy relations exist between s and j in H_o, a user assigned to s can always acquire the permissions of j in H_o — through the use of either the inheritance or the activation semantics. One special case is when in H_n, we have $s \geq_i r$ and $r \geq_a j$ (e.g., row a5, b5, and c5 in Table IX), where the original relation between s and j will be completely lost.

When a new role is added, various new constraints related to the new role may need to be added as well. We consider the five scenarios (S1 through S5), shown in Table VIII to capture such new constraints associated with r. Whether or not we can allow these constraints to be specified on the new role r depends on the hierarchical relation that r has with s and/or j in H_n, as shown in the right part of Table IX. Here, $\surd$ means the corresponding constraint can be introduced for r and $\times$ means that the corresponding constraint cannot be supported for r. S1 indicates a scenario where no extra constraint is added. If we want to add a permission-centric activation constraint to r (S2), we require that $s \geq_a r \in H_n$ [Joshi et al. 2002]; if we want to add a user-centric activation constraint to r (S3), we require that $s \geq_i r \in H_n$ or $s \geq r \in H_n$ [Joshi et al. 2002]. Note that DSoD constraints can be defined among roles related by A hierarchy [Joshi et al. 2002]. Thus, S4 is applicable if $s \geq_a r \in H_n$ and S5 is supported if $r \geq_a j \in H_n$.

6.2 Role Deletion

When a role is deleted from a hierarchy, the crucial issue is what to do with the permissions associated with it and the users assigned to it. Generally, it will be required that the permissions be retained in the system and make them available through other roles in the hierarchy. This requires redistributing the

Table IX. Transformation with Criteria Satisfied for Different Scenarios

$(s\langle f\rangle j) \in H_o$			$(s\langle f_1\rangle r), (r\langle f_2\rangle j) \in H_n$	Criteria Satisfied	S1	S2	S3	S4	S5
a	$s \geq_a j$	1	$(s \geq_a r), (r \geq_a j)$	C1, C2	√	√	×	√	√
		2	$(s \geq_a r), (r \geq_i j)$	C2	√	√	×	√	×
		3	$(s \geq_a r), (r \geq j)$	C1, C2	√	√	×	√	×
		4	$(s \geq_i r), (r \geq_i j)$	$C2^r$	√	×	√	×	√
		5	$(s \geq_i r), (r \geq_a j)$	C2	√	×	√	×	√
		6	$(s \geq_a r), (r \geq j)$	$C2^r$	√	×	√	×	×
		7	$(s \geq r), (r \geq_i j)$	$C2^r$	√	×	√	×	×
		8	$(s \geq r), (r \geq_a j)$	C1, C2	√	×	√	×	√
		9	$(s \geq r), (r \geq j)$	C1, $C2^r$	√	×	√	×	×
b	$s \geq_i j$	1	$(s \geq_a r), (r \geq_a j)$	$C2^r$	√	√	×	√	√
		2	$(s \geq_a r), (r \geq_i j)$	C1, $C2^r$	√	√	×	√	×
		3	$(s \geq_a r), (r \geq j)$	$C2^r$	√	√	×	√	×
		4	$(s \geq_i r), (r \geq_i j)$	C1, C2	√	×	√	×	×
		5	$(s \geq_i r), (r \geq_a j)$	C1	√	×	√	×	√
		6	$(s \geq_a r), (r \geq j)$	C1, C2	√	×	√	×	×
		7	$(s \geq r), (r \geq_i j)$	C1, C2	√	×	√	×	×
		8	$(s \geq r), (r \geq_a j)$	$C2^r$	√	×	√	×	√
		9	$(s \geq r), (r \geq j)$	C2	√	×	√	×	×
c	$s \geq j$	1	$(s \geq_a r), (r \geq_a j)$	C1, $C2^r$	√	√	×	√	√
		2	$(s \geq_a r), (r \geq_i j)$	$C2^r$	√	√	×	√	×
		3	$(s \geq_a r), (r \geq j)$	C1, $C2^r$	√	√	×	√	×
		4	$(s \geq_i r), (r \geq_i j)$	C2	√	×	√	×	×
		5	$(s \geq_i r), (r \geq_a j)$	None	√	×	√	×	√
		6	$(s \geq_a r), (r \geq j)$	C2	√	×	√	×	×
		7	$(s \geq r), (r \geq_i j)$	C2	√	×	√	×	×
		8	$(s \geq r), (r \geq_a j)$	C1, $C2^r$	√	×	√	×	√
		9	$(s \geq r), (r \geq j)$	C1, $C2^r$	√	×	√	×	√

permissions associated with the deleted role to other roles in the hierarchy and reassigning the users originally assigned to the deleted role. We identify the following three approaches for the deletion of a role from a hierarchy with respect to the privilege distribution: (1) the *first approach* is to reassign the permissions of the deleted role to its immediate seniors; (2) the *second approach* is to reassign the permissions of the deleted roles to its immediate juniors; and (3) the *third approach* is to reassign the permissions of the deleted role to each of the senior roles through which the permissions of the deleted role can be acquired within the original hierarchy.

One key problem with these approaches is the reassignment of the users who were originally assigned to the deleted role. As users assigned to the deleted roles need to be reassigned to junior roles or the senior roles, any reassignment will result in either a *privilege escalation* or *privilege depletion* of some users

Table X. Deletion of a Role Using the First and Second Approach

	P_r Is Assigned to Role s		P_r is Assigned to Role j	
Reassignment	U_r Assigned to s	U_r Assigned to j	U_r Assigned to s	U_r Assigned to j
Results	Privilege escalation for users in U_r	Privilege depletion for users in U_r	Privilege escalation for users in U_r and U_j	Privilege escalation for users in U_j

Table XI. Result of Deletion of a Role Satisfying C1 and C2 Using the First and Second Approach

For $(r\langle f\rangle j) \in H_o$, $\langle f\rangle is$	For $(s\langle h\rangle r) \in H_o$, $\langle h\rangle$ is	For $s\langle f\rangle j \in H_n$, $\langle f\rangle$ is (For appropriate transformation)	
		First approach	Second approach
A hierarchy ($\geq_a$)	I hierarchy ($\geq_i$)	none appropriate	none appropriate
	A hierarchy ($\geq_a$)	A hierarchy ($\geq_a$)	A hierarchy ($\geq_a$)
	IA hierarchy ($\geq$)	A hierarchy ($\geq_a$)	A hierarchy ($\geq_a$)(restrictive)
I hierarchy ($\geq_i$)	any	I hierarchy ($\geq_i$)	I hierarchy ($\geq_i$)
IA hierarchy ($\geq$)	for any $\langle h\rangle$	$\langle h\rangle$	$\langle h\rangle$

assigned to the roles in the hierarchy. The *third approach* is ad-hoc in nature and inefficient as permissions are explicitly assigned to all senior roles through which they could be acquired before the transformation. Hence, it defeats the purpose of a hierarchy structure. In practice, this approach may be applicable when the whole hierarchy needs to be restructured. We do not discuss the *third approach* further.

As before, let H_o be the original hierarchy and H_n the new hierarchy obtained by deleting role r. Furthermore, let U_r and P_r be the sets of users and permissions explicitly assigned to role r. For each immediate junior j of r, let U_j be the set of users assigned to j. Let s be an immediate senior of r. Table X depicts what permissions users assigned to r in H_o can now acquire in H_n.

As shown in the table, for both approaches, it is possible that the users are reassigned to senior or the junior roles. *Privilege escalation* of users in U_r occurs in the *first approach* if they are reassigned to senior roles; *privilege depletion* occurs if the users in U_r are reassigned to the junior roles. In practice, a choice can be made based on the risk factor related to the privilege escalation and privilege depletion resulting from reassignment of U_r. Note that U_s and U_j are not affected in this case. In the *second approach*, if the users in U_r are reassigned to s, privilege escalation, similar to that in the *first approach*, occurs with respect to U_r. In the *second approach*, privilege escalation also occurs with respect to the users in U_j. Table XI further depicts different cases of transformations for the first and the second approaches that attempt to meet the criteria C1 and C2 introduced earlier (Note here the conditions are checked for all $r \in H_n$ not for $r \in H_o$ as indicated in the Table VII).

Figure 9 depicts various transformations when $(r \geq_a j) \in H_o$ under the *first approach*. Note that s may be related to its immediate senior by any of the three hierarchical relations. To show the overall picture, we include roles x, y, and z as seniors of s with respect to I, A, and IA relations, respectively. Let $\langle h\rangle$ be the

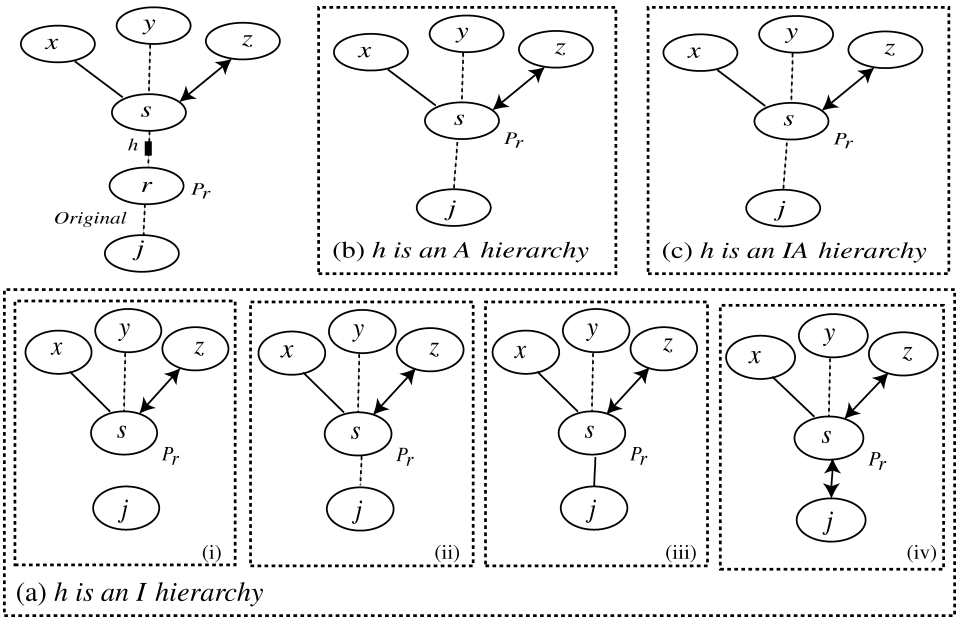

Fig. 9. Deletion of role r when $(r \geq_a j)$.

original relation between s and r. When $\langle h \rangle$ is an I hierarchy, s and j are not hierarchically related, as s does not inherit j's permissions, neither is any user assigned to s or its seniors able to activate j in H_o. Hence, case (i) in Figure 9a (i.e., "*no relation*" between s and j) retains the original derived relation between s and j (as indicated in the table). The choices (ii), (iii), and (iv) in Figure 9a result in undesirable situations as each one makes something possible that was not originally possible. Similarly, when $\langle h \rangle$ is an A or IA hierarchy, s and j have a derived relation $(s \geq_a j)$. Hence, as shown in Figures 9b and c, after the deletion of role r, we can introduce the direct relation $s \geq_a j$. We note that after the deletion of role r, if we have $(s \geq j)$, it makes the inheritance of j's permissions by s possible, which was not originally allowed.

The cases for $(r \geq_i j)$ or $(r \geq j)$ in H_o can be similarly explained. When $(r \geq_i j) \in H_o$, for all relations between s and r, the resulting relation between s and j will be $(s \geq_i j)$ as shown in the table. It is straightforward to see that it is so when $\langle h \rangle$ is an I-relation. If $\langle h \rangle$ is an IA-relation, then $(s \geq_i j)$ is the derived relation in H_o and, hence, after the transformation, the relation is maintained. However, if $\langle h \rangle$ is an A relation, then the original relation between s and j would be $(s[\{r\}] \geq_i j)$. If in the transformed hierarchy, we use relation $(s \geq_i j)$ then users who can activate s cannot activate j, but still can acquire j's permission by activating s in place of the deleted role r. Hence, the semantics about a user *not being able to activate it but being able to acquire its permissions by activating some senior role* is still present in the hierarchy with the new relation $(s \geq_i j)$. It is, however, to be noted that this transformation affects the original relations between j and role s or those above it. The change is in terms of what needs to be activated to acquire j's permissions.

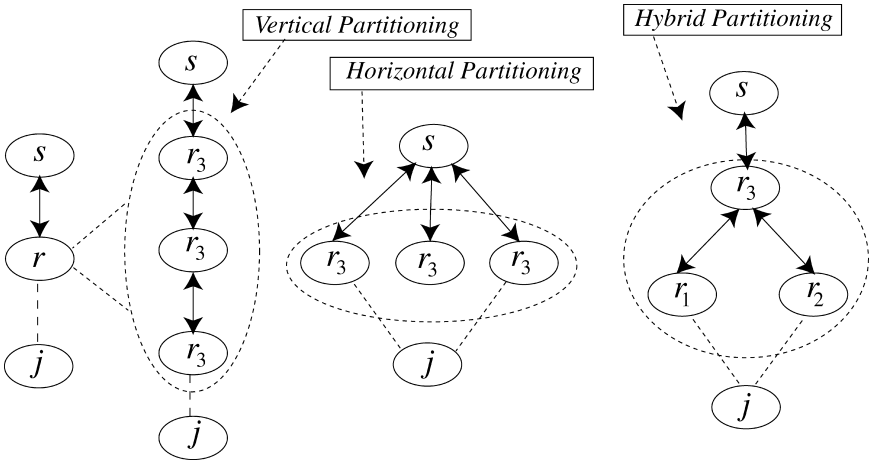

Fig. 10. Partitioning a role r into three roles r_1, r_2 and r_3.

Various cases for the *second approach* can be similarly explained. The key difference is when $(r \geq_a j) \in H_o$. Here, if $(s \geq_i r) \in H_o$, no hierarchical relation between s and j can be derived in H_o; hence, we cannot have any relation between s and j. However, *no relation* between s and j means that the permission set P_r, now assigned to j, cannot be used by any user who can activate s. Note that in H_o, a user who is assigned to s can inherit permissions of r and hence acquire P_r. Therefore, for this case, there is no appropriate transformation. Similarly, **[AQ5]** if $(s \geq r) \in H_o$, the only possible new relation is $(s \geq_a j)$. However, it is somewhat *restrictive* in the sense that, in H_o, a user u who can activate s need not explicitly activate r to acquire its permissions, but in H_n, u needs to activate j to acquire its permissions.

6.3 Partitioning of an Existing Role

Sometimes, it is essential that an existing role be simply partitioned to change the semantics of the hierarchy. In particular, partitioning may indicate the requirement for separating the role's permissions into different subsets. We identify the following three ways to partition a role: (1) *vertical partitioning:* here a role is partitioned into a set of new roles that form a linear path with the permission set of the old role distributed among the new roles; (2) *horizontal partitioning:* here the role's permission set is partitioned into a number of disjoint sets, each of which is assigned to a new role; the new roles do not have any hierarchical relations between them; and (3) *hybrid partitioning:* here both vertical and horizontal partitioning are applied on the role, which result in an arbitrary hierarchy over the new roles. Figure 10 illustrates these partitions.

In each case, the set of new roles replaces the partitioned role in the hierarchy. Once a role is partitioned, it is possible that an administrator completely redefines the hierarchical relationships in the part of the hierarchy above the partitioned role. Such a case requires offline redesign of the system. However, it may be necessary to retain the original hierarchical semantics as defined

by criteria C1 and C2 (Table VII). Table XII lays out various transformation characteristics of the three approaches with emphasis in retaining the original derived relation between s and j. In particular, Table XII depicts cases where *vertical partitioning* creates a monotype linear path and *hybrid partitioning* creates multiple monotype linear paths. We discuss hybrid linear paths resulting from *vertical* and *hybrid partitioning* at the end of this section. Here, role r of the original hierarchy H_o is partitioned into a set of new roles RP = $\{x_1, x_2, \ldots, x_n\}$. As usual, let s and j represent a senior and a junior of r.

Row 1 shows various hierarchy characteristics associated with the roles in RP. As already indicated above, in *vertical partitioning*, the new roles form a linear path. As shown in the table, if originally $(s \geq_i r), (r \langle f \rangle j) \in H_o$ where $\langle f \rangle \in \{\geq_i, \geq\}$, or $(s \geq r), (r \geq_i j) \in H_o$, then in the new hierarchy H_n, the monotype hierarchy over the roles in RP should be of type $\geq_i$ or $\geq$. This is necessary to retain the original derived relation $(s \geq_i j)$ in the transformed hierarchy. If $(s \geq r), (r \geq j) \in H_o$ or $(s \geq r), (r \geq_a j) \in H_o$, then the new linear path over the roles in RP should be of type $\geq$. Similarly, if $(s \geq_a r), (r \langle f \rangle j) \in H_o$, then the new linear path over the roles in RP should be of type $\geq_a$ or $\geq$.

The original semantics as defined by criteria C1 and C2 are ensured in the vertical partitioning by these transformations and by the new relations defined in rows 3 and 4. For *horizontal partitioning*, the roles in RP are not hierarchically related. For *hybrid partitioning*, the roles in RP form multiple linear paths. The condition for the *hybrid partitioning* states that at least one linear path must allow inferring the derived relation $(s \langle f \rangle j)$ of H_n. For the linear path that maintains the original derived relation $(s \langle f \rangle j)$, we can use the transformations outlined for *vertical partitioning* in the *if–then* columns.

Entries in row 2 indicate the reassignments of the users in U_r originally assigned to role r, to new role(s) in RP. The reassignments shown here are defined on the basis that the original access capabilities of the users are to be retained, although they may result in privilege escalation for some users. In practice, this may not be the actual case and the relations among roles in the partition shown in row 1 may need to be accordingly adjusted. Rows 3 and 4 indicate how the roles s and j are related to the new roles in the partition. For a *vertical partitioning* approach, the original relation between s and j is used between s and x_1, and x_n and j, as indicated. Note that x_1 and x_n are the senior- and the junior-most roles of the new linear path created by the roles in RP. In case of *horizontal partitioning*, s and j are made senior and junior of each of the roles in RP. The case for *hybrid partitioning* is similar to that of the horizontal partitioning except that the role s is made senior to the senior-most roles of each of the linear paths formed over the roles in RP, whereas j is made the junior of each of junior-most roles of these linear paths.

Thus far, we have considered monotype linear components during the vertical and the hybrid partition. In general, the *vertical* and *hybrid partitioning* may result in hybrid linear path components. In such a case, the users originally assigned to the partitioned role need to be assigned to the role set from the partitioned set. First, consider the vertical partitioning. Here, the original users of the partitioned role is assigned to the maximal subset of the partition set, say MR, such that the roles in MR do not belong to any elements of the $\bigsqcup(H)$

Table XII. Transformation Characteristics for Different Approaches to Role Partitioning

Role $r \in \texttt{Roles}(H_o)$ *is partitioned into* $\texttt{RP} = \{x_1, x_2, \ldots, x_n\} \subset \texttt{Roles}(H_n)$		Vertical partitioning (monotype linear path)			Horizontal partitioning	Hybrid partitioning
1	Hierarchy characteristics	$LH = (\texttt{RP}, \langle f \rangle)$ (i.e. forms linear path)			No pair is hierarchically related	$H = (\texttt{RP}, [f]) = \{LH_1, LH_2, \ldots, LH_n\}$ for $n > 1$(i.e. a hierarchy which is not a linear path) such that $\texttt{Roles}(LH_i) \subset \texttt{RP}$ *Condition*: if $(s\langle f \rangle j)$ is a derived relation in H_o then at least one linear path LH_i must allow deriving relation $(s\langle h \rangle j) \in H_o$.
		if		*then*		
		$(s \geq_i r), (r\langle f \rangle j) \in H_o$ or $(s \geq r), (r \geq_i j) \in H_o$		$\langle f \rangle \in \{\geq_i, \geq\}$		
		$(s \geq r), (r \geq j) \in H_o$ or $(s \geq r), (r \geq_a j) \in H_o$		$\langle f \rangle = \geq$		
		$(s \geq_a r), (r\langle f \rangle j) \in H_o$		$\langle f \rangle \in \{\geq_a, \geq\}$		
2	Reassignment of U_r	For all $u \in U_r$, u is assigned to x_1			For all $x \in \texttt{RP}$, $u \in U_r$, u is assigned to x	For all $x \in \{S_{LH_1}, S_{LH_2}, \ldots, S_{LH_n}\}$, $u \in U_r$, u is assigned to x
3	Relation with s where $(s\langle f \rangle r) \in H_o$	$(s\langle f \rangle x_1)$			For all $x \in \texttt{RP}$, $(s\langle f \rangle x)$	For all $x \in \{S_{LH_1}, S_{LH_2}, \ldots, S_{LH_n}\}$, $(s\langle f \rangle x)$
4	Relation with r where $(r\langle f \rangle j) \in H_o$	$(x_n\langle f \rangle j)$			For all $x \in \texttt{RP}$, $(x\langle f \rangle j)$	For all $x \in \{J_{LH_1}, J_{LH_2}, \ldots, J_{LH_n}\}$, $(x\langle f \rangle j)$

of the senior-most role. In the case of the hybrid hierarchy, the original users are assigned to the set of roles that represents union of the MPs of individual elements of the complete LPDHH.

As indicated above, the need for such partitioning is primarily to restructure or redistribute permission sets in a hierarchy. Another reason for doing such partitioning may be because of the temporal properties. For example, a role may need to be vertically partitioned to arrange the temporal properties in such a way that the intervals associated with a senior role contain the intervals associated with the junior roles. Similarly, a horizontal partition may need to be done to create roles with distinct nonoverlapping intervals. Furthermore, a hybrid partitioning may be needed to properly structure very complex temporal properties. Analysis of such partitioning, based on temporal properties, has been considered in detail in a slightly different context in Joshi et al. [2005a] and also details the pros and cons of such partitioning and provides design guidelines.

6.4 Edge Deletion and Insertion

GTRBAC allows events of type (*enable*/*disable* h), which essentially adds or removes the hierarchical edge between a prespecified pair of role. Using this event, periodicity and duration constraints on hierarchical relation can be expressed. Hence, edge deletion and insertion issues can be considered as related to the design of the time-based RBAC policies that includes hierarchical relations. In a generic nontemporal RBAC framework with hybrid hierarchy, edge deletion, and insertion are important operations. However, both these operations can be viewed as operations on role addition and deletion discussed above. For instance, an edge deletion can be viewed as the deletion of the junior role of the edge and the addition of the same role with all the edges other than the deleted edge reinserted by considering the issues addressed earlier for role deletion. Similarly, an edge insertion can be viewed as role addition operation(s), if either or both of the roles in the edge did not exist in the original hierarchy. Alternatively, if both the roles of the edges are present in the hierarchy, then edge addition can be viewed as removing the junior role of the edge and reinserting it with all its original hierarchical relations as well as the new relation.

7. RELATED WORK

Several researchers have addressed issues related to inheritance semantics in RBAC [Giuri 1995, 1996; Moffett 1998; Nyanchama and Osborn 1999; Sandhu 1996, 1998] Our earlier work has addressed issues concerning the inheritance relation when temporal properties are introduced [Joshi et al. 2002]. Furthermore, to the best of our knowledge, no work has been reported in the literature that thoroughly analyzes the coexistence of different types of hierarchical relations on a set of roles. In Joshi et al. [2002, 2005b], we use the separate notion of hierarchy using *permission-usage* and *role-activation* semantics similar to the one proposed by Sandhu [1998] and strengthen Sandhu's argument that the

distinction between the two semantics is very crucial. This argument is based on the fact that the simple usage semantics is inadequate for expressing desired inheritance relation when certain *dynamic* SoD constraints are used between two roles that are hierarchically related, whereas, we emphasize the need for such distinction to capture the inheritance semantics in the presence of various temporal constraints. Sandhu's notion of activation hierarchy extending the inheritance hierarchy corresponds to the *IA* hierarchy and our *A* hierarchy corresponds to Sandhu's relation that relates two roles by activation hierarchy, but not by inheritance semantics [Sandhu 1998]. Giuri [1995, 1996] has proposed an activation hierarchy based on AND and OR roles. However, these AND–OR roles can be easily simulated within Sandhu's ER-RBAC96 model that uses usage and activation hierarchies, making Giuri's model a special case of ER-RBAC96 [Sandhu 1998]. In order to address the needs of control principles in an organization, which include *separation of duty*, *decentralization*, and *supervision and review*, Moffet [1998] and Moffett and Lupu [1999] have identified three types of hierarchies: *is a* hierarchy *activity* hierarchy, and *supervision* hierarchies. They show that for addressing more completely these control principles, we need a dynamic access-control model and a hierarchy that allows restrictive inheritance as well as dynamic propagation of access rights [Moffett and Lupu 1999]. We believe that GTRBAC's temporal constraint framework with trigger and constraint-enabling mechanisms and temporal hierarchies can provide the modeling capabilities to address such dynamic issues. Nyanchama et al. address the transformation of hierarchies in terms of addition, deletion, and partitioning of roles in the context of access rights administration [Nyanchama and Osborn 1994]. However, the analysis is limited to monotype hierarchies and does not indicate how the transformations are affected by the presence of other constraints on hierarchical roles.

Sandhu et al. [1999] have presented ARBAC97 model for administrating RBAC policies using structural properties of RBAC96 hierarchy. Similarly, in Crampton et al. [2003] have proposed a Scoped Administration of RBAC (SARBAC) using the notion of an administration scope as a unit of administration to impose conditions on hierarchy operations. The aim of both the models has been to define administrative control by defining range or scope of control for the administration of roles and hierarchical relations. By using the flexible transformation primitives presented in Section 6, the development of a more complete RBAC administration model than the ARBAC97 and SARBAC models is possible and is left as a future work. Note that although both ARBAC97 and SARBAC emphasize hierarchy management, they do not consider the coexistence of SoD constraints and role hierarchies and applies to monotype hierarchies only. Furthermore, role partitioning is a hierarchy transformation primitive that has not been addressed in the literature before.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an extensive analysis of hybrid temporal role hierarchies for GTRBAC model. We have introduced the notion of uniquely activable set of a hierarchy that identifies access capabilities of a user assigned to a role in a hierarchy in a single session. The formal results we have presented

allows determining uniquely activable sets for hybrid temporal role hierarchies and provide a basis for controlling privilege distribution to the users by restricting activable sets associated with the roles they are assigned to. The results related to the AC-equivalence between different role hierarchies also show that, in cases where the principle of least privilege is not a concern, a monotype hierarchy may be used. Furthermore, as an A hierarchy does not allow direct permission inheritance, the results show that the *A* hierarchy provides the most needed flexibility. In particular, an *A* hierarchy allows DSoD constraints to be defined on hierarchically related roles. Furthermore, the inherit-all-permission semantics of *I* hierarchy, as well as *IA* hierarchy, has several pitfalls in terms of their ability to handle many organizational control principles [Moffett 1998]. We have also introduced a set of inference rules, which can be employed to infer hierarchical relationships between pairs of roles that are not directly related. We have formally showed that the set of inference rules is sound and complete. In a complex hybrid hierarchy, these rules provide a formal basis for analyzing the permission-acquisition and role-activation semantics. We have also introduced the notion of *conditioned-derived relation* that augments the three hierarchies (*I*, *A*, and *IA* hierarchies) and facilitates capturing much fine-grained derived permission-acquisition and activation semantics within a hierarchy. We have also addressed the issue of hierarchy transformation with respect to role addition, deletion and partitioning. These transformations essentially form the basis for policy evolution in an organization. It is to be noted that transformations that retain original hierarchical semantics in a hybrid hierarchy need to be based on what type of additional role constraints exist or will be added in the hierarchy. The results presented in this paper provide a formal basis for developing administrative tools for the management of GTRBAC systems. Such security administrative functions are crucial for a well-planned, timely control of unauthorized accesses, as well as for distributing least access capabilities to users in order to allow them to carry out their activities and, at the same time, minimize damage that may be caused by misuse of privileges. We plan to extend the present work in various directions. We also plan to develop an SQL-like language for specifying temporal properties for roles and to develop a prototype of such language on top of a relational DBMS. Using the results presented here, we plan to develop efficient security administration and management tools.

APPENDIX

A

The proofs are included in the technical report version of this paper, which is available as a CERIAS technical report at https://www.cerias.purdue.edu/ with the same title.

B

The proofs are included in the technical report version of this paper, which is available as a CERIAS technical report at https://www.cerias.purdue.edu/ with the same title.

C

Here, we present an alternate way to compute the $\bigsqcup(H)$ of a hierarchy. The key issue is that a hierarchy relation generates a partial order of role sets (posets). That is, for a role set R and $\langle f \rangle \in \{\geq_{\mathrm{i}}, \geq_a, \geq\}$, a hierarchy $H = (R, \langle f \rangle)$ denotes a poset of roles, which means for $x, y, z \in R$

1. $(x \langle f \rangle x)$ (We assume that a role is senior to itself) [*reflexivity property*]
2. $(x \langle f \rangle y)$ and $(y \langle f \rangle x)$ implies $x = y$; [*antisymmetry property*]
3. $(x \langle f \rangle y)$ and $(y \langle f \rangle z)$ implies $x \langle f \rangle z$; [*transitivity property*]

Note that cases (1) and (2) are actually not allowed in the RBAC models; however, for mathematically viewing a role hierarchy as a poset, it does not introduce any problem. Case (3) is implied from the definition of each hierarchy type. Based on this, a hybrid hierarchy can be considered as a combination of the two posets over a role set related to the I and A relation (called *ISet* and *ASet*, respectively), as defined below:

$ASet = (R, \geq_{\mathrm{a}})$ and $ISet = (R, \geq_{\mathrm{i}})$ are posets defined on R that satisfy the following criteria:

a. $\forall (x \langle f \rangle y) \in H, (\langle f \rangle \in \{\geq_{\mathrm{a}}, \geq\}) \rightarrow ((x \geq_{\mathrm{a}} y) \in ASet)$
b. $\forall (x \langle f \rangle y) \in H, (\langle f \rangle \in \{\geq_{\mathrm{i}}, \geq\}) \rightarrow ((x \geq_{\mathrm{i}} y) \in ISet)$

Furthermore, a set X is called a *chain* or *total order* if, for all $x, y \in X$, either $(x \langle f \rangle y)$ or $(y \langle f \rangle x)$; and X is called an *antichain* if, for all $x, y \in X$, $(x \langle f \rangle y)$ only if $(x = y)$. Let $antichain\_set(ISet)$ be the function that computes the set of all *antichains* of $ISet$. Furthermore, let $can\_activate\_set(ASet)$ be the set of roles that a user assigned to the senior-most role in H can activate according to the A *hierarchy* relation defined in $ASet$, which is a subset of H. Let S_H denote the senior-most role in H. Now, the computation of $\bigsqcup(H)$ is given by the following theorem:

THEOREM ($\bigsqcup(H)$ *using anti_chain*): *Given a hybrid hierarchy H over roles R,* $\bigsqcup(H) = antichain\_set(ISet) \bigcap 2^{can\_activate\_set(ASet)}$

PROOF. The proof follows as $\bigsqcup(H)$ contains only those incomparable sets of roles (*antichains*), indicated by $antichain\_set\ (ISet)$, and that can be activated together, as indicated by the *chains* in the $ASet$. For instance, when there is no I hierarchy, the $antichain\_set\ (ISet)$ is a set of all the combinations of the elements of R. Based on this theorem, steps to complete $\bigsqcup(H)$ will be as follows:

1. Given original hierarchy, create two hierarchies
   a. An A-$hierarchy\ (ASet)$ containing edges for each A and IA relation in H
   b. An I-$hierarchy\ (ISet)$ containing edges for each I and IA relation in H
2. Compute the $antichain\_set$ for the $ISet$
3. Remove from $ISet$ any role set that cannot be activated by the user assigned to H_s as per the $ASet$. □

This approach characterizes the uniquely activable set (UAS) in a much simpler way. However, to compute the $\bigsqcup(H)$, it requires computing the

antichain_set and the set of roles that the user assigned to S_H can activate. The difference between this approach and the one presented in Section 4 is that the former computes $\bigsqcup(H)$ of the entire hierarchy incrementally from that of the subhierarchies, while the approach described above operates on the entire hierarchy. The approach described in Section 4, hence, facilitates the maintenance of the $\bigsqcup(H)$ of the hierarchy incrementally. That is, a tool can be easily designed to compute and maintain the $\bigsqcup(H)$ information for any subhierarchy and use that to reconstruct the $\bigsqcup(H)$ of the entire hierarchy. The $\bigsqcup(H)$ information for each role in the hierarchy is needed in order to support the authorization decision process. This would not be easy using the approach described here. Furthermore, the computation of the *antichain_set* for a general I hierarchy itself is not a straightforward task. The approach in Section 4 essentially provides one incremental way to compute the *antichains* that form the $\bigsqcup(H)$ of the given hierarchy.

REFERENCES

Barkley, J., Cincotta, A., Ferraiolo, D., Gavrila, S., and Kuhn, D. R. 1997. Role based access control for the world wide web. In *Proceedings of 20th National Information System Security Conference*. NIST/NSA.

Bertino, E. and Ferrari, E. 1999. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security 2,* 1 (Feb.), 65–104.

Bertino, E., Bonatti, P. A., and Ferrari, E. 2001. Trbac: A temporal role-based access control model. *ACM Transactions on Information and System Security 4,* 3 (Aug.), 191–233.

Biskup, J., Flegel, U., and Karabulut, Y. 1998. Secure mediation: Requirements and design. In *Proceedings of 12th Annual IFIP WG 11.3 Working Conference on Database Security*. Chalkidiki, Greece.

Crampton, J. and Loizou, G. 2003. Administrative scope: A foundation for role-based administrative models. *ACM Transsactions on Information System Security 6,* 2, 201–231.

Ferraiolo, D. F., Gilbert, D. M., and Lynch, N. 1993. An examination of federal and commercial access control policy needs. In *Proceedings of NISTNCSC National Computer Security Conference*. Baltimore, MD.

Giuri, L. 1995. A new model for role-based access control. In *Proceedings of 11th Annual Computer Security Application Conference*. New Orleans, LA.

Giuri, L. 1996. Role-based access control: A natural approach. In *RBAC '95: Proceedings of the first ACM Workshop on Role-based access control*. ACM Press, New York. 13.

Jaeger, T. and Tidswell, J. E. 2001. Practical safety in flexible access control models. *ACM Transactions on Information and System Security 4*, 2, 158–190.

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

Joshi, J. B. D., Ghafoor, A., Aref, W., and Spafford, E. H. 2001b. Digital government security infrastructure design challenges. *IEEE Computer 34*, 2 (Feb.), 66–72.

Joshi, J. B. D., Bertino, E., and Ghafoor, A. 2002. Temporal hierarchies and inheritance semantics for gtrbac. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM Press, New York. 74–83.

Joshi, J. B. D., Bertino, E., Latif, U., and Ghafoor, A. 2005a. Analysis of expressiveness and design issues for a temporal role based access control model. *IEEE Transactions on Dependable and Secure Computing 2*, 2, 157–175.

Joshi, J. B. D., Bertino, E., Latif, U., and Ghafoor, A. 2005b. Generalized temporal role based access control model. *IEEE Transactions on Knowledge and Data Engineering 17*, 1 (Jan.), 4–23.

Koch, M., Mancini, L., and Parisi-Presicce, F. 2002. A graph-based formalism for rbac. *ACM Transactions on Information and System Security 5*, 3, 332–365.

MOFFETT, J. D. 1998. Control principles and role hierarchies. In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*. ACM Press, New York. 63–69.

MOFFETT, J. D. AND LUPU, E. C. 1999. The uses of role hierarchies in access control. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*. ACM Press, New York. 153–160.

NYANCHAMA, M. AND OSBORN, S. 1999. The role graph model and conflict of interest. *ACM Transactions on Information and System Security 2*, 1, 3–33.

NYANCHAMA, M. AND OSBORN, S. L. 1994. Access rights administration in role-based security systems. In *Proceedings of the IFIP WG11.3 Working Conference on Database Security VII*. North-Holland, Amsterdam. 37–56.

OSBORN, S., SANDHU, R., AND MUNAWER, Q. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security 3*, 2, 85–106.

PARK, J. S., SANDHU, R., AND AHN, G. J. 2001. Role-based access control on the web. *ACM Transactions on Information and System Security 4*, 1 (Feb.), 37–71.

SANDHU, R. 1996. Role hierarchies and constraints for lattice-based access controls. *Computer Security—Esorics'96, LNCS N. 1146*, 65–79.

SANDHU, R. 1998. Role activation hierarchies. *Proceedings of 2rd ACM Workshop on Role-based Access Control*, 33–40.

SANDHU, R., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Computer 29*, 2, 38–47.

SANDHU, R., BHAMIDIPANI, V., AND MUNAWER, Q. 1999. The arbac97 model for role-based administration of roles. *ACM Transactions on Information and System Security 1*, 2, 105–135.

Received January 2003; revised April 2005; accepted September 2006

Title: Formal Foundations for Hybrid Hierarchies in GTRBAC
Authors: James B. D. Joshi, Elisa Bertino, Arif Ghafoor and Yue Zhang,

Author Queries

AQ1 Au: Pls. Complete Categories and Subject Descriptors and also provide appropriate General Terms.

AQ2 Au: Joshi et al. 2001a or b?

AQ3 Au: derived?

AQ4 Au: ok adding see later?

AQ5 Au: ok? Hence to Therefore?