CERIAS Tech Report 2006-30

CONFORMANCE TESTING OF TEMPORAL ROLE BASED ACCESS CONTROL

by Ammar Masood, Arif Ghafoor, Aditya Mathur

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Conformance Testing of Temporal Role Based Access Control Systems

Ammar Masood* Arif Ghafoor† Aditya Mathur‡

September 6, 2006

Abstract

Access control is a key security service at the foundation of information and system security. It has been extended with temporal constraints to support real-time considerations. Conformance testing of an access control implementation is crucial to ensure that it correctly enforces any required temporal and non-temporal policies for access control. We propose an approach for conformance testing of implementations required to enforce access control policies specified using Temporal Role Based Access Control (TRBAC) model. The proposed approach uses Timed Input Output Automata (TIOA) to model the behavior specified by a TRBAC policy. The TIOA model is then transformed to a deterministic se-FSA model that captures any temporal constraint by using two special events *Set* and *Exp*. Finally we adapt the W-method and use an integer programming based approach to construct a conformance test suite from the transformed model. The conformance test suite so generated provides complete fault coverage with respect to the proposed fault model for TRBAC specifications.

Keywords: Role Based Access Control (RBAC), Temporal Role Based Access Control (TRBAC), Finite state models, Timed Input Output Automata (TIOA), W-method, Fault model, se-FSA transformation, integer programming (IP).

1 Introduction

Access control is one of the key security services used for information and system security. To control the time-sensitive activities present in various applications such as work flow management systems and real time databases, access control specifications are augmented with temporal constraints. As an example, workflow applications used in health care setups are required to enforce temporal access constraints to ensure the security of patient records [23]. One such constraint can be to allow a doctor access to the patient record

*Purdue University, Electrical and Computer Engineering; West Lafayette, IN 47907. ammar@ece.purdue.edu.

†Purdue University, Electrical and Computer Engineering; West Lafayette, IN 47907. ghafoor@ecn.purdue.edu. Corresponding Author.

‡Purdue University, Computer Science; West Lafayette, IN 47907. apm@purdue.edu.

for only a specified duration. Role based access control (RBAC) [12], particularly well suited for specifying access control policies and rules for any arbitrary organization-specific security model, has been extended with temporal constraints to enforce time based access requirements [5, 16].

Access policies in RBAC are specified by mapping permissions to user assigned roles. The permissions map the possible authorizations of a role in terms of specific operations that a user activating that role can perform on the corresponding system resource. A user assigned to a role cannot invoke the permissions of that role until the role has been activated. We focus on temporal RBAC (TRBAC) systems as they are based on RBAC that provide simplified security management [6] by allowing capabilities such as the abstraction of roles and the use of role hierarchy, principles of least privilege and separation of duty (SoD), and policy neutrality [15].

In this paper we focus on conformance testing of software implementations required to enforce TRBAC policies. We refer to such an implementation as "TRBAC system," or "access control implementation under test," or simply as ACUT. Our aim is to devise a scalable and effective conformance testing technique that provides complete fault coverage with respect to a reasonable TRBAC fault model. This fault model is developed using a mutation based approach similar to the one used in other work [25].

A Timed Input Output Automaton (TIOA) referred as TRBAC_{M} is used to model the expected behavior of an ACUT with respect to a TRBAC policy. A conformance test suite (CTS) is then generated from the TRBAC_{M} by first transforming it to an se-FSA [18], and then using integer programming for determining the time stamps that satisfy the required temporal constraints in the access control policy by considering the sending of inputs and the monitoring of outputs at times that are an integral multiple of a minimum resolution. The ACUT is then executed against all elements of the CTS using the test architecture proposed in [17].

The proposed conformance testing approach only targets conformance testing of the ACUT with respect to a specific TRBAC policy. Additionally, functional testing is required to guarantee that ACUT will correctly enforce all TRBAC policies. As the set of all TRBAC policies is infinite, a representative subset of policies is used. Functional testing of ACUT is carried out as per the methodology presented in [24].

Contributions: (a) A fault model corresponding to TRBAC policy ($TRBAC_P$) specification. (b) A technique for modeling the expected behavior of TRBAC systems (ACUT) using TIOA. (c) Conformance testing strategy which provides complete fault coverage with respect to faults in the TRBAC fault model.

Organization: Section 2 outlines the sequence of steps in the proposed conformance testing approach. Section 3 defines the TRBAC policy specification, i.e., $TRBAC_P$, and presents an example used throughout the paper to explain the proposed conformance testing technique. The syntax and semantics of TIOA is also reviewed in Section 3. Section 4 discusses the "conformance relation" used to signify the connotation of conformance for ACUT. The conformance relation is used in Section 5 to study the fault model corresponding to any TRBAC policy specification. Construction of a TIOA based model of any TRBAC policy is discussed in Section 6. Section 7 describes our procedure for the generation of CTS from this model. Two

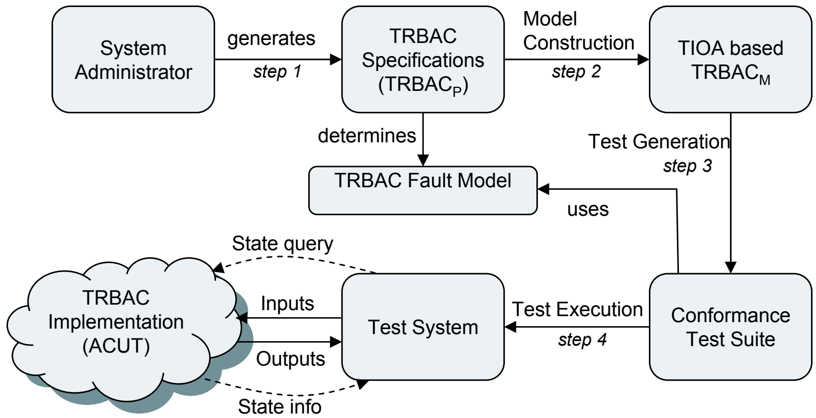


Figure 2-1: Proposed approach for conformance testing of TRBAC Systems (ACUT)

heuristics are briefly discussed in Section 8 to illustrate the application of state abstraction based techniques to reduce the size of the TIO model for any given policy. Section 9 briefly reviews the related work and Section 10 summarizes the current work.

2 Proposed Approach for Conformance Testing

The proposed approach for conformance testing of a $TRBAC_P$ implementation, an ACUT, is shown in Figure 2-1. In step-1 a system administrator generates a TRBAC policy referred to as $TRBAC_P$. This policy is used to construct the expected behavior of the ACUT in step 2. This behavior is captured as a TIOA model referred to as TRBAC_M. The test generator uses TRBAC_M as input and generates the CTS in step 3. A specific test system architecture, discussed in detail in Section 7.3, is used to execute the elements of the CTS against the TRBAC ACUT in step 4. The results of test execution are then compared with the expected behavior implied by TRBAC_M to validate ACUT conformance with respect to $TRBAC_P$.

Note that the four steps described in Figure 2-1 deal with the testing of an ACUT with respect to only one TRBAC policy. In practical environments, one would expect, or at least desire, that while policies change the ACUT does not. A functional testing procedure to check for the correctness of an ACUT against all policies is proposed elsewhere [24].

3 Background

3.1 Temporal RBAC Policy Specification

Temporal RBAC extends the RBAC model by time constraining a user's access to system resources. This is achieved by restricting the time duration of user-role activations, assignments, and permission-role assignments. A TRBAC specification also includes the rules for non-temporal user-role assignment (activation), separation of duty (SoD) constraints [2], role hierarchy semantics, and static/dynamic user (role) cardinality constraints. A formal definition of TRBAC policy specification follows.

Definition 3.1 ($\mathbf{TRBAC_P}$) *A TRBAC policy $TRBAC_P$ is a 17-tuple* $(U, R, Pr, Status, Permitted, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s, \Re)$, *where*

- *U, R and Pr are, respectively, finite sets of users, roles and permissions,*
- *$Status = UR_{assign} \cup UR_{active} \cup PR_{assign}$ is a set of status predicates partitioned as follows:*

  (a) UR_{assign}: $U \times R \rightarrow \{0,1\}$ where a 1(0) indicates that the given user is assigned (not assigned) to the given role.

  (b) UR_{active}: $U \times R \rightarrow \{0,1\}$ where a 1(0) indicates that the given user has activated (not activated) the given role.

  (c) PR_{assign}: $Pr \times R \rightarrow \{0,1\}$ where a 1(0) indicates that the given permission is assigned (not assigned) to the given role.

- *$Permitted = UR_{canAssign} \cup UR_{canActivate} \cup PR_{canAssign}$ is a set of allowable predicates partitioned as follows:*

  (a) $UR_{canAssign}$: $D_1 \subseteq U \times R \rightarrow \{1\}$ where the value of 1 indicates that the given user can be assigned to the given role.

  (b) $UR_{canActivate}$: $D_2 \subseteq U \times R \rightarrow \{0,1\}$ where a 1(0) indicates that the given user can activate (not activate) the given role.

  (c) $PR_{canAssign}$: $D_3 \subseteq Pr \times R \rightarrow \{1\}$ where the value of 1 indicates that the given permission can be assigned to the given role.

- *$\leq_A \subseteq R \times R$ and $\leq_I \subseteq R \times R$ are, respectively, activation and inheritance hierarchy relations on roles,*
- *$I = \{AS, DS, AC, DC, AP, DP\}$ is a finite set of allowable input requests, where AS, DS, AC, DC, AP, DP are, respectively* Assign, Deassign, Activate, *and* Deactivate *requests for user-role assignment and activation and* Assign *and* Deassign *for permissions-role assignments. AS, AC, AP inputs optionally specify $t \in Z^+$, where t stipulates the maximum time duration of the corresponding assignment or activation and Z^+ denotes the set of non-negative integers..*
- *$S_u, D_u : U \rightarrow Z^+$ are, respectively, static and dynamic cardinality constraints on U*
- *$S_r, D_r : R \rightarrow Z^+$ are, respectively, static and dynamic cardinality constraints on R*

- *$SSoD, DSoD \subseteq 2^R$ are, respectively, static and dynamic Separation of Duty (SoD) sets*

- *$S_s : SSoD \rightarrow Z^+$ specifies the cardinality of the $SSoD$ sets*

- *$D_s : DSoD \rightarrow Z^+$ specifies the cardinality of the $DSoD$ sets*

- *$\Re$ is the rule set as given in Definition 3.2.*

For clarity of presentation, a policy $TRBAC_P$ is simply referred as P unless noted otherwise. P is explicitly attached with each element of the above 17-tuple when there is a need to distinguish it from that of another policy. For example $Status(P)$ and $Status(P')$ are the status predicates corresponding, respectively, to policies P and P'.

The activation hierarchy relation (A-hierarchy [27]) $r_i \leq_A r_j$ implies that a user u_k assigned to r_j is also able to activate r_i without being assigned to it i.e. $UR_{assign}(u_k, r_i) = 0$. The inheritance hierarchy relation (I-hierarchy [27]) $r_i \leq_I r_j$ means that a permission p_k assigned to r_i is also accessible by r_j without being directly assigned to it i.e. $(p_k, r_j) \notin PR_{canAssign}$. The static (dynamic) cardinality of a user specifies the maximum number of roles it can be assigned to (can activate). Similarly, the static (dynamic) cardinality of each role specifies the maximum number of users who can be assigned to (can activate) this role.

The $SSoD$ ($DSoD$) [2] specifies the sets of roles to which users can only be simultaneously assigned (can simultaneously activate) provided such assignments (activations) do not violate the $SSoD$ ($DSoD$) set cardinality constraint i.e. $S_s(SSoD)$ ($D_s(DSoD)$). $S_s(SSoD)$ ($D_s(DSoD)$) constrains the maximum number of roles to which a user can be simultaneously assigned (can simultaneously activate) in the given $SSoD$ ($DSoD$) set.

The access control decisions are guided by the formally specified rules in the rule set ($\Re$). The set of $Status$ and $Permitted$ predicates in P are used to define these rules, which constrain the possible assignments and activations within the given TRBAC policy. The rule set ($\Re$) is defined below.

Definition 3.2 ($\Re$) *The rule set $\Re = \{\gamma_{urAssignCard}, \gamma_{urActiveCard}, \gamma_{urSSoD}, \gamma_{urDSoD}, \gamma_{urHier}, \gamma_{prHier}, \gamma_1, \gamma_2, \gamma_3\}$, given in Table 1, describes the set of system rules that dictate the access control decisions in a given $TRBAC_P$.*

Table 1: Rules in the rule set $\Re$

Rule	**Explanation**
$\gamma_{urAssignCard}(u \in U, r \in R) = 1$ iff $UR_{canAssign}(u,r) = 1 \wedge UR_{assign}(u,r) = 0 \wedge \sum_R UR_{assign}(u,r_i) < S_u(u) \wedge \sum_U UR_{assign}(u_i,r) < S_r(r)$	$\gamma_{urAssignCard}$ can only be 1 if the static cardinality constraints corresponding to the given user u and role r are not violated by the assignment of u to r

Continued on next page

Rule	Explanation
$\gamma_{urActiveCard}(u \in U, r \in R) = 1$ iff $[UR_{canActivate}(u,r) = 1 \vee \gamma_{urHier}(u,r) = 1] \wedge UR_{active}(u,r) = 0 \wedge \sum_{R} UR_{active}(u,r_i) < D_u(u) \wedge \sum_{U} UR_{assign}(u_i,r) < D_r(r)$	$\gamma_{urActiveCard}$ can only be 1 if the dynamic cardinality constraints corresponding to the given user u and role r are not violated by the activation of r by u
$\gamma_{urSSoD}(u \in U, r \in R) = 1$ iff $\forall R' \in SSoD \| r \in R', \sum_{R'} UR_{assign}(u,r_i) < S_s(R')$	γ_{urSSoD} can only be 1 if user u can be assigned to r such that the total number of user-role assignments corresponding to u in all the sets $R' \| r \in R'$ are less than the cardinality of that set
$\gamma_{urDSoD}(u \in U, r \in R) = 1$ iff $\forall R' \in DSoD \| r \in R', \sum_{R'} UR_{active}(u,r_i) < D_s(R')$	γ_{urDSoD} can only be 1 if user u can activate r such that the total number of user-role activations corresponding to u in all the sets $R' \| r \in R'$ are less than the cardinality of that set
$\gamma_{urHier}(u \in U, r \in R) = 1$ iff $UR_{active}(u,r) = 0 \wedge \exists r' \in R' \| R' \subseteq R \wedge UR_{assign}(u,r') = 1$ where R': $\{r' \| r \leq_A r'\}$. R' is the set of all roles senior to r as per A-hierarchy semantics (r is also member of this set)	$\gamma_{urHier}(u,r) = 1$ implies that there is at least one such role r' (could be r), senior to r, to which u is currently assigned. A-hierarchy semantics thus permit activation of a junior role by the user provided that the user is assigned to at least one role senior to the former
$\gamma_{prHier}(p \in Pr, r \in R) = 1$ iff $\exists r' \in R' \| R' \subseteq R \wedge PR_{assign}(p,r') = 1$ where R': $\{r' \| r' \leq_I r\}$. R' is the set of all roles junior to r as per I-hierarchy semantics (r is also member of this set)	$\gamma_{prHier}(p,r) = 1$ implies that there is at least one such role r' junior to r to which p is currently assigned. I-hierarchy semantics thus permit assignment of permissions to a senior role on the basis of there being assigned to a junior role
γ_1: $AS(u \in U, r \in R, t \in Z^+) \Rightarrow Update_{status}[UR_{assign}(u,r) = 1, t] \wedge Update_{permitted}$ $[UR_{canActivate}(u,r) = 1]$ iff $\gamma_{urAssignCard}(u,r) = 1 \wedge \gamma_{urSSoD}(u,r) = 1$ and $DS(u \in U, r \in R) \Rightarrow Update_{status}$ $[UR_{assign}(u,r) = 0] \wedge Update_{permitted}$ $[UR_{canActivate}(u,r) = 0] \wedge Update_{status}$ $[UR_{active}(u,r') = 0]$ $\forall r' \in R' \| R' \subseteq R$, R': $\{r' \| r' \leq_A r\}$. $Update_{status}$ $[x \in Status = 1, t]$ implies that assignments or activations in the current TRBAC state are updated so that the current value of the corresponding predicate becomes 1, t specifies the maximum time duration after which the value would again change from 1 to 0. It is considered that the deassignment and deactivation inputs are automatically issued by the system on completion of time duration t	This rules ensures that the user-role assignment corresponding to the input $AS(u,r,t)$ is allowed such that (1) the user/role static cardinality constraints and role $SSoD$ constraints are not violated by such assignment, (2) the assignment is only valid for time duration t. After t time units the de-assignment input $DS(u,r)$ will be automatically applied by the system without any user intervention. This rule thus restricts the maximum time duration of an assignment to t specified in the input request and after that duration it also deactivates the (u,r) pair and all such user-role pairs which correspond to the user u activation of the junior roles of r

Continued on next page

Rule	Explanation
γ_2: $AC(u \in U, r \in R, t \in Z^+) \Rightarrow Update_{status}[UR_{active}(u,r) = 1, t]$ iff $\gamma_{urActiveCard}(u,r) = 1 \wedge \gamma_{urDSoD}(u,r) = 1$ and $DC(u \in U, r \in R) \Rightarrow Update_{status}$ $[UR_{active}(u,r) = 0]$	This rule ensures that the user-role activation corresponding to the input $AC(u,r,t)$ is allowed such that (1) the user/role dynamic cardinality constraints, role $DSoD$/A-hierarchy constraints are not violated by such activation, (2) the activation is only valid for time duration t. The rule $\gamma_{urActivationCard}(u,r)$ ensures that either u is assigned to r at the time of this request, or the given activation is permitted via A-hierarchy semantics. In case of conflict in durations permitted by AS and AC, the duration permitted by former will have precedence on the later as the expiry of former would automatically result into deactivation of the given and junior user-role pairs.
γ_3: $AP(p \in Pr, r \in R, t \in Z^+) \Rightarrow Update_{status}[PR_{assign}(p,r) = 1, t]$ iff $PR_{canAssign}(p,r) = 1 \vee \gamma_{prHier}(p,r) = 1$ and $DP(p \in Pr, r \in R) \Rightarrow Update_{status}$ $[PR_{assign}(p,r') = 0]$ $\forall$ $r' \in R' \| R' \subseteq R$, R': $\{r' \| r \leq_I r'\}$. For all the roles senior to r corresponding to I-hierarchy semantics, the request $AP(p,r',t)$ $r' \in R' \| R' \subseteq R$, R': $\{r' \| r \leq_I r'\}$, is considered to be automatically issued by the system.	This rules ensures that the permission-role assignment corresponding to the input $AP(p,r)$ is allowed such that (1) either such assignment is permitted directly in the policy or is allowed by virtue of I-hierarchy semantics (2) the assignment is only valid for time duration t. After the expiration of duration t, the deassignment input DP will be automatically applied by the system for the given and senior permission-role pairs

Example 1. Consider a simple TRBAC policy specification for a medical information system. There are two roles `SeniorDoctor` and `TraineeDoctor` (for simplicity we refer these as r_1 and r_2 respectively) and two users Bob and Alice (referred to as u_1 and u_2 respectively). Figure 3-1(a) shows the various basic user-role assignment scenarios allowed by the TRBAC policy of the domain. S1 scenario shows that both u_1 and u_2 are allowed to be assigned to r_1 and r_2 respectively. However, S2 indicates that if u_1 is already assigned to r_1 then it cannot be assigned to r_2 simultaneously. Thus this represents the domain static SoD policy that a user cannot be simultaneously assigned to both `SeniorDoctor` and `TraineeDoctor` roles. Scenarios S3 and S4 are complementary to S1 and S2 as they indicate that if u_1 is initially assigned to r_2 then it cannot be simultaneously assigned to r_1.

Figure 3-1(b) illustrates the temporal impact of various user-role assignment and activation inputs. The horizontal axis represents the value of global clock t. The lines above the horizontal axis, which show various status predicates, denote that corresponding user-role activation or assignment is valid for the time duration specified along the horizontal axis. The given $TRBAC_P$ is: $U = \{u_1, u_2\}$, $R = \{r_1, r_2\}$, $SSoD = \{S_1 = (r_1, r_2)\}$, $S_s(S_1) = 1$, $UR_{canAssign}(u_1, r_1) = UR_{canAssign}(u_1, r_2) = UR_{canAssign}(u_2, r_2) = 1$, $S_u(u_1) = D_u(u_1) = 2$, $S_u(u_2) = D_u(u_2) = 1$, $S_r(r_1) = D_r(r_1) = 1$, $S_r(r_2) = D_r(r_2) = 2$.

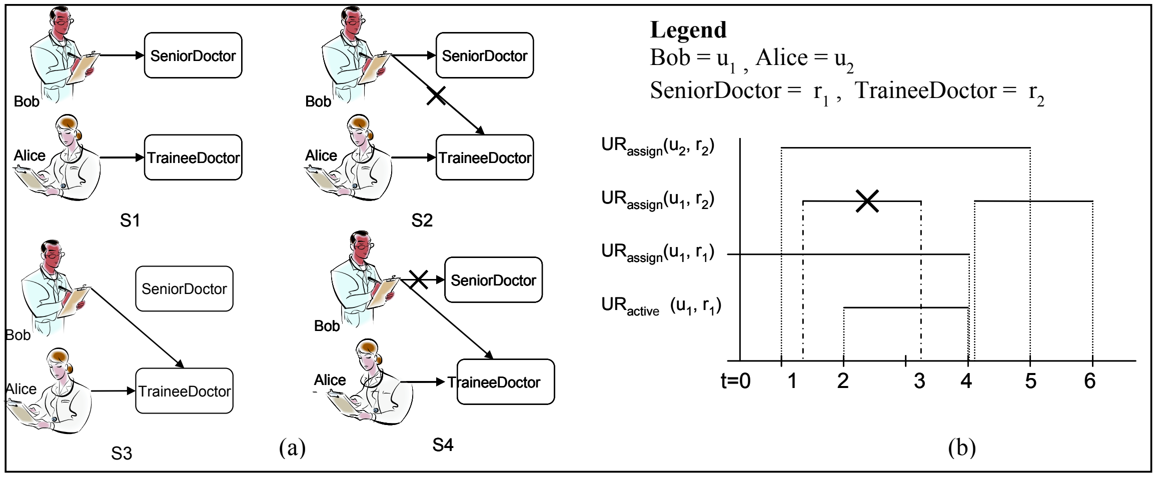


Figure 3-1: (a) Various scenarios of user-role assignments permitted by domain TRBAC specification, (b) Effect of temporal and non-temporal constraints on user-role assignments and activation

Consider that in relation to scenario S1, at $t = 0$, u_1 is assigned to r_1 for 4 time units ($AS(u_1, r_1, 4)$ *issued at* $t = 0$), and u_2 is assigned to r_2 at $t = 1$ for same duration ($AS(u_2, r_2, 4)$ *issued at* $t = 1$), correspondingly the horizontal lines for status predicates $UR_{assign}(u_1, r_1)$ and $UR_{assign}(u_2, r_2)$ extends from 0 to 4 and from 1 to 5, respectively. Now if a request for u_1 activation of r_1 is made at $t = 2$ for 3 time units ($AC(u_1, r_1, 3)$ *issued at* $t = 2$), then by virtue of the duration constraint on the assignment of u_1 to r_1 the activation would also be terminated at $t = 4$ together with the removal of the assignment.

Considering the static SoD constraint, corresponding to scenario S2, on the concurrent assignment of a user to roles r_1 and r_2, it restricts u_1 ability to be assigned to r_2 before $t = 4$; because, earlier than that rule γ_1 will be violated if such assignment is made ($\gamma_{urAssignCard}(u_1, r_2) = 1 \wedge \gamma_{urSSoD}(u_1, r_2) = 1$ should be true but $\gamma_{urSSoD}(u_1, r_2) = 0$ at that time). Hence the input request $AS(u_1, r_2, 2)$ cannot be granted before $t = 4$ but will be valid at/after $t = 4$. ■

3.2 Timed Input Output Automata (TIOA)

The inclusion of temporal constraints in TRBAC requires precise modeling of real-time considerations which cannot be achieved by traditional finite state machines. We therefore use timed automata [3, 13], in particular Timed Input Output Automaton (TIOA) to model real-time constraints in a TRBAC specification. The syntax and semantics of TIOA is explained in detail next.

TIOA [13] are finite automata which partition the actions into input and output actions. Time is incorporated through the use of (real-valued) clocks; thus TIOA is based on dense time semantics. A TIOA is a finite state automaton with a finite set of locations and a set of labeled transitions. Each transition is labeled with an action which belongs to either the set of input actions or the set of output actions (hence the name timed input output automaton). The input and output actions begin with "?" and "!", respectively. The set of real-valued clocks C is used to specify timing constraints (known as guards) on the transitions. A transition is only executed if the associated guard satisfies the current value of clocks; moreover, a subset of clocks may be reset on executing a transition.

Definition 3.3 (TIOA) *A Timed Input Output Automaton (TIOA) is a tuple* $A = \{L, l_0, I, O, C, T\}$ *where:*

- L *is a finite set of locations,* $l_0 \in L$ *is the initial location,* I *is a finite of set of input actions, where each input action begins with "?",* O *is a finite of set of output actions, where each output action begins with "!" and* C *is a finite set of clocks.*
- $T \subseteq L \times (I \cup O) \times \Phi(C) \times 2^C \times L$ *is a set of transitions. A transition* $(l, \{?i, !o\}, g, R, l')$ *represents an edge from the location* l *to* l' *on input or output action* $?i$ *or* $!o$*. The guard* $g \in \Phi(C)$ *specifies the clock constraint which must be satisfied to enable execution of this transition. The set* $R \subseteq 2^C$ *gives the set of clocks which are reset to* 0 *on executing this transition.*

The transitions are assumed to be instantaneous in a TIOA. The term $\Phi(C)$ signifies the set of clock constraints specified using the clocks in C, where a guard $g \in \Phi(C)$ is defined by the grammar: $g := x \leq c|c \leq x|x < c|c < x|x = c|g \wedge g$. In this grammar $x \in C$ and $c \in Z^+$ and Z^+ is the set of non-negative integers. A clock valuation is a function $v : C \rightarrow \mathbb{R}_{\geq 0}$ that assigns a non-negative real number to each clock in C. For a valuation v, $v + \delta(\delta \in \mathbb{R}_{\geq 0})$ denotes the valuation that assigns each clock $x \in C$ the value $v(x) + \delta$. For $Y \subseteq C$, $v[Y := 0]$ denotes the clock valuation for C which assigns 0 to each $x \in Y$ and agrees with v over other clocks. The set of all clock valuations is denoted by V_C. A valuation v satisfies a guard g if and only if g holds under v (it is represented as $v \approx g$). In the definition of TIOA it is assumed that the domain of each clock $x \in C$ is bounded to $[0, C_x] \cup \{\infty\}$ where $C_x = max\{c|c \textit{ is used in a g over } x\}$.

The semantics of a TIOA A is defined by associating an infinite state graph or Labeled Transition System (LTS) $S_A = \{Q, \mathbb{R}_{\geq 0} \cup (I \cup O), \xrightarrow{a}\}$ with A where $a \in \mathbb{R}_{\geq 0} \cup (I \cup O)$ and:

- Q is the set of all states where each state is a pair (l, v) such that l is a location of $A(l \in L)$ and v is a clock valuation for $C(v \in V_C)$. The initial state of S_A is represented by (l_0, v_0) where $v_0(x) = 0$ for all clocks $x \in C$.
- Edges of S_A are given by the relation $\xrightarrow{a}$ and are labeled with labels from the alphabet $\mathbb{R}_{\geq 0} \cup (I \cup O)$. There are two types of edges, discrete and timed edges. A discrete edge corresponds to a transition (l, a, g, r, l') of A and is represented as $(l, v) \xrightarrow{a} (l', v[r := 0])$ where $a \in (I \cup O)$ and $v \approx g$. For a state (l, v) and a time increment $\delta \in \mathbb{R}_{\geq 0}$ the timed edge $(l, v) \xrightarrow{\delta} (l, v + \delta)$ represents passing of time.

Infinite states of S_A are by virtue of the infinite number of timed edges that can exist in S_A. A timed word over the alphabet $\mathbb{R}_{\geq 0} \cup (I \cup O)$, is a sequence $w = (a_0, t_0), (a_1, t_1), \ldots, (a_k, t_k)$ where each $a_i \in$

$\mathbb{R}_{\geq 0} \cup (I \cup O)$, each $t_i \in \mathbb{R}_{\geq 0}$, $0 \leq i \leq k$ and the occurrence time t_i increases monotonically. A *run* of A (starting from the initial location l_0) over w is a series $(l_0, v_0) \xrightarrow{t_0} (l_0, v_0 + t_0) \xrightarrow{a_0} (l_1, v_1) \xrightarrow{t_1 - t_0} (l_1, v_1 + (t_1 - t_0)) \rightarrow \cdots \xrightarrow{a_k} (l_{k+1}, v_{k+1})$ where $v_i = v_{i-1} + (t_{i-1} - t_{i-2}), i > 1$

The set of timed words accepted by A is denoted by $L(A)$ and it signifies the valid runs of A over the alphabet $\mathbb{R}_{\geq 0} \cup (I \cup O)$. For a state $s \in Q$ and a timed word w, we write $s \xrightarrow{w}$ iff $s \xrightarrow{w} s'$ for some $s' \in Q$. Time can progress in A iff the automaton is timelock free [29], i.e., if every infinite run of S_A is *strongly non-Zeno*.

The *non-Zeno* property ensures that A does not force its environment to provide an input by blocking time [20]. A is strongly non-Zeno iff for any state $s \in Q$ and any $t \in \mathbb{R}_{\geq 0}$ there is a timed output trace $w = (a_0, t_0), (a_1, t_1), \ldots, (a_k, t_k)$ where $a_i \in (\mathbb{R}_{\geq 0} \cup O)$, $0 \leq i \leq k$ such that $s \xrightarrow{w}$ and $\sum_i (t_{i+1} - t_i) \geq t$. A *timed automaton* would be *strongly non-Zeno* if it is ensured that at least one unit of time lapses in each of its loop [29], in case of TIOA A this requirement is to hold for only such loops with actions from the set $(\mathbb{R}_{\geq 0} \cup O)$. It is important to verify that a TIOA is strongly non-Zeno as timelocks are modeling errors which should be resolved.

Next we describe the conformance relation used in Section 5 to study the TRBAC fault model.

4 Conformance Relation

Let P be a TRBAC policy in effect, ACUT a correct implementation that enforces P and no other policy, and a possibly faulty ACUT′ required to enforce P. Let $Rq(up, r), up \in (U \cup Pr)$, be a well formed request such that $Rq \in I$ and $(up, r) \in (U \times R)$ for $up \in U$, and $(up, r) \in (Pr \times R)$ for $up \in Pr$. $Rq(up, r)$ is considered ill-formed when any one or more of the following conditions does not hold: $Rq \in I$, $up \in (U \cup Pr)$, and $r \in R$. The state of the ACUT with respect to P is the set $Status = UR_{assign} \cup UR_{active} \cup PR_{assign}$. All the status predicates of $Status$ are 0 at the start of ACUT execution. $Status$ changes in response to requests $Rq(up, r) \in I$. We write $Status'_{\text{ACUT}} = Status_{\text{ACUT}}[Rq(up, r)]$ to indicate that the updated status of ACUT in response to request Rq is $Status'_{\text{ACUT}}$ if the status prior to receiving $Rq(up, r)$ was $Status_{\text{ACUT}}$.

ACUT′ is said to conform *behaviorally* to ACUT with respect to policy P, under the following conditions.

1. For all requests $Rq(up, r) \in I$, if $Status'_{\text{ACUT}} = Status_{\text{ACUT}}[R(up, r)]$ then $Status'_{\text{ACUT}'} = Status'_{\text{ACUT}} = Status_{\text{ACUT}'}[Rq(up, r)]$.

2. For all ill-formed requests $Rq(up, r)$, $Status_{\text{ACUT}}[Rq(up, r)]$ and $Status_{\text{ACUT}'}[Rq(up, r)]$ remain unchanged.

3. While there are no requests $\forall t \in \mathbb{R}_{\geq 0}$, $Status_{\text{ACUT}'} = Status_{\text{ACUT}}$.

Table 2: TRBAC faults due to mutations of elements of P

Structures Mutated	**Possible Impact on TRBAC ACUT′ (Fault)**
$UR_{canAssign}, S_u, S_r, SSoD, S_s$	UR1, UR2
$PR_{canAssign}, \leq_I$	PR1, PR2
$UR_{canActivate}, \leq_A, D_u, D_r, DSoD, D_s$	UA1, UA2
$\gamma_1, \gamma_2, \gamma_3$	All Temporal and Non-temporal faults

Stated informally, the first two conditions imply that ACUT′ (a) assigns (deassigns) and activates (deactivates) a role only if such assignment (deassignment) and activation (deactivation) is allowable by the current policy, (b) assigns (deassigns) a set of permissions to (from) a role only if allowable by the current policy, and (c) ignores all ill-formed requests. The last condition implies that for all times the $Status$ of ACUT and ACUT′ should remain unchanged in the absence of any input. Note that this condition does not imply that the state of an ACUT cannot change with time, rather if there is a change in the ACUT state then correspondingly similar change is expected in the state of the conforming ACUT′.

5 TRBAC Fault Model

The TRBAC fault model is derived using a mutation based approach [25]. The mutants $P' \neq P$ are obtained by applying the *set mutation* operators to the sets $Permitted, \leq_A, \leq_I, SSoD$ and $DSoD$ in P, *element modification* operators to the range of functions S_u, D_u, S_r, D_r, S_s and D_s and *rule mutation* operators to the rules γ_1, γ_2 and γ_3 in $\Re(P)$. We consider three types of set mutation operators: modification of an element, addition of an element, and removal of an element. The semantics of element modification depends on the type of the element, which in case of another set implies recursive application of set mutation operators on the element. Only the application of *rule mutation* operator on a premise $i \in I$ varies in constructing the TRBAC fault model. As an $i \in I(TRBAC_P)$ can also specify $t \in Z^+$, therefore, in case of specification of duration the rule mutation operator will replace t with $t+1$ or $t-1$. The details of application of other mutation operators are not given due to space limitation and can be found in [24].

Table 2 illustrates that the application of a mutation operator to P results in a policy P' which implies the possible presence of one more more faults in the ACUT′. As observed from Table 2 and Figure 5-1, TRBAC faults can be broadly classified into two types: non-temporal and temporal faults. Non-temporal faults are related to user-role assignment, permission-role assignment, and user-role activation. Temporal faults are further categorized into hierarchical enforcement, duration widening, and duration restriction faults.

As shown in Figure 5-1, each type of non-temporal fault is further categorized into two subcategories. Fault type UR1 restricts an authorized user from being assigned to a role or leads to an unauthorized deassignment. Fault type UR2 may lead to unauthorized role assignments. PR1 faults restrict a permission

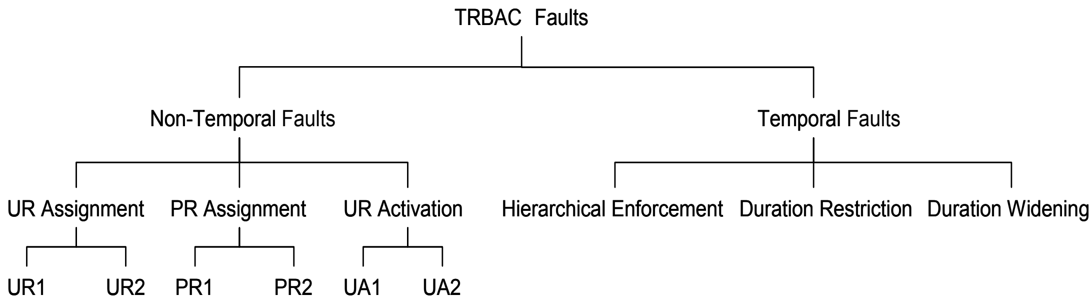


Figure 5-1: A fault model for evaluating the effectiveness of tests for TRBAC implementations.

being assigned to an authorized role or cause an unauthorized deassignment. PR2 faults assign a permission to an unauthorized role. UA1 and UA2 faults are similar to UR1 and UR2 and impact role activation. The temporal faults are explained in detail next.

5.1 Hierarchical Enforcement Faults

A-hierarchy semantics allows user u assigned to a role r to activate a junior role $r' \leq_A r$ without any explicit assignment to r'. Similarly a permission-role assignment can allow the automatic assignment of corresponding permission to all the senior roles by virtue of I-hierarchy semantics. P requires that the temporal constraints on explicit user-role and permission-role assignments are also consistently enforced on implied activations or assignments (for clarity of discussion we consider a user-role activation corresponding to a user-role assignment, also as implied activation, e.g., u_1r_1 activation corresponding to u_1r_1 assignment). However, errors in an ACUT$'$ may lead to erratic enforcement of temporal constraints on implied activations or assignments. Therefore such faults are considered as hierarchical enforcement faults.

An instance of a hierarchical enforcement fault is illustrated in Figure 5-2(a). P considered in this instance is different from the one given in Example 1 as in this case we consider that r_1 is senior to r_2 by virtue of A-hierarchy. Hence u_1 assignment to r_1 also enables u_1 to activate r_2 without any explicit assignment to r_2. Furthermore there is no SoD constraint, i.e. $SSoD = \{\}$. The temporal constraint that u_1r_1 assignment is restricted to a total duration of 4 time units (tu's) also requires discontinuation of u_1r_2 activation at $t = 4$, however the presence of hierarchical enforcement fault in the faulty ACUT$'$ permits u_1r_2 activation to continue beyond this time.

5.2 Duration Restriction Faults

The duration constraint on an event (user-role or permission-role assignment or user-role activation) requires that starting from the time the event request is generated, e.g., $AS(u_1, r_1, 4)$ issued at $t = 0$ in Figure 5-2(b), the duration for which the corresponding event ($UR_{assign}(u_1, r_1) = 1$ in this case) is valid should be

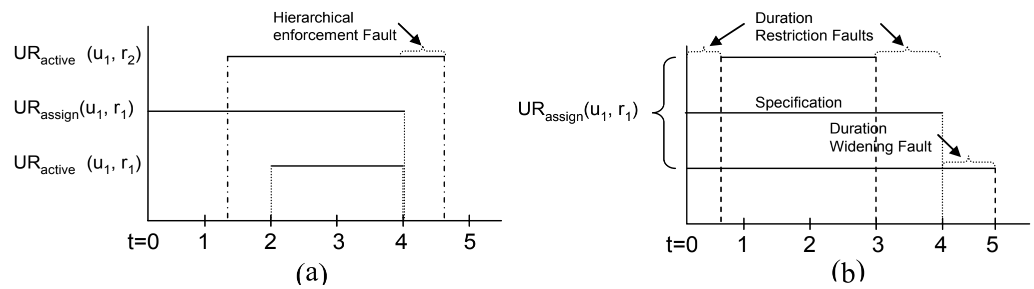


Figure 5-2: Examples of temporal faults in an ACUT$'$, (a) A hierarchical enforcement fault, (b) Duration widening and restriction faults.

accurately enforced to be equal to the value specified by the constraint (4 tu's in this case). The presence of duration restriction faults in an ACUT$'$ would restrict the actual duration of the event to be less than the one specified in P. Duration restriction faults in an ACUT$'$ can limit the actual duration of an event in two ways: by applying the deassignment/deactivation inputs before the time as required by P, or by delaying the required activations/assignments in the ACUT$'$ as compared to P.

Figure 5-2(b) illustrates both ways by which duration restriction faults affect the behavior of an ACUT$'$. It shows that although P requires $u_1 r_1$ assignment to be valid from $t = 0$ to $t = 4$, yet the same assignment is initially delayed in the ACUT$'$ and then deassignment occurs before $t = 4$ contrary to P. Duration restriction faults that delay the required activation/assignment can also be considered as non-temporal faults (of type UR1, UA1 or PR1).

5.3 Duration Widening Faults

As the name suggests, the impact of such faults is obvious in an ACUT$'$ – the presence of duration widening faults would cause the duration of the associated event to be larger then the one allowed by P. The example in Figure 5-2(b) demonstrates one such fault where the faulty behavior of ACUT$'$ leads to an extension of the duration of $u_1 r_1$ assignment to more than specified by P.

We are justified in treating the hierarchical enforcement faults as duration widening or restriction faults. However, these faults are treated as a separate class because of their direct relationship with the constraints related to hierarchy semantics.

In the next section we discuss the details of our proposed technique for capturing the expected behavior of a TRBAC ACUT.

6 Modeling The Expected Behavior of ACUT

As already mentioned we use TIOA to model real time constraints in P. For a conformance testing approach to be effective in detecting all types of faults that can exist in an ACUT (i.e., all the temporal and non-temporal faults in TRBAC fault model identified in Section 5), it is essential that the TRBAC model, referred as TRBAC_M, should encapsulate all possible behaviors of the ACUT. This requires TRBAC_M to be able to capture the state of the ACUT in terms of all possible user-role assignments/activations and permission-role assignments that can exist in the ACUT, and the state transitions as valid actions determined by P.

There are two options in constructing the TRBAC_M: (1) the requirements implied by P for user-role activations and assignments and permission-role assignments are treated in a single monolithic model, and (2) divide the ACUT behavior into parts and thus describe the ACUT compositionally. We opted for the second option because of the convenience in reasoning the correctness of ACUT behavior with respect to P in compositional construction. Further, we consider it easier to extend TRBAC_M to model additional temporal constraints as they are added to TRBAC, if the ACUT behavior is described compositionally. We have considered compositional construction of TRBAC_M in terms of parallel composition of user-role and permission-role models, i.e., $\text{TRBAC}_\text{M} = \text{UR}_\text{M} \parallel \text{PR}_\text{M}$. The parallel composition ($\parallel$) considered here is the one defined for parallel composition of timed automata in [29].

6.1 UR Model

The UR model (UR_M) captures the desired response of an ACUT, as required by P, corresponding to all sequences of user-role assignments, deassignments, activations, and deactivations. UR_M thus encapsulates the conforming behavior of an ACUT where the behavior is captured with respect to user-role assignments and activations only. As indicated above we use TIOA representation for UR_M to model temporal constraints on user-role assignments and activations. The construction of UR_M is considered in terms of parallel composition of basic UR models UR_b's, i.e., $\text{UR}_\text{M} = \text{UR}_\text{b1} \parallel_{ur} \text{UR}_\text{b2} \parallel_{ur}, \ldots, \text{UR}_\text{bk}$ where k is the total number of UR_b's. A UR_b is constructed corresponding to a user-role assignment, thus the UR_M is composed by constructing a UR_b corresponding to each user-role assignment possible in P, i.e., $k = |U||R|$.

6.1.1 UR_b Model

There could be three types of UR_b's: UR_b^1, UR_b^2 and UR_b^3, where UR_b^1 is the most general type and others are its special cases. In the subsequent discussion a UR_b, unless otherwise noted, refers to UR_b^1. UR_b models are only constructed for those user-role pairs for which P provides an explicit assignment, i.e., $(u, r) \in D_1$ (Section 3.1).

A UR_b is composed corresponding to a specific user-role pair (u, r). In a $\text{UR}_\text{b}(u, r)$, a pair of location variables is used to characterize the value of status predicates $UR_{assign}(u, r)$ and $UR_{active}(u, r)$. The A-hierarchy semantics are captured by using location variables corresponding to $UR_{active}(u, r')$ for all such

(u, r') pairs where $r' \in (R' - \{r\})$ and $R' : \{r' | r' \leq_A r\}$, is the set of all roles junior to r as per A-hierarchy semantics (r is also member of R'). We assume that P is consistent [1, 22], therefore there are no explicit user-role assignments corresponding to the (u, r') pairs as such assignments are redundant. The value of location variables in a UR_b depicts the assigned/unassigned or active/inactive status of the corresponding user-role pair.

Assumptions: In constructing a UR_b it is assumed that the deassignment and deactivation inputs (DS, DC) are only initiated by the system (ACUT) as per the requirements of rules γ_1 and γ_2, respectively. Thus a user or a system administrator does not provide these inputs before the system initiates the deassignments and deactivations automatically. For the sake of clarity the system generated deassignment and deactivation events are modeled as output actions in the TIOA model. It is further assumed that an assignment/activation input leading to a particular user-role assignment/activation is no longer available until the time the corresponding assignment/activation terminates. This is a reasonable assumption as in practice a user-role assignment/activation cannot be redone without a deassignment/deactivation first. The above assumptions can be relaxed and have been kept primarily for simplicity of presentation. Relaxation of these assumptions would increase model complexity.

We have considered the outputs to be *urgent* [28], i.e., an output action transition is traversed soon after it gets enabled. The urgency assumption is required to correctly model the preemptive termination of user-role activations envisaged by γ_1 and the temporal constraints enforcement required by rule γ_2. We next formally define UR_b as a TIOA model.

Definition 6.1 *(UR_b) The basic user-role model (UR_b) corresponding to a user-role pair (u, r) is a TIOA $UR_b(u, r) = \{L, l_0, I, O, C, T\}$ where:*

- $L = \{l_0, l_1, l_2, \ldots, l_t\}$ *is finite set of* t *locations such that* $l_s = \{UR_{assign}(u, r)$, $UR_{active}(u, r)$, $UR_{active}(u, r') | r' \in (R' - \{r\})$, $R' : \{r' | r' \leq_A r\}\}$ $1 \leq s \leq t$ *is a set of status predicates.* $l_0 = \{0, 0, \ldots, 0\}$ *is the initial location.*
- $I = \{?AS(u, r, t)$, $?AC(u, r', t) | r' \in R'$, $R' : \{r' | r' \leq_A r\}\}$ *is a finite of set of input actions.*
- $O = \{!DS(u, r)$, $!DC(u, r') | r' \in R'$, $R' : \{r' | r' \leq_A r\}\}$ *is a finite of set of output actions.*
- $C = \{x_1, x_2, \ldots, x_j\}$ *where* $j = |R'| + 1$ *is a finite set of clocks such that each clock* x_i, $1 \leq i \leq j$ *corresponds to an input action.*
- $T \subseteq L \times (I \cup O) \times \Phi(C) \times 2^C \times L$ *is a set of transitions which are defined by the application of rules* γ_1 *and* γ_2 *on the input actions as explained in algorithm ContructUR$_b$, given in Appendix A.1.*

It can be observed that UR_b satisfies the time progress requirement discussed in Section 3.2, i.e., it is *strongly non-Zeno* [29], as at least one unit of time would lapse in each loop of UR_b (the minimum value of t cannot be less than 1). Although we have considered the general case where all inputs are for temporal

L0	$UR_{assign}(u_1, r_1)=0$, $UR_{active}(u_1,r_1)=0$
L1	$UR_{assign}(u_1, r_1)=1$, $UR_{active}(u_1,r_1)=0$
L2	$UR_{assign}(u_1, r_1)=1$, $UR_{active}(u_1,r_1)=1$
L0′	$UR_{assign}(u_1, r_2)=0$, $UR_{active}(u_1,r_2)=0$
L1′	$UR_{assign}(u_1, r_2)=1$, $UR_{active}(u_1,r_2)=0$
L2′	$UR_{assign}(u_1, r_2)=1$, $UR_{active}(u_1,r_2)=1$

e_1	$?AC\ (u_1, r_1, t_2)$
e_2	$?AS\ (u_1, r_1, t_1), x_1{:=}0$
e_3	$x_1=t_1, !DS\ (u_1, r_1)$
e_4	$?AC\ (u_1, r_1, t_2), x_2{:=}0$
e_5	$x_2=t_2, !DC\ (u_1, r_1)$
e_6	$x_1=t_1, !DS\ (u_1, r_1)$

e_1'	$?AC\ (u_1, r_2, t_{2'})$
e_2'	$?AS\ (u_1, r_2, t_{1'}), x_{1'}{:=}0$
e_3'	$x_{1'}=t_{1'}, !DS\ (u_1, r_2)$
e_4'	$?AC\ (u_1, r_2, t_{2'}), x_{2'}{:=}0$
e_5'	$x_{2'}=t_{2'}, !DC\ (u_1, r_2)$
e_6'	$x_{1'}=t_{1'}, !DS\ (u_1, r_2)$

L0″	L0, L0′
L1″	L0, L1′
L2″	L0, L2′
L3″	L1, L0′
L4″	L2, L0′

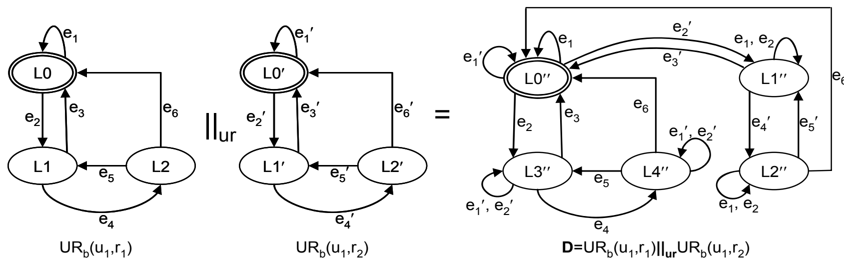


Figure 6-1: $UR_b(u_1, r_1)$ and $UR_b(u_1, r_2)$ and their parallel composition

accesses, yet non-temporal inputs can also be easily modeled using TIOA. The UR_b's for the user-role pairs (u_1, r_1) and (u_1, r_2) of Example 1 are illustrated in Figure 6-1. The proof of Lemma 6.1 which shows the correctness of UR_b construction is given in Appendix A.1.

Lemma 6.1 (Correctness of ContructUR_b): Given a P and a (u, r) pair the algorithm ContructUR_b constructs a UR_b which correctly represents the application of rules from the rule set $\Re(P)$ to each state of ACUT corresponding to the user-role assignment/activations modeled by UR_b.

As already mentioned in Section 4, the state of ACUT at a certain time is determined as the valuation of $Status$ predicates at that time. Thus each location of UR_b corresponds to a distinct state of ACUT.

A UR_b for (u, r) pair is constructed in isolation with other UR_b's by considering that other then the user-role assignments/activations specified by the $Status$ predicates corresponding to location variables of UR_b, no other user-role assignments/activations or permission-role assignments exist in ACUT. As a result not all the SoD and user/role cardinality constraint are modeled in UR_b (these constraints are fully imposed while constructing the UR_M).

6.1.2 UR_b^2 and UR_b^3 Models

The UR_b^2 and UR_b^3 models, respectively, are constructed for all user-role pairs $(u, r) \in \{(U \times R) - (D_1 \cup D_4)\}$ and $(u', r') \in D_4$. Where $D_4 = \{(u, r') \in (U \times R) - D_1 \mid (u, r) \in D_1, r' \leq_A r\}$. UR_b^2's are thus constructed corresponding to all those user-role pairs for which P does not provide information on both explicit assignment and implicit activation, whereas UR_b^3's correspond to such user-role pairs for which P does not contain explicit assignment information but does allow implicit activation.

The UR_b^2 and UR_b^3 models are special cases of UR_b^1. The UR_b^2 model corresponding to a user-role pair (u, r) is a TIOA $UR_b^2(u, r) = \{L, l_0, I, O, C, T\}$ where $L = l_0 = \{UR_{assign}(u, r), UR_{active}(u, r)\}$, $I =$

$\{?AS(u,r,t), ?AC(u,r,t)\}$, $O = C = \{\}$ and $T = \{(l, ?AS(u,r,t), -, -, l), (l, ?AC(u,r,t), -, -, l)\}$. The UR_b^3 model corresponding to (u,r) is also a TIOA defined as $UR_b^3(u,r) = \{L, l_0, I, O, C, T\}$ where $L = l_0 = \{UR_{assign}(u,r)\}$, $I = \{?AS(u,r,t)\}$, $O = C = \{\}$ and $T = \{(l, ?AS(u,r,t), -, -, l)\}$.

Lemma 6.2 (Correctness of UR_b^2 and UR_b^3): Given a P and $(u,r) \in \{(U \times R) - (D_1 \cup D_4)\}$ and (u', r') $\in D_4$ $UR_b^2(u,r)$ and $UR_b^3(u', r')$, correctly represent the application of rules from the rule set $\Re(P)$ to each state of ACUT corresponding to the user-role assignment/activations modeled by UR_b^2 and UR_b^3 respectively.

The UR_b^2 and UR_b^3 are constructed in isolation with any other UR_b's and thus it can be easily shown that they correctly represent the application of rules to each state of ACUT.

6.2 UR_M Construction

As discussed in Section 6.1, the UR_M corresponding to a P is constructed as a parallel composition of basic UR_b's of all types, i.e. $UR_M = UR_{b1} \parallel_{ur} UR_{b2} \parallel_{ur}, \ldots, UR_{bk}$ where $k = |U||R| = |UR_b^1| + |UR_b^2| + |UR_b^3|$. We next define the binary operator $\parallel_{ur}$ for parallel composition of two UR_b's. UR_M is constructed by recursive application of $\parallel_{ur}$ operator on all the UR_b's. As UR_b's are *strongly non-Zeno* therefore it can be easily shown, by using the approach similar to Lemma 3 in [29], that there parallel composition UR_M would also be *strongly non-Zeno*.

Definition 6.2 ($\parallel_{ur}$) *Given two UR_b's, $A = \{L, l_0, I, O, C, T\}$ and $B = \{L', l'_0, I',$ $O', C', T'\}$ where $C \cap C' = \emptyset$, $I \cap I' = \emptyset$ and $O \cap O' = \emptyset$, the parallel composition $D = A \parallel_{ur} B$ is defined as $D = \{L_D \subseteq L \times L'$, (l_0, l'_0), $I \cup I'$, $O \cup O'$, $C \cup C'$, $T_D\}$ such that L_D and T_D are the smallest relations defined by the application of rules γ_1 and γ_2 on the input actions as explained in algorithm ParallelComposition in Appendix A.2.*

The parallel composition $D = A \parallel_{ur} B$ is constructed by a constrained Cartesian product of the locations of A and B and the union of their clocks, inputs, and outputs. The locations L_D and transitions T_D of D are determined by the algorithm ParallelComposition. The total number of locations of D is less than or equal to the Cartesian product of L and L', i.e., $|L_D| \leq |L \times L'|$.

A TIOA constructed by parallel composition of two UR_b's can also be considered as another UR_b. The parallel composition $D = UR_b(u_1, r_1) \parallel_{ur} UR_b(u_1, r_2)$ corresponding to P of Example 1 is illustrated in Figure 6-1. Lemma 6.3 formally shows (proof given in Appendix A.2) that the parallel composition $D = A \parallel_{ur} B$ for the UR_b's A and B correctly models ACUT behavior with respect to the user-role assignments/activations modeled by A and B. D is constructed from the UR_b's A and B in isolation with all other UR_b's, i.e., the impact of other user-role assignments/activations in the ACUT is not considered in constructing D. The recursive application of $\parallel_{ur}$ ensures that the final UR_M correctly models ACUT with respect to all the user-role assignments/activations.

Lemma 6.3 (Correctness of ParallelComposition): Given a P and two UR_b's A and B the algorithm ParallelComposition constructs $D = A \parallel_{ur} B$ which correctly represents the application of rules from the rule set $\Re(P)$ to each state of ACUT corresponding to the user-role assignments/activations modeled by A and B.

Corollary 6.1 (Correctness of UR_M): UR_M correctly represents the application of rules from the rule set $\Re(P)$ to each state of ACUT with respect to all the user-role assignments/activations.

The proof is simply based on the correctness of ParallelComposition shown by Lemma 6.3, as UR_M is constructed by recursive application of ParallelComposition on all the UR_b's.

6.3 PR Model

Permission-role model (PR_M) encapsulates the behavior of an ACUT with respect to permission-role assignments specified explicitly in P or implicitly allowed via I-hierarchy semantics. PR_M is constructed in a way similar to UR_M by considering parallel composition of basic permission-role models (PR_b's), i.e. $\text{PR}_\text{M} = \text{PR}_\text{b1} \parallel_{pr} \text{PR}_\text{b2} \parallel_{pr}, \ldots, \text{PR}_\text{bj}$ where $j = |D_6| + |D_3|$ is the total number of PR_b's, such that $D_6 = \{(p,r) \in (P \times R) | (p,r) \notin (D_3 \cup D_5)\}$ and $D_5 = \{(p,r') \in (P \times R) - D_3 \, | (p,r) \in D_3, r \leq_I r'\}$. Set $(D_3 \cup D_5)$ corresponds to all such permission-role pairs for which assignments are either explicitly specified in P or implicitly permitted via I-hierarchy semantics, whereas the set D_6 represents such permission-role pairs for which P does not contain any assignment information.

6.3.1 PR_b Model

There could be two types of PR_b's: PR_b^1 and PR_b^2, where PR_b^1 is the most general type. In the subsequent discussion, unless otherwise noted, PR_b refers to PR_b^1. A PR_b is composed corresponding to a specific permission-role pair (p,r). In the absence of roles senior to r by virtue of I-hierarchy semantics, PR_b would be very simple as only one location variable is used to characterize the value of status predicate $PR_{assign}(p,r)$. The I-hierarchy semantics are captured by using location variables corresponding to $PR_{assign}(p,r')$ for all such (p,r') pairs where $r' \in (R' - \{r\})$ and $R' : \{r' | r \leq_I r'\}$, is the set of all roles senior to r as per I-hierarchy semantics (r is also a member of R'). The value of location variables in PR_b therefore depicts the assigned/unassigned status of the corresponding permission-role pair. Next we formally define PR_b as a TIOA model.

Definition 6.3 *(PR_b) The basic permission-role model (PR_b) corresponding to a permission-role pair (p,r) is a TIOA $\text{PR}_\text{b}(p,r) = \{L, l_0, I, O, C, T\}$ where:*

- $L = \{l_0, l_1\}$ *is a set of two locations such that* $l_s = \{PR_{assign}(p,r') | r' \in R' \, , R' : \{r' | r \leq_I r'\}\}$ $s \in \{0,1\}$*, is a set of status predicates.* $l_0 = \{0, 0, \ldots, 0\}$ *is the initial location.*

- *$I = \{?AP(p, r', t) | r' \in R'\ ,\ R' : \{r' | r \leq_I r'\}\}$ is a finite of set of input actions.*

- *$O = \{\wedge(!DP(p, r') | r' \in R'\ ,\ R' : \{r' | r \leq_I r'\})\}$ is a set of single output action.*

- *$C = \{x_1\}$ is a set of single clock x_1 which corresponds to the input action $?AP(p, r, t)$.*

- *$T \subseteq L \times (I \cup O) \times \Phi(C) \times 2^C \times L$ is a set of transitions, defined by the application of rule γ_3 on inputs, as explained in the algorithm ContructPR$_b$, given in Appendix A.3.*

While constructing a PR$_b$ it is assumed that the deassignment input is only initiated by the system (ACUT) as per the requirements of rule γ_3 and thus a system administrator does not provide these inputs before the system automatically initiates the deassignments. PR$_b$ also satisfies the time progress requirement as it is *strongly non-Zeno*. The proof of Lemma 6.4 which shows the correctness of PR$_b$ construction is in Appendix A.3.

Lemma 6.4 (Correctness of ContructPR$_b$): Given P and (p, r), ConstructPR$_b$ constructs a PR$_b$ that correctly represents the application of rules from the rule set $\Re(P)$ to each state of ACUT corresponding to the permission-role assignments modeled by PR$_b$.

PR$_b^2$ models are constructed for all the permission-role pairs $(p, r) \in D_6$ for which P does not provide both the explicit and implicit assignment information. A PR$_b^2$ model corresponding to a permission-role pair (p, r), being a special case of PR$_b^1$, is a TIOA PR$_b^2(p, r) = \{L, l_0, I, O, C, T\}$ where $L = l_0 = \{PR_{assign}(p, r)\}$, $I = \{?AP(p, r, t)\}$, $O = C = \{\}$ and $T = \{(l, ?AP(p, r, t), -, -, l)\}$. The correctness of PR$_b^2$ is simple to observe.

6.4 PR$_M$ Construction

As already mentioned, PR$_M$ is obtained as parallel composition of PR$_b$'s of both types, i.e., PR$_M$ = PR$_{b1}$ $\|_{pr}$ PR$_{b2}$ $\|_{pr}, \ldots,$ PR$_{bj}$ where $j = |D_6| + |D_3| = |\text{PR}_b^1| + |\text{PR}_b^2|$. The binary operator $\|_{pr}$ is defined next for parallel composition of two PR$_b$'s. PR$_M$ is constructed by recursive application of $\|_{pr}$ operator on all the PR$_b$'s. As PR$_b$'s are *strongly non-Zeno* therefore their parallel composition PR$_M$ would also be *strongly non-Zeno* (Lemma 3 in [29]).

Definition 6.4 ($\|_{\mathbf{pr}}$) *Given two PR$_b$'s, $A = \{L, l_0, I, O, C, T\}$ and $B = \{L', l'_0, I', O', C', T'\}$ where $C \cap C' = \emptyset$, $I \cap I' = \emptyset$ and $O \cap O' = \emptyset$, the parallel composition $D = A \|_{pr} B$ is defined as $D = \{L \times L', (l_0, l'_0), I \cup I', O \cup O', C \cup C', T_D\}$ such that T_D is the minimum set of composite transitions formed by interleaving of individual transitions of A and B. Corresponding to a transition pair (e_1, e_2) where $e_1 = (l_s, \{?i, !o\}, g, R, l_t) \in T$ and $e_2 = (l'_s, \{?i, !o\}', g', R', l'_t) \in T'$, two composite transitions, $e'_1 = ((l_s, l'_s), \{?i, !o\}, g, R, (l_t, l'_s))$ and $e'_2 = ((l_s, l'_s), \{?i, !o\}', g', R', (l_s, l'_t))$, will be added to T_D by virtue of interleaving of e_1 and e_2.*

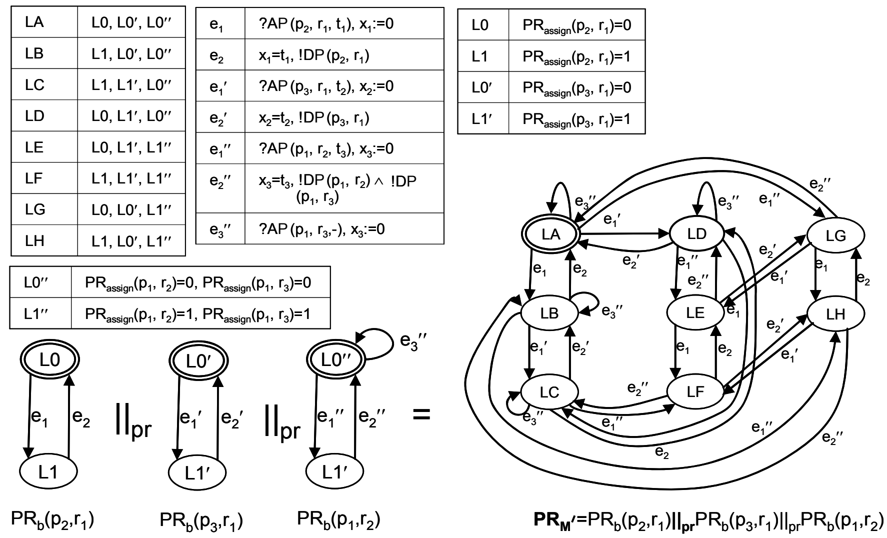


Figure 6-2: Example of parallel composition of three PR$_b$'s

The correctness of $\|_{pr}$ comes easily from Definition 6.4. As a corollary, it is simple to argue the correctness of PR$_M$. An example of parallel composition of three PR$_b$'s is illustrated in Figure 6-2. The P in Figure 6-2 corresponds to Example 1 with an added role `ResidentDoctor` (r_3) senior to r_2 in the I-hierarchy. Assume there are three permissions p_1 =`Update patient record`, p_2 =`Erase patient record`, and p_3 =`Create patient record` which can be assigned to roles r_1 and r_2. Consider that P specifies that p_2 and p_3 can be assigned to r_1, whereas only p_1 can be assigned to r_2. There would be three PR$_b^1$ models tallying with p_2r_1, p_3r_1, and p_1r_2 explicit permission-role assignments in P. PR$_b(p_1, r_2)$ illustrates that, by virtue of I-hierarchy, p_1r_2 assignment would also automatically lead to p_1r_3 assignment without requiring any other input. The PR$_{M'}$ constructed as a parallel composition PR$_b(p_2, r_1)$ $\|_{pr}$ PR$_b(p_3, r_1)$ $\|_{pr}$ PR$_b(p_1, r_2)$ is also shown in Figure 6-2. Note that PR$_M$ would be constructed by parallel composition of PR$_{M'}$ with all PR$_b^2$ models, i.e. PR$_M$ $=$ PR$_{M'}$ $\|_{pr}$ PR$_b^2(p_1, r_1)$ $\|_{pr}$ $\ldots$ $\|_{pr}$ PR$_b^2(p_3, r_3)$.

Theorem 6.1 formally shows the correctness of TRBAC$_M$ $=$ UR$_M$ $\|$ PR$_M$.

Theorem 6.1 (Correctness of TRBAC$_M$ $=$ UR$_M$ $\|$ PR$_M$): TRBAC$_M$ correctly represents the application of rules from the rule set $\Re(P)$ to each state of ACUT with respect to all the user-role assignments and activations and permission-role assignments in P.

Proof of Theorem 6.1 is based on the correctness of UR$_M$ and PR$_M$ shown earlier, as the parallel composition UR$_M$ $\|$ PR$_M$ does not cause any violation of the rules from the rule set $\Re(TRBAC_P)$.

We have already presented the TRBAC fault model in Section 5 and have now completed a description of the technique for the construction of TRBAC$_M$ for a given TRBAC policy specification. In the next section we focus on a procedure for the generation of the conformance test suite from TRBAC$_M$ that provides complete coverage with respect to the faults in the proposed TRBAC fault model.

7 Test Generation from TRBAC Model (TRBAC_{M})

Key steps in the proposed technique for construction of conformance test suite from TRBAC_{M} are enumerated below and explained subsequently.

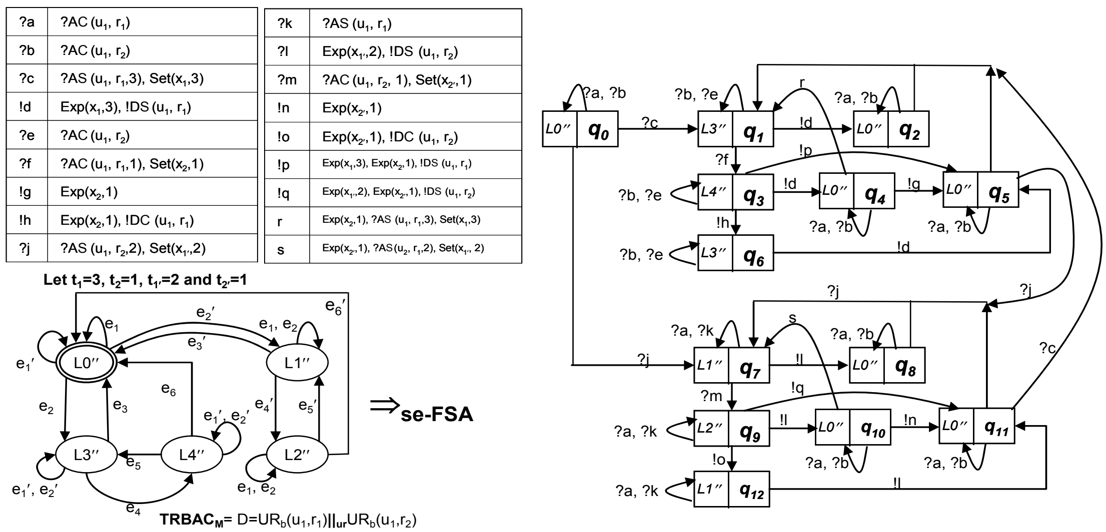
1. Transform TRBAC_{M} into se-FSA (se-TRBAC_{M}) by adopting the procedure given in [18].
2. Construct the test tree (Tr) corresponding to the se-TRBAC_{M}.
3. Generate the conformance test suite. The conformance test suite is then executed against the ACUT using the test architecture proposed in [17].

7.1 Transformation of TRBAC_{M} into se-TRBAC_{M}

It is important to note that the semantic graph S_A of a TIOA A (Section 3.2), which encapsulates the information about all the accepting runs of A, is of infinite size. The primary purpose of se-FSA transformation is to capture the timed semantics of A by using a Finite State Automaton (FSA). The se-FSA transformation converts a TIOA into an *equivalent* finite state automaton which in addition to the events of the TIOA has the two special types of events: *Set* and *Exp* that model setting and expiring of clocks, respectively. The se-FSA is *equivalent* to its corresponding TIOA, as shown in [18], in the sense that both specify the same order and timing constraints of events. We omit the finer details of se-FSA transformation and refer the interested to [18] for the complete algorithm.

A se-FSA is an FSM $S_E = (Q, q_0, X, Y, \delta, O)$ where Q is a finite set of states, $q_0 \in Q$ a unique initial state, X and Y, respectively, the input and output alphabets, $\delta : Q \times X \rightarrow Q$ the state transition function, and $O : Q \times X \rightarrow Y$ the output function. A member such as Q of S_E is referred as $Q(S_E)$. The salient features of se-FSA transformation of TRBAC_{M} are illustrated by considering a simpler version of Example 1 with the single user u_1. For this case $\text{TRBAC}_{\text{M}} = \text{UR}_{\text{M}} = D$. We have adapted the se-FSA procedure of [18] to handle urgent outputs in TRBAC_{M}. In general, the se-FSA transformation of a deterministic TIOA may result in a non-deterministic FSM, however, the se-FSA corresponding to TRBAC_{M} would always be deterministic because TRBAC_{M} considers urgent outputs and any clock resets in it are only associated with input actions.

The se-FSA transformation of TRBAC_{M}, i.e., se-TRBAC_{M} of simplified Example 1, is illustrated in Figure 7-1. It can be observed that there are three types of events in se-TRBAC_{M}: input events corresponding to input actions and/or clock resets in TRBAC_{M}, output events corresponding to output actions in TRBAC_{M} and/or clock expirations, and complex events occurring as combination of the previous two. The duration constraints specified in TRBAC_{M} become explicit in the se-TRBAC_{M} as all the input actions,



?a	?AC (u_1, r_1)	?k	?AS (u_1, r_1)
?b	?AC (u_1, r_2)	?l	Exp$(x_{1'},2)$, !DS (u_1, r_2)
?c	?AS $(u_1, r_1,3)$, Set$(x_1,3)$	?m	?AC $(u_1, r_2, 1)$, Set$(x_{2'},1)$
!d	Exp$(x_1,3)$, !DS (u_1, r_1)	!n	Exp$(x_{2'},1)$
?e	?AC (u_1, r_2)	!o	Exp$(x_{2'},1)$, !DC (u_1, r_2)
?f	?AC $(u_1, r_1,1)$, Set$(x_2,1)$	!p	Exp$(x_1,3)$, Exp$(x_2,1)$, !DS (u_1, r_1)
!g	Exp$(x_2,1)$	!q	Exp$(x_{1'},2)$, Exp$(x_{2'},1)$, !DS (u_1, r_2)
!h	Exp$(x_2,1)$, !DC (u_1, r_1)	r	Exp$(x_2,1)$, ?AS $(u_1, r_1,3)$, Set$(x_1,3)$
?j	?AS $(u_1, r_2,2)$, Set$(x_{1'},2)$	s	Exp$(x_{2'},1)$, ?AS $(u_2, r_1,2)$, Set$(x_{1'}, 2)$

Figure 7-1: se-FSA transformation of D i.e se-TRBAC$_M$

associated with transitions other than self loops, would cause setting of a clock whose expiry will result in the corresponding deactivation/deassignment.

7.2 Construction of Test Tree

As mentioned, se-TRBAC$_M$ is deterministic, i.e., no two edges out of a state have the same labels. In addition, it has a finite number of states. Therefore we can use any test generation technique for deterministic FSA's to construct the conformance test suite. For our test generation for TRBAC ACUT we used the W-method [9] because of its proven fault coverage. Tests are generated in this method by concatenating the test sequences obtained from the test tree (Tr) with the determined state characterization set referred to as the W set.

We assume the existence of reliable methods in the ACUT that can be used to directly query the current state. Hence the W set is not required and the test set can be generated directly from the testing tree. This assumption is not very restrictive and has also been used in the Binder round trip method [7] for class testing in object-oriented programs. By virtue of state observability, the test suite is directly constructed from Tr. When the states are not observable then, using the procedure given in [17], the se-TRBAC$_M$ can be made input complete and W set can be determined. The test suite generation in this case would be more complex and the test suite size would be much larger.

Note that state observability here implies the ability to determine the valuation of $Status$ predicates in the ACUT, which actually corresponds to location observability. In se-TRBAC$_M$ this could cause a problem in detecting such transfer, missing state, and extra state faults where the correct and incorrect destination states correspond to the same location but different range of clock variables. However, as discussed later in Section 7.5, by virtue of the structure of TRBAC$_M$ and its se-TRBAC$_M$ transformation, transfer, missing state, and extra state faults leading to incorrect states corresponding to same location as of the correct state

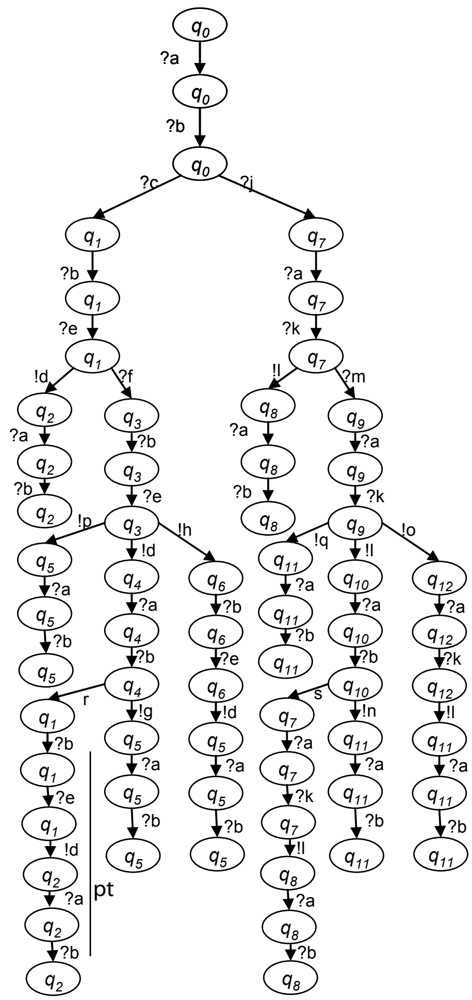


Figure 7-2: Tr for se-TRBAC$_{\text{M}}$ of Figure 7-1

would always lead to output faults which can be detected with only location observability.

While constructing the Tr from the FSM if the added node at depth k is the same as some other node at depth $i, i \leq k$, then that node is terminated with no further edge out of it [9]. For our conformance test suite we considered a modified Tr to reduce the total number of test sequences by including the repeated nodes explicitly in the path only when they are encountered for the first time (corresponding to a specific event). Note that the paths in Tr still form a *transition cover set* as required by the W-method [9]. A Tr corresponding to se-TRBAC$_{\text{M}}$ of Figure 7-1 is illustrated in Figure 7-2.

As the finite sequences accepted by an se-FSA should terminate in a state without outgoing *Exp* events (accepting state) [18], the terminals in a test tree have to be from among the set of accepting states. Hence, if at any level in the Tr a non-accepting node is repeated, the corresponding path is not terminated untill the time the terminal node is an accepting state. We suggest that the shortest path among all paths from a repeating non-accepting node to any of the accepting states be used for this path augmentation. The path "pt" in the Tr of Figure 7-2, depicts that although q_1 is repeated i for the same events "?b" and "?e" yet the path is not terminated because q_1 is not an accepting state.

7.3 Generation of Conformance Test Suite (CTS)

The se-TRBAC$_M$ allows us to use the test system given in Figure 7-3 for conformance testing of the given ACUT. This test system has been first proposed in [17] (there is a minor variation in our test system as the state queries are used to get the current state information from the ACUT). The purpose of the Test-Controller is to control the execution of all the test sequences of the conformance test suite. It sends inputs to the ACUT and *Set* events to the Clock-Handler at the specified times and receives the outputs from the ACUT and *Exp* events from the Clock-Handler. After each input or output event the Test-Controller queries the ACUT to determine the current state. A test sequence is considered passed if all the outputs from the ACUT match the corresponding *Exp* event and the state information agrees with the one required by the test sequence. An ACUT is thus considered conforming if all the tests pass.

Each path in Tr represents a unique test sequence. For a given path p_t in Tr, the test sequence is constructed by associating all the edges $e \in p_t$ with monotonically increasing time stamps. Constraints are imposed on the time stamps by virtue of the semantics of se-TRBAC$_M$. The time stamp associated with an input action indicate the time at which Test-Controller should generate the corresponding input for the ACUT and the Clock-Handler. An ACUT will pass the subject sequence if outputs are generated by the ACUT and the Clock-Handler at times corresponding to the time stamps of output actions.

To illustrate the semantics of a test sequence and the procedure used for determining the feasible value of time stamps so that the sequence can be executed by the test system on the ACUT, we consider as an example the following sequence obtained from the Tr of Figure 7-2:

$TS_1 = q_0$, $(?a, t_1)q_0$, $(?b, t_2)q_0$, $(?c, t_3)q_1$, $(?b, t_4)q_1$, $(?e, t_5)q_1$, $(?f, t_6)q_3$, $(?b, t_7)q_3$, $(?e, t_8)q_3$, $(!d, t_9)q_4$, $(?a, t_{10})q_4$, $(?b, t_{11})q_4$, $(!g, t_{12})q_5$, $(?a, t_{13})q_5$, $(?b, t_{14})q_5$ with the temporal constraints, $t_1 = 0$, $t_{i+1} > t_i$, $t_9 - t_3 = 3$ and $t_{12} - t_6 = 1$.

TS_1 corresponds to the execution sequence of ACUT where the u_1r_1 activation is pre-empted by the deassignment output. The constraints $t_9 - t_3 = 3$ and $t_{12} - t_6 = 1$ represent the time difference between the matching *Set* and *Exp* events in this sequence. The feasible value of time stamps should satisfy the required temporal constraints, which can be represented as:

$dt_2 = t_2, dt_i > 0, 2 \leq i \leq 12$, $dt_4 + dt_5 + dt_6 + dt_7 + dt_8 + dt_9 = 3$ and $dt_7 + dt_8 + dt_9 + dt_{10} + dt_{11} + dt_{12} = 1$ where $dt_i = t_i - t_{i-1}$.

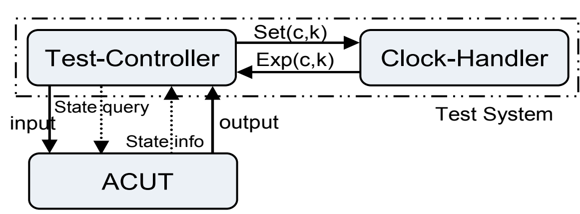


Figure 7-3: Structure of Test System (Test harness) [17]

This problem can be treated as a linear program by considering that the objective function $\sum_i dt_i$ is minimized subject to the given constraints. As we are dealing with dense time semantics, the obtained solution can have very minute resolution (the smallest value among all the dt_i's), thus the execution of the given sequence might not be practically possible. We overcome this problem by specifying the minimum resolution and formulating the feasible time stamp determination problem as an Integer Program (IP) [30]. The integer program corresponding to the above problem will be:

$\min \sum_i dt_i$, subject to $dt_i/k = c_i, 2 \leq i \leq 12$, $dt_4 + dt_5 + dt_6 + dt_7 + dt_8 + dt_9 = 3$, $dt_7 + dt_8 + dt_9 + dt_{10} + dt_{11} + dt_{12} = 1$ and $c_i \in Z^+$, where k specifies the minimum resolution. Solving the IP for $k = 0.1$, the *test sequence* would be:

q_0, $(?a, 0)q_0$, $(?b, 0.1)q_0$, $(?c, 0.2)q_1$, $(?b, 2.3)q_1$, $(?e, 2.4)q_1$, $(?f, 2.5)q_3$, $(?b, 3)q_3$, $(?e, 3.1)q_3$, $(!d, 3.2)q_4$, $(?a, 3.3)q_4$, $(?b, 3.4)q_4$, $(!g, 3.5)q_5$, $(?a, 3.6)q_5$, $(?b, 3.7)q_5$.

When no feasible solution exists for a specific value of k, we can continue reducing the value of k untill the time a solution is obtained. The CTS is thus obtained by determining the feasible time stamps for all the test sequences, where as mentioned above, each sequence corresponds to a unique path in the Tr.

7.4 Relation between TRBAC, TIOA, and se-FSA Fault Models

The TIOA based fault model considered in [11] comprises of two types of faults: timing faults and "action (output) and transfer". We considered an extended TIOA fault model with missing location and extra location faults, not considered in [11] because of the test hypothesis considered there. The action, transfer, missing and extra location faults in TIOA are similar to the output (operation), transfer, missing and extra state faults in finite state machines [9].

There could be three types of timing faults in an ACUT′ [11], (1) reset of a clock fault, (2) time constraint restriction fault, and (3) time constraint widening fault. An ACUT′ would have a clock reset fault if it does not reset the expected clocks or resets wrong clocks not stated by the specification. We consider the other two types of faults specifically with respect to our TIOA TRBAC_{M} model in which guards only restrict the timings of output transitions through equality constraints on the clock values. An ACUT′ would have a time constraint restriction fault (time constraint widening fault) if a constraint $x = t'$ is replaced by $x = t$ such that $t' < t (t' > t)$.

The relation between the TRBAC faults, described in Section 5, and the TIOA faults in the TRBAC_{M} model is illustrated in Table 3. As all the TRBAC faults can be associated with some fault type in the TRBAC_{M} model, a CTS capable of detecting all TRBAC_{M} faults would automatically guarantee complete fault coverage for TRBAC faults.

The se-FSA based fault model considered in [17] consists of only the output and transfer faults. We extend this fault model through inclusion of missing state and extra state faults. The output, transfer, missing location and extra location faults in the TRBAC_{M} model have similar representation in the se-TRBAC_{M}. The time constraint restriction and widening faults are represented in the form of combination of output and

Table 3: Relation between TRBAC and TIOA Faults

TRBAC Faults	**TIOA Faults**
UR1, UA1, PR1	Transfer, Missing location, Output
UR2, UA2, PR2	Extra location, Output, Transfer
Hierarchical enforcement	Transfer, Output
Duration restriction	Time constraint restriction, clock reset
Duration widening	Time constraint widening, clock reset

transfer faults. However, a clock reset fault does not have any direct correspondence with the faults in the se-TRBAC$_M$ model as the se-FSA model always considers pairs of *Set* and related *Exp* events and thus a missing or extra *Set* event carries no semantics in the se-TRBAC$_M$ model.

7.5 Fault Coverage of CTS

CTS is generated by applying the W-method on se-TRBAC$_M$, where the W-method provides complete fault coverage for output, transfer, missing state and extra state faults under the assumption that the number of states in the ACUT are accurately estimated [9]. In Section 7.2 we highlighted the issue of detecting such transfer, missing state and extra state faults which lead to incorrect states with same location as of correct states but a differing range of clock variables. Note that states $q \in Q(S_E)$ of an se-FSA contain information corresponding to both locations of the source TIOA and the range of clock variables [18], referred here as ΔC. Consider that ACUT is the correct implementation corresponding to se-TRBAC$_M$= S_E, and there is a faulty ACUT′ which implements $S_{E'}$, where S_E and $S_{E'}$ differ only in $\delta(S_E)$ in case of a transfer fault in the ACUT′ and in both $Q(S_E)$ and/or $\delta(S_E)$ and $O(S_E)$ in case of a extra or missing state fault in the ACUT′. Consider the execution of ACUT′ against test sequence $TS \in$ CTS where transition $\tau = (q_i, (i, t)q) \in TS$ corresponds to fault $f = (q_i, (i', t)q'), q' \neq q$ in ACUT′ such that q and q' only differ in ΔC. Consider that TS corresponds to the path p_t in Tr.

Note that ΔC would only vary in q' from q if i' and i differ in Set or Exp or both events. If i and i' differ in Set events, which corresponds to clock reset fault, then as discussed next such faults would be detected by the CTS as output faults. If the difference between the two is only in Exp events then we separately consider the cases of missing, extra or modified Exp events in i' in relation with the semantics of se-FSA. A missing Exp event would only occur if there is a missing clock reset fault earlier in the path p_t before the edge corresponding to τ. An extra Exp event would similarly correspond to incorrect clock reset. A modified Exp would occur because of either a missing or incorrect clock reset fault or combination of two faults and as discussed later the clock reset faults would be detected as output faults. From this discussion it is simple to observe that if i' differs from i in both Set and Exp events then this would also correspond to clock reset faults.

From the above discussion we infer that if an ACUT′ has transfer, missing state and extra state faults that lead to incorrect states with same location as of correct states but differing range of clock variables, then such faults would always occur in combination with output faults which would then be always detected by at least one element in CTS. Thus if an ACUT′ passes the CTS then it would be free of all the output, transfer, missing and extra state faults. The correlation between TIOA and se-FSA faults, established in Section 7.4, implies that CTS would be able to detect all the output, transfer, missing and extra location, and time constraint restriction and widening faults in the TRBAC_{M}. We claim that the clock reset faults would also be detected by the CTS.

Next consider a missing clock reset fault in the ACUT′. Note that the CTS includes at least one such test sequence which contains both the *Set* event associated with the missing reset and an output event containing the corresponding *Exp* event. As this sequence is executed against the faulty ACUT′, the time of occurrence of the output event and the *Exp* event would not be the same and thus the missing clock reset fault would be detected by at least one element of CTS as an output fault. By using a similar approach it can be easily shown that the reset of incorrect clock fault would also be detected by the CTS. It is important to note that an incorrect clock reset fault would only alter the semantics of the TIOA model, in case if the corresponding clock is used in some guard subsequently (before its correct reset) in the semantic graph of TIOA.

Proposition 7.1 CTS detects all transfer, output, missing and extra location, and timing faults in the TRBAC_{M}, and hence it must detect all TRBAC faults in a ACUT′ given that there are no faults in the ACUT′ because of user/system-administrator initiated deassignment/deactivation requests.

The proof of Proposition 7.1 is based on the fact that CTS is able to detect all the faults in se-TRBAC_{M}, the correlation between fault models established in Section 7.4, and the correctness of TRBAC_{M} established by Theorem 6.1.

To illustrate the fault coverage of CTS, consider that corresponding to the ACUT which correctly enforces simplified P (Section 7.1) of Example 1, there are two faulty ACUT's: ACUT′ and ACUT″ which, respectively, enforce policies $TRBAC_{P'}$ and $TRBAC_{P''}$. $TRBAC_{P'} = (U, R, Pr, \dots, S_s, D_s, \Re^1)$ and $TRBAC_{P''} = (U, R, Pr, \dots, S_s, D_s, \Re^2)$ where $\Re^1$ differs from $\Re$ in only γ_{urSSoD} such that $\forall (u, r) \in U \times R$ $\gamma_{urSSoD}(u, r) = 1$. $\Re^2$ differs from $\Re$ in γ_1 in enforcing the duration constraint on $AS(u, r, t)$ by increasing the duration to $t + 1$ in the $TRBAC_{P''}$.

Table 4 records the results of executing ACUT and the two faulty ACUTs against TS_1. The various notations used in Table 4 correspond to: I_s = initial state, N_s = next state, (i, t) = (input,time) and (o, t) = (output,time). The error in $TRBAC_{P'}$ would lead to a UR2 fault as despite the existence of static SoD constraint on simultaneous u_1 assignment to r_1 and to r_2, the concurrent $u_1 r_1$ and $u_1 r_2$ assignments would exist in the ACUT′. The error in $TRBAC_{P''}$ would cause a duration widening fault as the duration of $u_1 r_1$ assignment would be inappropriately extended to 4 tu's, against 3 tu's specified by P. As indicated by

Table 4: Comparison of TS_1 execution on Conforming and faulty ACUT's

<table>
<tr><th>TS_1</th><th>ACUT</th><th>ACUT$'$</th><th>ACUT$''$</th></tr>
<tr><td>I_s</td><td>q_0</td><td>q_0</td><td>q_0</td></tr>
<tr><td>i, t</td><td>$?a, 0$</td><td>$?a, 0$</td><td>$?a, 0$</td></tr>
<tr><td>N_s</td><td>q_0</td><td>q_0</td><td>q_0</td></tr>
<tr><td>i, t</td><td>$?b, 0.1$</td><td>$?b, 0.1$</td><td>$?b, 0.1$</td></tr>
<tr><td>N_s</td><td>q_0</td><td>q_0</td><td>q_0</td></tr>
<tr><td>i, t</td><td>$?c, 0.2$</td><td>$?c, 0.2$</td><td>$?c, 0.2$</td></tr>
<tr><td>N_s</td><td>q_1</td><td>q_1</td><td>q_1</td></tr>
<tr><td>i, t</td><td>$?b, 2.3$</td><td>$?b, 2.3$</td><td>$?b, 2.3$</td></tr>
<tr><td>N_s</td><td>q_1</td><td>q_1</td><td>q_1</td></tr>
<tr><td>i, t</td><td>$?e, 2.4$</td><td>$?e, 2.4$</td><td>$?e, 2.4$</td></tr>
<tr><td>N_s</td><td>q_1</td><td>q_1'</td><td>q_1</td></tr>
<tr><td>i, t</td><td>$?f, 2.5$</td><td rowspan="8">q_1' differs from q_1 in the value of UR_{assign} (u_1, r_2) which is 1 in the former and 0 in the later</td><td>$?f, 2.5$</td></tr>
<tr><td>N_s</td><td>q_3</td><td>q_3</td></tr>
<tr><td>i, t</td><td>$?b, 3$</td><td>$?b, 3$</td></tr>
<tr><td>N_s</td><td>q_3</td><td>q_3</td></tr>
<tr><td>i, t</td><td>$?e, 3.1$</td><td>$?e, 3.1$</td></tr>
<tr><td>N_s</td><td>q_3</td><td>q_3</td></tr>
<tr><td>o, t</td><td>$!d, 3.2$</td><td>$\mathbf{E}xp(x_1), 3.2$</td></tr>
<tr><td>N_s</td><td>q_4</td><td>$\mathbf{q}_3$</td></tr>
</table>

Table 4, the UR2 fault in $TRBAC_{P'}$ and the duration widening fault in $TRBAC_{P''}$ would be detected by TS_1 as extra state and output/transfer faults respectively.

8 Heuristics for CTS Reduction

Though promising, the test generation approach based on construction of CTS from TRBAC_M presented in Section 7.3 can be expensive and thus impractical in terms of the size of the model required to capture the ACUT behavior and the size of the corresponding CTS. We propose two heuristics, labeled HT1 and HT2, to reduce the size of the model and of the corresponding CTS.

8.1 Heuristic HT1

This heuristic considers a reduced TRBAC_M, referred to as $\text{TRBAC}_{\text{M}'}$. The size of $\text{TRBAC}_{\text{M}'} = \text{UR}_{\text{M}'} \parallel \text{PR}_{\text{M}'}$ is reduced by considering fewer number of UR_b's and PR_b's, respectively, in the construction of $\text{UR}_{\text{M}'}$ and $\text{PR}_{\text{M}'}$ as compared to UR_M and PR_M. For $\text{UR}_{\text{M}'}$, the number of UR_b's is reduced by considering UR_b's corresponding to only those user-role pairs for which explicit assignment is provided by P, i.e.

$\forall(u,r) \in D_1$, hence $k = |D_1|$. Thus in case of Example 1 although the total number of possible user-role assignments is four, i.e., assignments consequent to u_1r_1, u_1r_2, u_2r_1 and u_2r_2 pairs; however, only three UR_b's are considered in constructing $UR_{M'}$ because u_2r_1 assignment is not explicitly stated by P.

By using the proposed strategy for reducing the number of UR_b's , the size of the resultant $UR_{M'}$ can be significantly trimmed. However, this trimming is at the expense of reduced fault detection effectiveness of CTS. Specifically, $UR_{M'}$ does not encapsulate sufficient information which can reveal all the UR and UA faults in the ACUT′ (even under the assumption that the events corresponding to user-role assignments/activations and permission-role assignments can be considered as independent). To lessen the impact of this shortcoming, in addition to the tests generated from $TRBAC_{M'}$, we suggest separate validation of all such user-role assignments and activations not captured by $UR_{M'}$.

The separate validation is performed by verifying that for the given user-role pairs, corresponding to the application of inputs AS and AC, the ACUT′ response matches the one permitted by $TRBAC_P$. Note that this validation does not guarantee absence of all the UR and UA faults in the ACUT′ as there could be faults that are only depicted during a specific sequence of events. As an example consider a UA2 fault that leads to u_2r_1 assignment in the ACUT′ corresponding to Example 1. If this fault is only visible after the occurrence of a u_2r_2 activation, i.e., when $UR_{active}(u_2, r_2) = 1$, then it cannot be revealed by such validation.

Similarly, the reduction in the size of $PR_{M'}$ is achieved by constructing the PR_b's for only those permission-role pairs for which explicit assignment is specified by the $TRBAC_P$, i.e. $\forall(p,r) \in D_3$, hence $j = |D_3|$. This will likely lead to reduced fault detection effectiveness of the CTS. Specifically, $PR_{M'}$ does not capture enough information to reveal all the PR faults in an ACUT′ (even under the assumption that the events corresponding to user-role assignments/activations and permission-role assignments can be considered as independent). Therefore, we again suggest separate validation of all such permission-role assignments which are not captured by $PR_{M'}$. The validation of all such permission-role pairs is performed by applying corresponding AP inputs to the ACUT′ and comparing its response with the one specified by the $TRBAC_P$. As before, such validation does not guarantee absence of all the PR faults in the ACUT′ as there could be such faults which are only depicted during a specific sequence of inputs.

As an example, consider the $PR_{M'}$ illustrated in Figure 6-2. Consider a PA2 fault in the ACUT′ that leads to p_2 assignment to r_2 only after the occurrence of a p_2r_1 assignment, i.e., when $PR_{assign}(p_2, r_1) = 1$ becomes true. This fault cannot be revealed by the separate validation.

8.2 Heuristic HT2

In this heuristic, the size of the CTS is reduced by generating it independently from the UR_M and PR_M models. As UR_M and PR_M models do not capture the complete ACUT behavior specified by P, therefore unless it is possible to assume that all the events in UR_M and PR_M are independent, complete fault coverage cannot be guaranteed. Note that by abstracting the details captured by a location in the TIOA based TRBAC model, various other heuristics can be designed such as constructing separate TIOA models for each user and each role.

9 Related Work

Though significant amount of research has been reported in relation to modeling [3, 4, 13, 26, 31] and test generation for real-time systems [8, 11, 20, 19, 17], we are not aware of any work that addresses the problem of test generation for access control systems employing policies with temporal constraints.

We have used timed automata [3, 13], in particular Timed Input Output Automaton (TIOA) to model real-time constraints in a TRBAC specification. Although there exist other formalisms such as timed Petri nets, timed process algebras, and real time logics [4, 26, 31], which can be used for specifying real-time systems, we considered timed automata in modeling TRBAC because it allows us to leverage the significant amount of research on test generation from timed automata [8, 11, 20, 19, 17, 21, 28].

The proposed test generation procedure (Section 7) has been inspired by the se-FSA based testing technique proposed by Khoumsi [17]. The se-FSA technique has various advantages as compared to others as it provides good fault coverage without the disadvantage of significant loss of scalability. Although the first step, i.e. se-FSA transformation of TRBAC_{M}, of our test generation procedure given in Section 7 is similar to Khoumsi's approach, subsequent steps differ considerably due to two reasons. First, our se-FSA transformation results in a deterministic FSA. Second, the ability to directly monitor the states considerably simplifies the CTS generation. We have also studied the problem of making the tests executable by determining the time stamps at which inputs should be generated and, at which corresponding outputs should occur.

Timed-Wp method [11] also provides complete fault coverage of TIOA faults but, at the expense of a significant loss of scalability. Timed-Wp method first samples the Region Graph (RG) [3] of the underlying TIOA to obtain a Grid Automaton (GA) which is transformed to a Nondeterministic Timed FSM (NTFSM). The test suite is generated from the NTFSM by using a generation technique based on state characterization sets. The exponential complexity of timed-Wp is primarily because of the construction of RG and GA which is exponential on the number of clocks and constants used as bounds in the time constraints [11]. Whereas, in general for most of the TIOA's the state space of se-FSA's does not increase with the magnitude of constants used in timing constraints (only increase with number of clocks) [18].

For the comparison of complexity, the region graph of $\text{UR}_{\text{b}}(u_1, r_1)$ contains 47 states as compared to only 8 states in its se-FSA. The size of the NTFSM corresponding to region graph of $\text{UR}_{\text{b}}(u_1, r_1)$, from which tests would be generated, will be even larger by virtue of sampling. Another issue with the general applicability of the Timed-Wp method is that the fault coverage is only guaranteed for a specific Implementation Under Test (IUT) architecture by assuming that clock resets are observable.

The testing approaches presented in [20, 19] and [21] are based on symbolic clustering of states into partitions coarser then regions and thus are better scalable as compared to region graph based test generation techniques. With some minor variations both approaches consider conformance between the specification

Table 5: Comparison of test generation techniques for TIOA based specifications

Test Generation Technique	Specification Model	Effectiveness*	Scalability (Cost)
se-FSA based [17]	TIOA	Complete	Medium
Symbolic states [20, 19]	TIOA with deadlines	Not measurable	Low
Symbolic states [21]	TIOA with location invariants	Not measurable	Low
Region Graph based [11]	TIOA	Complete	Very High
CTS (proposed)$^{\diamond}$	$TRBAC_M$ (constrained TIOA)	Complete	Low

**Effectiveness measured with respect to TIOA fault model.*

$^{\diamond}$HT1 and HT2 also provide complete fault coverage for TIOA faults with respect to the reduced $TRBAC_M$ considered.

and the IUT as a timed input output conformance (tioco) relation, i.e., for all the traces of the specification the IUT always produces outputs within the given temporal bounds. The problem with these approaches is that they do not consider any fault model and thus are not able to provide any guarantees of fault coverage. The *digital-clock* test generation of [19] and [20] is similar to our IP based approach used in CTS construction in Section 7.3, in terms of the semantics of the generated test sequences.

Based on the above discussion the test generation techniques for TIOA based specifications can be broadly classified into three types: se-FSA based [17], Region Graph based [11] and those based on symbolic clustering of states [20, 19, 21]. Table 5 summarizes the comparison between scalability and fault detection effectiveness of these approaches and the proposed CTS.

10 Summary and Discussion

A technique for behavior modeling of TRBAC systems and a conformance testing procedure for TRBAC ACUT's is proposed. The proposed procedure provides complete fault coverage with respect to a proposed TRBAC fault model studied in Section 5. The fault model is obtained by following the mutation based approached described in [25]. The complete fault coverage of the generated CTS is by virtue of the correctness of the TIOA based behavior modeling technique presented in Section 6, and the correlation between TRBAC, TIOA, and se-FSA faults established in Section 7.4.

The proposed conformance testing technique is based on a transformation of $TRBAC_M$ to se-$TRBAC_M$ and then using the W-method to generate the test tree (Tr). CTS is then constructed from the Tr by using an IP based approach that ensures that the test sequences satisfy the temporal constraints by only considering sending of inputs and monitoring of outputs at some integral multiple of minimum resolution k. Finally we use a specific test system architecture to execute the CTS against the ACUT and to compare the results so as to validate the ACUT conformance with respect to $TRBAC_P$. In Section 8 we show how to reduce the size of the CTS through two heuristics based on state abstraction. The decision on whether to use the complete CTS or a smaller test suite can be based on the extent of resources available for the testing process and the desired level of fault detection effectiveness.

The proposed conformance testing technique can be used to derive timed test cases for any real time system employing duration constraints. It provides complete fault coverage under the assumption that there are no faults in the ACUT$'$ due to user/system-administrator initiated deassignment/deactivation requests. This assumption is not overly restrictive and can be relaxed by explicitly including the effect of user (systems-administrator) initiated deassignment/deactivation (deassignment) requests in the UR_b (PR_b) model construction. Its consequence would be an increase in the complexity of the CTS generation procedure and the corresponding increase in the execution time of the conformance tests.

In addition to the set of temporal constraints (duration constraints) considered in our definition of TRBAC (Section 3.1), others such as periodicity constraints and temporal role hierarchies have also been proposed for RBAC models [16]. Although we consider that our modeling technique, presented in Section 6, is general enough to allow representation of these additional constraints through modifications to the proposed model generation process, yet further research is needed to precisely identify the required changes in the model generation and conformance test suite construction processes.

Functional testing of TRBAC systems is carried out as per the functional testing methodology proposed in [24] which considers a policy meta test set for test generation. The proposed CTS generation procedure is utilized in Step 3 of the proposed functional testing methodology for constructing the meta test set.

References

[1] T. Ahmed and A. R. Tripathi. Static verification of security requirements in role basedCSCW systems. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 196–203, New York, NY, USA, 2003. ACM Press.

[2] G-J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information Systems Security*, 3(4):207–226, 2000.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

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

[6] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transaction on Information and System Security*, 2(1):65–104, 1999.

[7] R. V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Massachusetts, 1999.

[8] R. Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.

[9] T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, May 1978.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Massachusetts, second edition, 2001.

[11] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed wp-method: Testing real-time systems. *IEEE Transactions on software engineering*, 28(11):1023–1038, 2002.

[12] D. Ferraiolo and R. Kuhn. Role-based access control. In *Proceedings of the NIST-NSA National (USA) Computer Security Conference*, pages 554–563, 1992.

[13] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[14] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Pearson Education, second edition, 2001.

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

[16] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005.

[17] A. Khoumsi. A method for testing the conformance of real time systems. In *7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT*, pages 331–354, 2002.

[18] A. Khoumsi and L. Ouedraogo. A new method for transforming timed automata. *Electronic Notes in Theoretical Computer Science*, 130:101–128, 2005.

[19] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *Model Checking Software: 11th International SPIN Workshop*, pages 109–126, 2004.

[20] M. Krichen and S. Tripakis. An expressive and implementable formal framework for testing real-time systems. In *17th IFIP TC6/WG 6.1 International Conference on Testing of Communicating Systems, TestCom'05*, pages 209–225, 2005.

[21] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using uppaal. In *Formal Approaches to Testing of Software, FATES 2004*, pages 79–94, 2004.

[22] E. C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[23] F. Malamateniou, G. Vassilacopoulos, and P. Tsanakas. A workflow-based approach to virtual patient record security. *IEEE Transactions on Information Technology in Biomedicine*, 2(3):139–145, 1998.

[24] A. Masood, A. Ghafoor, and A. Mathur. Scalable and effective test generation for access control systems that employ rbac policies. Technical Report CERIAS TR 2006-24, Purdue University, 2006.

[25] A. Petrenko, G. v. Bochmann, and M. Yao. On fault coverage of tests for finite state specifications. *Computer Networks and ISDN Systems*, 29(1):81–106, 1996.

[26] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58(1-3):249–261, 1988.

[27] R. Sandhu. Role activation hierarchies. In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, pages 33–40, New York, NY, USA, 1998. ACM Press.

[28] J. Springintveld, F. W. Vaandrager, and P. R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001.

[29] S. Tripakis1 and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.

[30] L. A. Wolsey. *Integer Programming*. John Wiley, New York, 1998.

[31] W. Yi. Ccs + time = an interleaving model for real time systems. In *Proceedings of the 18th international colloquium on Automata, languages and programming*, pages 217–228, 1991.

Appendix A : Algorithms and Proofs

A.1 UR_b Construction

The UR_b model is constructed by the algorithm ContructUR_b given in Figure A-1. In this algorithm $I|l$ represents the set of available inputs in location l which would be constrained by the *assumptions* discussed in Section 6.1.1. ContructUR_b uses the standard Breadth First Search (BFS) [10] to explore all the locations connected with the initial location (l_0). Initially all the unexplored locations are colored *white* and a location is colored *gray* (to indicate that is partially explored) as it is explored for the first time and also added to a First In First Out (FIFO) queue. A completely explored location is colored *black* once all the possible outgoing transitions from this location are evaluated for next location.

The procedure $update_{Location}$ is executed on each partially explored location. Details of its execution are discussed in the proof of Lemma 6.1, given below, which shows the correctness of UR_b. It is to be noted that the application of rule γ_1 or γ_2 on an input may not only cause a change in the state of the ACUT but may also require the enforcement of temporal constraints as required by the respective rule.

Proof of Lemma 6.1: Under the *assumptions* given in Section 6.1.1, we use structural induction [14] to prove the correctness of ContructUR_b by showing that each execution of the procedure $update_{Location}$ on a location correctly applies the rules of the $TRBAC_P$.

Basis: The base case is when there is only single location l_0 in the TIOA UR_b which corresponds to the ACUT state in which no user-role assignment/activations exist and we consider the first execution of the $update_{Location}$ on l_0. As discussed next the assignment and activation inputs are handled differently by the procedure $update_{Location}$.

Assignment Input: Based on our *assumptions* the sole assignment input $AS(u, r, t)$ would only be available in l_0. The operation of rule γ_1 on $AS(u, r, t)$ would require the state of ACUT to change to reflect the (u, r) assignment to be true for time t, i.e., $UR_{assign}(u, r)$ should be 1 in the next state for time t. In UR_b this is modeled as the procedure $update_{Location}$ adds the following two transitions corresponding to the application of input $AS(u, r, t)$ in l_0: (1) transition corresponding to the input action $?AS(u, r, t)$ between the current location l_0 and the next location l', which also resets the corresponding clock $x|I$, and (2) the transition corresponding to the output action $!DS(u, r)$ between l' and l_0. The two transitions combined correctly enforce the constraints (including duration constraint) on the (u, r) assignment as required by the rule γ_1.

Activation Inputs: As l_0 corresponds to the ACUT state in which u is unassigned to r, therefore application of rule γ_2 on any $AC(u, r', t)$ input should not permit the corresponding user-role activation. In UR_b this is modeled by ensuring that corresponding to the input action $?AC(u, r, t)$ only one transition representing the self loop for l_0 is added to UR_b. The procedure $update_{Location}$ thus correctly enforces the rule γ_2 on l_0. As l_0 is colored *black* only after evaluating the next location for all the input actions therefore it is

```
Algorithm ConstructUR_b
Input: (u,r), TRBAC
Output: UR_b(u,r)
l_0 ← {0,0,0,….0} where | l_0|=|R′+1|
for each l∈ L – {l_0}
  do color[l] ← white
end for
color[l_0] ← grey
Queue ← ∅
addToQueue[l_0]
while  Queue ≠ ∅
  do l← removeFromQueue()
  update_Location(l, I|l)
end while

update_Location(l, I|l)
for each  I∈ I|l
  do if I=AS(u,r,t)
      then apply rule ϒ_1 on I to determine l′←{ l | UR_assign(u,r)}
      T ← T+(l, ?AS(u,r,t),-, x|I:=0, l′)
      T ← T+(l′, !DS(u,r), x|I =t,-, l)
       t_assign ← t
       color[l′] ← grey
       addToQueue[l′]
  end if
  do if I=AC(u,r,t)
       then apply rule ϒ_2 on I to determine l′←{ l | UR_active(u,r_i)}
       do if l′ = l
         then T ← T+(l, ?AC(u,r_i,t),-,-, l′)
       else
         T ← T+(l, ?AC(u,r_i,t),-, x|I:=0, l′)
       do if color[l′] ← white                                   //first time explored
          T ← T+(l′, !DC(u,r_i), x|I =t,-, l)
          T ← T+(l′, !DS(u,r), x|?AS =t_assign,-, l_0)
          for each l″∈DeActiveSet(l′) – {l}
            T ← T+(l′, !DC(u,r_j), x_j=t_j,-, l)
          color[l′] ← grey
          addToQueue[l′]
       end if
  end if
color[l] ← black
end for
```

l|$UR_{active}(u,r)$ or l| $UR_{assign}(u,r)$ indicates new value of l with updated value of $UR_{active}(u,r)$ or $UR_{assign}(u,r)$ respectively and, DeActiveSet (l') represents the set of locations which can be reached from l' by deactivating any active user-role pair in l'

Figure A-1: Procedure for Constructing UR_b

fully explored.

Induction: Assume that the kth execution of $\text{update}_{\text{Location}}$ has correctly enforced the rules of $TRBAC_P$ on the location l_k. We further consider that the kth execution is not the last execution and there is at least on more execution, i.e., $k+1^{st}$ execution of $\text{update}_{\text{Location}}$ on the location l_{k+1}. We need to prove that the $k+1^{st}$ execution of $\text{update}_{\text{Location}}$ also correctly enforces the rules of $TRBAC_P$.

It is to be noted that by virtue of our *assumptions* only activation inputs would be available in any location other than l_0. The operation of rule γ_2 on any $AC(u,r,t)$ input applied in the ACUT state corresponding to l_{k+1} can result into two cases for the next state: either the next state is same as the current state by virtue of constraints enforced by γ_2, or the next state is different then the current one. In case of a different next state all the user-role activations in the next state are required to respect the temporal constraints imposed by both the rules γ_1 and γ_2. On its $k+1^{st}$ execution the procedure $\text{update}_{\text{Location}}$ correctively enforces the requirements of both rules in the UR_b by considering following two possible cases for the next locations corresponding to the application of $AC(u,r,t)$ inputs.

1. The next location is the same as the current location, i.e., $l_{k+1} = l_k$, which would be true if the corresponding user-role activation cannot be made due to the violation of constraints as identified by γ_2. In this case only one transition representing the self loop corresponding to the input action $?AC(u,r,t)$ is added to UR_b, thus correctly enforcing the rule γ_2.

2. The next location l' is different from l_{k+1}. In case if the location l' is unexplored (colored *white*) then in addition to the transition corresponding to the input action $?AC(u,r,t)$, various other transitions corresponding to the output actions are also added to UR_b. A transition corresponding to the output action $!DC(u,r)$ is added between the next location l' and the current location l_{k+1} and it enforces the temporal constraint implied by rule γ_2 on (u,r) activation. A transition corresponding to the output action $!DS(u,r)$ is added between l' and l_0 and it enforces the preemptive user-role deactivation as required by the rule γ_1. A number of transitions corresponding to deactivation output actions are added from l' to all such locations, which can be reached from l' by deactivating any active user-role pair in l', i.e., the elements of the set DeActiveSet(l'). It is to be noted that by virtue of BFS all the members of DeActiveSet(l') would be already explored, i.e., colored *gray* or *black*. The inductive argument that k^{th} execution of $\text{update}_{\text{Location}}$ has correctly enforced the rules of $TRBAC_P$ on the location l_k and the progress requirement imposed by the urgent outputs ensure that whenever guard of the transitions corresponding to $!DC(u,r)$ output actions is satisfied, such transitions are definitely traversed, thus correctly enforcing the rule γ_2. All the transitions combined thus correctly enforce the rules γ_1 and γ_2.

Hence the $k+1^{st}$ execution of $\text{update}_{\text{Location}}$ also correctly enforces the rules of $TRBAC_P$. This completes the inductive step and the proof. ■

A.2 UR$_M$ Construction

The algorithm ParallelComposition, given in Figure A-2, uses color coding to differentiate between explored and unexplored locations (l, l') where $l \in L$ and $l' \in L'$ of D. As initially all the locations of D are unexplored therefore they are colored *white*, and whenever a location is visited its colored *gray*. The set of transitions T_D is initialized to a null set and new transitions are added to it during execution of the recursive procedure *getNextLocations* which visits all locations in a depth first traversal [10]. In the procedure *getNextLocations* when a location (l, l') is visited for the first time it is marked *gray* and then the following process is used to determine the next location corresponding to all the transitions originating from l and l'.

- In case of a transition $(\{l, l'\}, ?i, g, R, l_t)$ on an input action, rule γ_1 or γ_2 is used to determine the next location, which could be the same location, i.e., (l, l') or a different one, i.e., (l_t, l') or (l, l_t), which if unexplored is recursively visited. The initial and next location would be same if the application of input on current state (l, l') violates the constraints in $TRBAC_P$.

- In case of a transition $(\{l, l'\}, !o, g, R, l_t)$ on an output action, the next location would always be different, which if unexplored will be recursively visited.

Proof of Lemma 6.3: We use, the fact that A and B are correct (Lemma 6.1 and Lemma 6.2), and "proof by contradiction" to prove the correctness of ParallelComposition.

Assume that ParallelComposition does not correctly apply the rules from the rule set $\Re(TRBAC_P)$ to at least one state s' of ACUT corresponding to the user-role assignments and activations modeled by A and B. Thus there is a location $\hat{l} = (l_s, l'_s)$, where $\hat{l} \in L_D$, $l_s \in L$ and $l'_s \in L'$, of D, which corresponds to the state s' in which rules γ_1 and γ_2 are not correctly applied in the recursive procedure *getNextLocations*. The rules γ_1 and γ_2 are respectively applied on the user-role assignment and activation inputs. Thus the error could be either in the application of the assignment or the activation input in $\hat{l}$. Both cases are considered separately.

Corresponding to each transition of A or B with its source l_s or l'_s respectively, *getNextLocations* adds one transition to D with its source set as (l_s, l'_s). As *getNextLocations* uses a similar procedure, for transitions with their source $l_s$or l'_s, to consider the target of the transitions added to D, therefore in the following we only consider the handling of transitions out of l_s.

Assignment Input:In case of a transition $(l_s, ?i, g, R, l_t) \in T$ on a user-role assignment input the operation of rule γ_1 with respect to the current state (l_s, l'_s) is used to determine the next state of ACUT. This is modeled in D by adding the transition $(\{l_s, l'_s\}, ?i, g, R, \{l_t, l'_s\})$ or $(\{l_s, l'_s\}, ?i, g, R, \{l_s, l'_s\})$. The first option represents the case in which the rule γ_1 permits the next state of ACUT to be (l_t, l'_s) whereas the second option corresponds to the complementary case in which the current state would not change. When the different next location (l_t, l'_s), which has to be currently unexplored, is recursively visited by *getNextLocations* the transition corresponding to user-role deassignment output action will also be added to D. As A

```
Algorithm ParallelComposition
Input: A, B, TRBAC_P
Output: D=A||_ur B
for each (l_A, l_B) ∈ (L × L′)
   do color (l_A, l_B) ←white
T_D←∅
getNextLocations (l_0, l_0′)

getNextLocations (l_s, l_s′)
do color(l_s, l_s′)←gray                                              // mark visited
for each  e= (l_s, ?i, g, R, l_t)∈ Source(l_s)^1 where ?i ≈AS               //implies  i is assignment input
   do apply rule ϒ_1 on ?i from (l_s, l_s′) to determine Permitted(l_t,l_s′)^2   //Assignment input
   do if Permitted = true
      then T_D ← T_D+({l_s, l_s′},?i ,g, R,{l_t,l_s′})
      do if color(l_t, l_s′)←white                                      // un-visited location
         getNextLocations (l_t, l_s′)                                   // recursively visit it
   else T_D ← T_D+({l_s, l_s′},?i ,g, -,{l_s,l_s′})                        // self loop
for each  e= (l_s, ?i, g, R, l_t)∈ Source(l_s) where ?i ≈AC                  //implies  i is activation input
   do apply rule ϒ_2 on ?i from (l_s, l_s′) to determine Permitted(l_t,l_s′)     //Activation input
   do if Permitted = true
      then T_D ← T_D+({l_s, l_s′},?i ,g, R,{l_t,l_s′})
      do if color(l_t, l_s′)←white
         getNextLocations (l_t, l_s′)
   else T_D ← T_D+({l_s, l_s′},?i ,g, -,{l_s,l_s′})                        // self loop
for each  e= (l_s, !o, g, R, l_t)∈ Source(l_s)                                 // Outputs
   do T_D ← T_D+({l_s, l_s′},!o ,g, R,{l_t,l_s′})
   do if color(l_t, l_s′)←white
      getNextLocations (l_t, l_s′)
for each  e= (l_s′, ?i, g, R, l_t′)∈ Source(l_s′) where ?i ≈AS
   do apply rule ϒ_1 on ?i from (l_s, l_s′) to determine Permitted(l_s,l_t′)     //Assignment input
   do if Permitted = true
      then T_D ← T_D+({l_s, l_s′},?i ,g, R,{l_s,l_t′})
      do if color(l_s, l_t′)←white                                      // un-visited location
         getNextLocations (l_s, l_t′)                                   // recursively visit it
   else T_D ← T_D+({l_s, l_s′},?i ,g, -,{l_s,l_s′})                        // self loop
for each  e= (l_s′, ?i, g, R, l_t′)∈ Source(l_s′) where ?i ≈AC
   do apply rule ϒ_2 on ?i from (l_s, l_s′) to determine Permitted(l_s,l_t′)     //Activation input
   do if Permitted = true
      then T_D ← T_D+({l_s, l_s′},?i ,g, R,{l_s,l_t′})
      do if color(l_s, l_t′)←white
         getNextLocations (l_s, l_t′)
   else T_D ← T_D+({l_s, l_s′},?i ,g, -,{l_s,l_s′})                        // self loop
for each  e= (l_s′, !o, g, R, l_t′)∈ Source(l_s′)                               // Outputs
   do T_D ← T_D+({l_s, l_s′},!o ,g, R,{l_s,l_t′})
   do if color(l_s, l_t′)←white
      getNextLocations (l_s, l_t′)
```

[1]Source(l) specifies the set of all such transitions which originates from l. [2]Permitted(l,l′) is a predicate whose value is determined by applying the rule Υ_1 or Υ_2 on the input from a starting location in D, which specifies the current state of $TRBAC_P$ to determine the validity of next location in D i.e. the next sate of $TRBAC_P$

Figure A-2: Procedure for parallel composition of two UR_b's

Algorithm ConstructPR$_b$
Input: (p,r), TRBAC$_P$
Output: PR$_b$(p,r)
$l_0 \leftarrow \{0,0,0,\ldots.0\}$ where $|l_0|=|R'|$
for each I'=AP(p′,r′,t)$\in$ I
  do apply rule Υ_3 on I′ to determine $l' \leftarrow \{ l_0 \mid PR_{assign}(p',r')\}$
  T $\leftarrow$ T+(l_0, ?AP(p′,r′,t),-, x_1:=0, l')
  do if $l' \neq l_0$
    then T $\leftarrow$ T+(l', $\wedge$\{!DP(p, r″)\}, x_1=t,-, l_0) for r″$\in$ R′ $\wedge$ R′:\{r′| r$\leq_I$ r′\}
  end if
end for
$l_0 \mid PR_{assign}$(p,r) indicates new value l' with updated value of PR$_{assign}$(p,r)

Figure A-3: Procedure for constructing PR$_b$

and B are correct therefore the input and output action transitions combined ensure that the constraints on user-role assignments as required by the rule γ_1 are correctly enforced in D.

Activation Input: In case of a transition $(l_s, ?i, g, R, l_t) \in T$ on an user-role activation input the operation of rule γ_2 with respect to the current state (l_s, l'_s) is used to determine the next state of ACUT. This is modeled in D by adding the transition $(\{l_s, l'_s\}, ?i, g, R, \{l_t, l'_s\})$ or $(\{l_s, l'_s\}, ?i, g, R, \{l_s, l'_s\})$. The first option represents the case in which the rule γ_2 permits the next state of ACUT to be (l_t, l'_s) whereas the second option corresponds to the complementary case in which the current state would not change. When the different next location (l_t, l'_s), which has to be currently unexplored, is recursively visited by *getNextLocations* the transitions corresponding to user-role de-assignment and de-activation output action will also be added to D. As A and B are correct therefore the input and output action transitions combined ensure that the constraints on user-role activation as required by the rules γ_1 and γ_2 are correctly enforced in D.

From the above discussion it can be concluded that ParallelComposition correctly applies the rules of $TRBAC_P$ on $\hat{l}$. This contradicts the assumption that rules are not correctly applied to state s'. Hence ParallelComposition correctly applies the rules. This completes the proof. ■

A.3 PR$_b$ Construction

PR$_b$ is constructed by the algorithm ConstructPR$_b$ given in Figure A-3. In this algorithm first the initial and final location is initialized. In the initial location of PR$_b(p, r)$ the status of all the permission-role pair's is unassigned and ACUT state would only change in response to the application of input $AP(p, r)$. As I-hierarchy semantics are automatically ensured by the system therefore as p is assigned to r, i.e., as $PR_{assign}(p, r) = 1$ becomes true, the permission-role assignment for all the roles senior to r in I-hierarchy is automatically applied.

Proof of Lemma 6.4: The application of rule γ_3 on $AP(p, r, t)$ would require the state of ACUT to change such that (p, r') assignments for $r' \in R'$, $R' : \{r' | r \leq_I r'\}$ is only valid for time t, i.e., $PR_{assign}(p, r') \forall r' \in$

R' should be 1 in the next state for time t. In $\mathrm{PR_b}$ this is modeled, as corresponding to the application of input $AP(p,r,t)$ in l_0, ConstructPR$_\mathrm{b}$ adds the following two transitions: (1) transition corresponding to the input action $?AP(p,r,t)$ between the current location l_0 and the next location l', which also resets the corresponding clock x_1, and (2) the transition corresponding to the output action $\wedge \{!DP(p,r')\}$, $r' \in R'$ between l' and l_0. The two transitions combined correctly enforce the constraints (including duration constraint) on the (p,r) assignment as required by the rule γ_3.

As l_0 corresponds to the ACUT state in which p is unassigned to r, therefore application of rule γ_3 on any $AP(p,r',t)$ for $r' \in (R'-r)$, input should not permit the corresponding permission-role assignment. In $\mathrm{PR_b}$ this is modeled by ensuring that corresponding to the input action $?AP(p,r',t)$ only one transition representing the self loop for l_0 is added to $\mathrm{PR_b}$, thus correctly enforcing the rule γ_3. This completes the proof. ■