CERIAS Tech Report 2005-77

SECURITY ANALYSIS AND ADMINISTRATIVE INSIDER THREAT ASSESSMENT IN ROLE-BASED ACCESS CONTROL

by Somesh Jha, Ninghui Li, Mahesh Tripunitara, Qihua Wang, William Winsborough

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

Security Analysis and Administrative Insider Threat Assessment in Role-Based Access Control

Somesh Jha* jha@cs.wisc.edu

Ninghui Li† Mahesh V. Tripunitara† Qihua Wang†
{ninghui, wangq, mtripuni}@cs.purdue.edu

William H. Winsborogh‡
wwinsborough@acm.org

November 22, 2005

Abstract

Specifying and managing access control policies is a challenging problem. We propose to develop formal verification techniques for access control policies to improve the current state of the art of policy specification and management. In this paper, we formalize classes of security analysis and administrative insider threat assessment problems in the context of Role-Based Access Control. We show that in general these problems are **PSPACE**-complete. We also study the factors that contribute to the computational complexity by considering a lattice of various subcases of the problem with different restrictions. We show that several subcases remain **PSPACE**-complete, several further restricted subcases are **NP**-complete, and identify two subcases that are solvable in polynomial time. We also discuss our experiences and findings from experimentations that use existing formal method tools, such as model checking and logic programming, for addressing these problems.

1 Introduction

Access control is one of the most fundamental and pervasive security mechanisms in use today. It controls which principals (users, processes, machines) have access to which resources in a system; for example, which files they can read, which programs they can execute, and how they share data with other principals. The specification and management of access control policies is a challenging problem, and today's administrators have no tools to assist them. As a result, a large number of security breaches are caused by policy misconfigurations. Administrators are often reluctant to change policy settings, as they do not have confidence in whether the resulting policy configurations indeed enforce the policy objectives. The current state of the art of access control policy specification and management is still "what you specify is what you get, but not necessarily what you want". This can be compared to software/hardware development before formal verification techniques [9, 37, 38] were developed and successfully deployed. We believe that formal verification techniques for access control policies can be developed to improve this current state of the art.

*Computer Sciences Department, University of Wisconsin at Madison, WI, USA

†Department of Computer Science and CERIAS, Purdue University, West Lafayette, IN, USA

‡Department of Computer Science, University of Texas at San Antonio, TX, USA

In almost all access control systems, there is a need to change the authorization state; for example, users and objects are added and removed, users start sharing resources at one moment and stop such sharing later, and users' job functionalities change. This dynamic aspect makes access control particularly challenging. A fundamental problem that deals with the dynamic aspect of access control is safety analysis, which was first formulated by Harrison, Ruzzo, and Ullman [22] for the access matrix model [19, 32]. Safety analysis decides whether undesirable right leakage could occur in future states. Recently, the notion of security analysis, which generalizes safety analysis, was introduced [34, 35]. A *Security Analysis Problem (SAP)* instance asks whether an access control system preserves security policy invariants (which encode desired security properties) across state changes. Security analysis also allows the explicit specification of trusted principals. This enables one to ask questions such as: suppose that a set of trusted principals will not initiate any potentially dangerous actions; does a policy invariant hold in all future states? A positive answer provides the assurance that the security of the system depends only on the cooperation of trusted principals.

In this paper, we introduce the notion of *Administrative Insider Threat Assessment Problems (AITAP)* in access control. An AITAP instance specifies a set A_I of principals and asks whether the system can preserve a policy invariant as long as no more than k principals in A_I actively initiate any action. The policy invariant can specify a security requirement that should always hold, in which case a "yes" answer assures that the system is robust to the collusion of any k insiders. An AITAP instance can also be used to verify that certain kinds of state changes can be made only when at least $k + 1$ principals are involved. In an AITAP instance, principals in A_I are neither completely trusted nor completely untrusted. They are insiders who have privileges and whose potential damage to the system is what we want to analyze. In other words, AITAP generalizes SAP to deal with threats posed by administrative insiders. Insider threat has long been recognized as a serious problem in security [52, 57]. Not only are insider attacks more pervasive, but they are also more destructive. Insiders have greater access to sensitive resources, deeper knowledge of internal systems, and greater opportunity to carry out their plans. Typically, malicious insiders exploit gaps in the enforcement of enterprise security policies. Even though the insider threat problem has been widely recognized as a serious problem, it has rarely been formulated in a way that technical measures can be developed to analyze and mitigate such threats.

In this paper we formally define SAP and AITAP, and study SAP and AITAP in the context of Role-Based Access Control (RBAC). We give the computational complexities of these problems, and describe our experiences building tools for performing these analyses using model checking and logic programming. Our choice of RBAC as the problem domain is motivated by the fact that RBAC [4, 14, 15, 16, 17, 50] is today's most influential access control model. The past decade has seen an explosion of research in RBAC. The industry's interest in RBAC has also increased dramatically, with most major information technology vendors offering products that incorporate some form of RBAC. Today, all major DBMS products support RBAC. Microsoft has brought RBAC to the Windows operating systems by introducing Authorization Manager in Windows Server 2003 [39]. RBAC has also been used in Enterprise Security Management Systems, such as IBM Tivoli Policy Manager [25] and SAM Jupiter [5, 26, 27, 28].

The spirit of our work can be viewed as providing techniques to help administrators precisely understand how well their access control systems can withstand various kinds of adversaries. In AITAP, we ask whether the system can withstand adversaries who control fewer than k administrative insiders.

The contributions of this paper are as follows:

- In Section 2, we provide precise definitions for SAP and AITAP. We also define URA-SAP and URA-AITAP, which are SAP and AITAP in RBAC with the URA97 administrative scheme [47, 49].

- In Section 3, we show that in general URA-SAP and URA-AITAP are $\mathbf{PSPACE}$-complete. We also study the factors that contribute to the computational complexity by considering a lattice of various subcases of the problem with different restrictions. We show that several subcases remain **PSPACE**-complete, several further restricted subcases are **NP**-complete, and we identify two subcases that are solvable in polynomial time. We observe that the administrative scheme implemented in Oracle's RBAC system falls into one of the two tractable subcases.

- In Section 4, we compare two approaches to using existing tools to perform URA-SAP and URA-AITAP and report our findings. One approach is to use model checking (specifically, the tool NuSMV [42]); the other is to use logic programming (specifically, the language XSB [20]).

We discuss related work in Section 5 and conclude in Section 6.

2 Problem Definitions

In this section, we give precise problem definitions for SAP and AITAP. We also describe the URA97 RBAC scheme and present the special cases of SAP and AITAP for the scheme.

2.1 Access Control Schemes

In existing work on security analysis in access control systems [34, 35], an access control scheme is defined as a state-transition system $\langle \Gamma, Q, \vdash, \Psi \rangle$, in which Γ is a set of states, Q is a set of queries, Ψ is a set of state-transition rules, and $\vdash$: $\Gamma \times Q \rightarrow \{true, false\}$ determines whether a query in Q is true or not in a given state in Γ. Each $\psi \in \Psi$ is viewed abstractly as a binary relation on Γ, i.e., $\psi \subseteq \Gamma \times \Gamma$; it determines whether one state can immediately reach another state. Abstracting a state transition rule as a binary relation on Γ suffices for the purpose of defining security analysis, which asks whether a security policy invariant holds in every state that is reachable from a given initial state. However, to define insider threat assessment problems, we need to identify which principals initiate a particular action to effect a state transition. We now give a definition of access control schemes that makes the initiators explicit.

Definition 1 (Access Control Schemes) An access control scheme is given by a 6-tuple $\langle \Gamma, Q, \vdash, \mathcal{A}, \Sigma, \Psi \rangle$, where Γ is a set of states, Q is a set of queries, $\vdash$: $\Gamma \times Q \rightarrow \{true, false\}$ determines whether a query is true or not in a state, $\mathcal{A}$ is a set of principals, Σ is a set of actions, and Ψ is a set of state-transition rules.

A *state*, $\gamma \in \Gamma$, contains all the information necessary to make access control decisions at a given time. When a query, $q \in Q$, arises from an access request, $\gamma \vdash q$ means that the access corresponding to the request q is granted in the state γ, and $\gamma \not\vdash q$ means that the access corresponding to q is denied. One may also ask a query that does not correspond to a specific request; for example, one may ask whether every principal that has access to a resource is an employee of an organization. Such queries are useful for understanding the properties of a complex access control system.

Each action $\sigma \in \Sigma$ is a function mapping Γ to Γ. We write $\sigma(\gamma)$ to denote the state that results from applying the action σ on the state γ. Note that $\sigma(\gamma)$ could be γ; for example, this would happen if the application of the action σ on the state γ fails. Each action σ is associated with a set of principals, denoted by $\mathsf{init}(\sigma)$, i.e., $\mathsf{init}(\sigma) \subseteq \mathcal{A}$. Principals in $\mathsf{init}(\sigma)$ are called the initiators of the action; these are the principals that actively carry out the action σ. In most existing access control schemes, each action is carried out by one initiator, in which case $\mathsf{init}(\sigma)$ is a singleton set. When $\mathsf{init}(\sigma)$ includes two principals u_1 and u_2, it means that the active participation of u_1 and u_2 is needed to carry out σ.

Each state-transition rule $\psi \in \Psi$ is given by a subset of Σ, i.e., $\psi \subseteq \Sigma$. The state transition from γ to γ_1 is allowed by ψ (we write $\gamma \rightarrow_\psi \gamma_1$) when there exists an action σ in ψ such that $\sigma(\gamma) = \gamma_1$.

Given an access control scheme $\langle \Gamma, Q, \vdash, \mathcal{A}, \Sigma, \Psi \rangle$, an access control system is specified by a pair (γ, ψ), where $\gamma \in \Gamma$ is the state of the system and $\psi \in \Psi$ is the state-transition rule that determines which state transitions are allowed.

We say that a set A of principals can take an access control system (γ, ψ) to a state γ_g if principals in A can initiate actions that change the state of the access control system from γ to γ_g, i.e., there exists a sequence of actions $\sigma_1, \sigma_2, \cdots, \sigma_n$ such that the following two conditions hold:

1. for each i such that $1 \leq i \leq n$, we have $\sigma_i \in \psi$ and $\mathsf{init}(\sigma_i) \subseteq A$.
2. $\sigma_n(\cdots \sigma_2(\sigma_1(\gamma)) \cdots) = \gamma_g$

Definition 2 (SAP) Given an access control scheme $\langle \Gamma, Q, \vdash, \mathcal{A}, \Sigma, \Psi \rangle$, a *security analysis problem (SAP) instance* is given by a 4-tuple $\langle A_T, \gamma, \psi, q \rangle$, where $A_T \subseteq \mathcal{A}$ is a finite set of trusted principals, (γ, ψ) defines an access control system, and $q \in Q$ is a query.

The answer to the instance is true if principals other than those in A_T can take the access control system (γ, ψ) to a state in which q evaluates to true. That is, this instance asks whether there exists a state γ_g such that principals in the set $\mathcal{A} - A_T$ can take (γ, ψ) to the state γ_g and $\gamma_g \vdash q$.

In an instance of SAP, q typically encodes an unsafe situation that should never occur; that is, $\neg q$ would be a policy invariant that should always hold. Oftentimes a security policy specifies that certain kinds of changes to the access control system can be made, but only through appropriate administrative channels. For example, a policy may require that in order to assign a user who is a normal employee to be a member of the manager role, at least two administrators need to be involved. Such a policy does not specify an unsafe condition that should not occur; rather, it specifies some properties about how state changes may occur and the initiators of the state changes. In this problem, the administrators are neither completely trusted nor completely untrusted. Rather, they are insiders who have privileges and whose potential to damage the system is what we want to analyze. This leads to the following definition of AITAP.

Definition 3 (AITAP) Given an access control scheme $\langle \Gamma, Q, \vdash, \mathcal{A}, \Sigma, \Psi \rangle$, an *administrative insider threat assessment problem (AITAP) instance* is given by a 6-tuple $\langle A_T, A_I, k, \gamma, \psi, q \rangle$, where A_T, γ, ψ, and q are as in Definition 2, $A_I \subseteq \mathcal{A} - A_T$ is a finite set of insiders and k is a nonnegative integer.

Given an AITAP instance, the set $\mathcal{A}$ of all principals is partitioned into three disjoint sets: A_T, the set of trusted principals, A_I, the set of insiders, and $A_U = A - (A_T \cup A_I)$, the set of untrusted principals.

The instance $\langle A_T, A_I, k, \gamma, \psi, q \rangle$ asks whether there exists a set M of k insiders (principals in A_I) such that principals in M and A_U can take the access control system to a state γ_g such that $\gamma_g \vdash q$. If the answer is no, then one knows that to take the access control system to a state satisfying q, the involvement of at least $k+1$ insiders is needed.

In our model, principals in A_T are trusted in the sense that we do not want to analyze the threat posed by them. Therefore, we assume that they do not initiate any actions. The set A_T can be empty if no principal is assumed to be trusted. Principals in A_I may possess privileges that enable them to change the state of the access control system, e.g., the privileges to grant certain permissions to other principals. In this sense they are administrative insiders. However, some principals in A_I may misuse their privileges and we want to assess the damages that may be caused by the collusion of principals in A_I. Principals in A_U are untrusted; they may collude with malicious insiders and each other to compromise security.

We observe that, given an algorithm that solves AITAP, one can use the algorithm to solve SAP, as each SAP instance $\langle A_T, \gamma, \psi, q\rangle$ can be translated into an equivalent AITAP instance $\langle A_T, \emptyset, 0, \gamma, \psi, q\rangle$. This implies that the time and space computational complexity of SAP is no worse than that of AITAP.

2.2 The URA97 RBAC Scheme

We now define the access control scheme that we study in this paper, the URA97 RBAC scheme, which is based on the ARBAC97 administrative scheme for RBAC [47, 49]. To our knowledge, ARBAC97 is the first comprehensive and the most influential administrative model for RBAC.

URA97 is one of the three components of ARBAC97 [49]. The other components of ARBAC97 are PRA97 and RRA97, for administering permission-role assignment/revocation, and the role hierarchy, respectively. In this paper, we study the effect of decentralizing user-role assignment and revocation, and assume that changes to the permission-role assignment relation and the role hierarchy are centralized, i.e, made only by trusted users. In other words, whoever is allowed to make changes to permission-role assignment and the role hierarchy will use security analysis and only make those changes that do not violate desirable security properties.

We assume that there are three countable sets: $\mathcal{U}$ (the set of all possible users), $\mathcal{R}$ (the set of all possible roles), and $\mathcal{P}$ (the set of all possible permissions). While the set of all users in any RBAC state is finite, the set of all users that could be added is potentially unbounded. One can think of $\mathcal{U}$ as the set of all possible user-identifiers in a system.

States (Γ): An RBAC state γ is a 6-tuple $\langle UA, PA, RH, CA, CR, CO\rangle$. We call UA, PA, and RH parts of the *basic state*, and CA, CR, CO parts of the *administrative state*. The basic state is described below; the administrative state is described when we discuss state transitions.

The user assignment relation $UA \subseteq \mathcal{U} \times \mathcal{R}$ associates users with roles, the permission assignment relation $PA \subseteq \mathcal{R} \times \mathcal{P}$ associates roles with permissions, and the role hierarchy relation $RH \subseteq \mathcal{R} \times \mathcal{R}$ is an irreflexive and acyclic relation over $\mathcal{R}$. We use $\succeq_{RH}$ to denote the partial order induced by RH, i.e., the transitive and reflexive closure of RH. That $r_1 \succeq_{RH} r_2$ means that every user who is authorized for r_1 is also authorized for r_2 and every permission that is associated with r_2 is also associated with r_1.

Given a state γ, each user has a set of roles for which the user is authorized. We formalize this by defining for every state γ a function $\mathsf{authorizedRoles} : \mathcal{U} \to 2^{\mathcal{R}}$

$$\mathsf{authorizedRoles}(u) \quad = \quad \{\ r \in \mathcal{R}\ \mid\ \exists r_1 \in \mathcal{R}\ [(u, r_1) \in UA \wedge (r_1 \succeq_{RH} r)]\ \}$$

When $r \in \mathsf{authorizedRoles}(u)$, we say that the user u is authorized for the role r, or equivalently, u is a member of r. We also define $down(r)$ to be the set of all roles dominated by r and $up(r)$ to be the set all roles that dominate r as follows:

$$down(r) \quad = \quad \{\ r' \in \mathcal{R}\ \mid\ r \succeq_{RH} r'\ \} \qquad\qquad up(r) \quad = \quad \{\ r' \in \mathcal{R}\ \mid\ r' \succeq_{RH} r\ \}$$

State Transition: $\mathcal{A}$, Σ, and Ψ: We now specify $\mathcal{A}$, Σ, and Ψ, which determine how states may change in the URA97 scheme. $\mathcal{A}$ is defined to be $\mathcal{U}$, the set of all possible users. Σ consists of two kinds of actions: assignment and revocation actions. Whether these actions succeed or not when applied in a state depends on the administrative state of γ, namely CA, CR, and CO, which we describe below.

- The relation $CA \subseteq \mathcal{R} \times C \times 2^{\mathcal{R}}$ determines who can assign users to roles and the preconditions these users must satisfy. C is the set of conditions, which are expressions formed using roles, the binary operators $\cap$ and $\cup$, the unary operator $\neg$, and parentheses. A tuple $\langle r_a, c, rset\rangle$ in CA means that members of the role r_a can assign any user whose role memberships satisfy the condition c, to any role $r \in rset$. For example, $\langle r_0, r_1 \cap r_2 \cap \neg r_3, \{r_4\}\rangle \in CA$ means that a user that is a member of the role r_0 is allowed to assign a user that is a member of both r_1 and r_2, and is not a member of r_3, to be a member of r_4.

- The relation $CR \subseteq \mathcal{R} \times 2^{\mathcal{R}}$ determines who can remove users from roles. That $\langle r_a, rset\rangle \in CR$ means that the members of role r_a can remove a user from a role $r \in rset$.

  We assume that CA and CR satisfy the property that the administrative roles are not affected by CA and CR. The administrative roles are those that appear in the first component of each tuple in CA or CR. These roles should not appear in the last component of any CA or CR tuple. This condition is satisfied in URA97, which assumes the existence of a set of administrative roles that is disjoint from the set of normal roles.

- CO is a set of mutually exclusive role constraints. Each constraint in CO has the form $\mathsf{smer}\langle\{r_1, \ldots, r_m\}, t\rangle$ where each r_i is a role, and m and t are integers such that $1 < t \leq m$. This constraint forbids a user from being a member of t or more roles in $\{r_1, \ldots, r_m\}$. We say that a set R of roles satisfies a constraint $\mathsf{smer}\langle\{r_1, \ldots, r_m\}, t\rangle$ if and only if $|R \cap \{r_1, \ldots, r_m\}| < t$, where $|\,|$ gives the cardinality of a set.

  For example, $\mathsf{smer}\langle\{r_1, r_2\}, 2\rangle$ means that no user is allowed to be a member of both r_1 and r_2. In an RBAC state γ, if $r_1 \in \mathsf{authorizedRoles}(u)$ for a user u, then an assignment action that assigns the user u to any role in $up(r_2)$ would fail because of the constraint.

Σ, the set of all actions, is defined as follows:

$$\Sigma = \{assign(u_a, u_t, r_t) \mid u_a, u_t \in \mathcal{U} \wedge r_t \in \mathcal{R}\} \cup \{revoke(u_a, u_t, r_t) \mid u_a, u_t \in \mathcal{U} \wedge r_t \in \mathcal{R}\}$$

- An assignment action $assign(u_a, u_t, r_t)$ means that the user u_a assigns the user u_t to the role r_t. When this action is applied to an RBAC state γ, it succeeds if and only if the following three conditions hold:
  - $(u_t, r_t) \notin UA$, i.e., the user u_t is not already assigned the role r_t.
  - There exists a tuple $\langle r_a, c, rset\rangle \in CA$ such that $r_a \in \mathsf{authorizedRoles}(u_a)$, $\mathsf{authorizedRoles}(u_t)$ satisfies c, and $r_t \in rset$.
  - $\mathsf{authorizedRoles}(u_t) \cup down(r_t)$ satisfies every constraint in CO, i.e., the new role memberships of u_t do not violate any constraint.

  We allow the assignment action $assign(u_a, u_t, r_t)$ to succeed when u_t is already authorized for r_t indirectly through other roles. For example, even when $(u_t, r_s) \in UA$ and $r_s \succeq_{RH} r_t$, $assign(u_a, u_t, r_t)$ can succeed. The rationale is that these memberships often represent independent relationships. For example, $(u_t, r_s) \in UA$ may represent a shorter-term role assignment for u_t because of temporary staff shortages, and $(u_t, r_t) \in UA$ may represent a longer-term role assignment; then we want to add (u_t, r_t) to UA so that when (u_t, r_s) is removed from UA, u_t is still authorized for r_t.

  When the assignment action is successfully applied to an RBAC state γ, the resulting state γ' differs from γ only in the user-role relation. The result of a successful application is $UA' = UA \cup \{(u_t, r_t)\}$. When the application is not successful, the state does not change.

- A revocation action is of the form $revoke(u_a, u_t, r_t)$, which means that the user u_a revokes the user u_t from the role r_t. When this action is applied to an RBAC state γ, it succeeds if and only if the following two conditions hold:
  - $(u_t, r_t) \in UA$, i.e., the user u_t is assigned to the role r_t.
  - There exists a tuple $\langle r_a, rset\rangle \in CR$ such that $r_a \in \mathsf{authorizedRoles}(u_a)$, and $r_t \in rset$.

  When the revocation action is successfully applied to an RBAC state γ, the resulting state γ' differs from γ only in the user-role relation. The result of a successful application is $UA' = UA - \{(u_t, r_t)\}$. When the application is not successful, the state does not change.

Ψ consists of a single state-transition rule, ψ, where ψ includes all actions in Σ.

2.3 SAP and AITAP in URA97

Definition 4 (URA-SAP) A URA-SAP instance is given by an RBAC state $\gamma = \langle UA, PA, RH, CA, CR, CO\rangle$, a set $A_T \subseteq \mathcal{U}$ of trusted users, and a query.

We deliberately leave the syntax for queries unspecified in the above definition. Different kinds of queries may be needed for different policy analyses. The simplest kind is to ask whether a user u is a member of a role r. More sophisticated queries may ask whether a user's role membership satisfy a condition (e.g., $r_1 \cup (r_2 \cap \neg r_3)$), or whether the set of members of one role is a subset of the set of members of another role.

Observe that a query that asks whether a user is a member of a role can be used to handle several other kinds of queries. For example, if one wants to know whether the system can reach a state in which u's role membership includes a set $\{r_1, r_2\}$ and excludes $\{r_3\}$, one can add a new user u_a, two new roles r_a and r_t, a user assignment (u_a, r_a), and a new tuple $(r_a, (r_1 \cap r_2 \cap \neg r_3), \{r_t\})$ to CA, and use $u \in r_t$ as the query. Similarly, if one wants to know whether the system can go to a state in which u possesses a certain set of permissions, one can compute the role condition that is necessary and sufficient to have the permissions and then translate that into a query about a single role.

Definition 5 (URA-RC-SAP) A URA-RC-SAP instance is a special case of URA-SAP in which a query has the form $u \in r$.

Definition 6 (URA-AITAP and URA-RC-AITAP) A URA-AITAP instance is given by an RBAC state $\gamma = \langle UA, PA, RH, CA, CR, CO\rangle$, a set $A_T \subseteq \mathcal{U}$ of trusted users, a set $A_I \subseteq \mathcal{U}$ of insiders, an integer k, and a query. A URA-RC-AITAP instance is a URA-AITAP instance where the query has the form $u \in r$.

2.4 An example

Figure 1 shows the role-hierarchy of a branch of a bank. This example is inspired by a case-study of a commercial bank that appears in the literature [51]. The bank has two departments, `Loans` and `Retail`. The bank prefers that an employee be a member of exactly one (functional) role. However, owing to practical considerations, employees have to sometimes be allowed to be members in multiple roles across the two departments. We quote from [51]: "*Ideally, each employee is assigned to one role. However, in special circumstances an employee may be given up to four roles (e.g., in case of illness of a colleague)*". Furthermore,

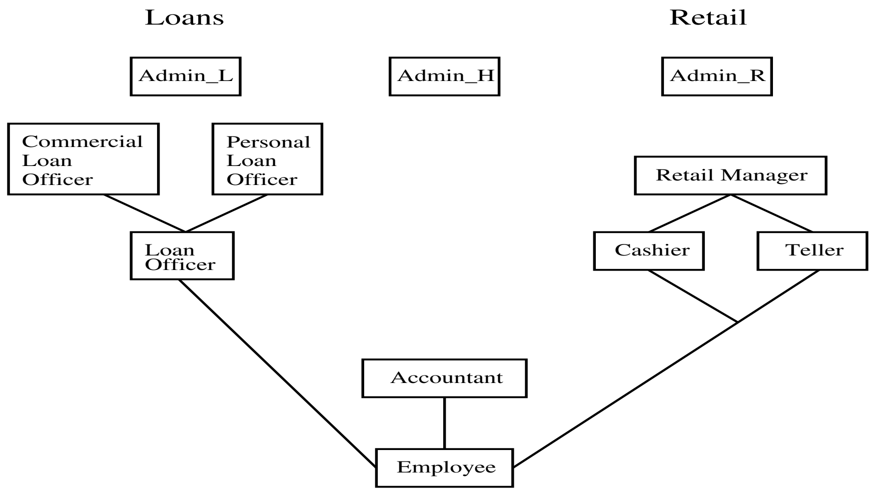

Figure 1: A branch of a bank that comprises the two departments, Loans and Retail. The roles Admin_L is the administrative role for the three roles in the Loans department; and the role Admin_R is the administrative role for three roles in the Retail department. The role Admin_H is the administrative role associated with the Human Resources department. This role has control over Accountant and Employee, which do not fall into either the Loans or the Retail department.

there is a role, Accountant, that does not fall in either department, and therefore is directly administered by the Human Resources department.

In Appendix A, we detail the state and a rather comprehensive state-change rule (assign and revoke operations, and constraints) associated with this role-hierarchy. Consider the query $q =$ "$Bob \in \{\text{Cashier}\}$", for some user Bob that is not a member of any role. The only entry in CA with Cashier in its third component is $\langle$Admin_R, Accountant $\wedge \neg$Loan Officer, $\{$Cashier, Teller, Retail Manager$\}\rangle$. In our example, the administrative user $Andy$ is a member of Admin_R. Therefore, provided Bob is already a member of Accountant and not a member of Loan Officer (the latter may be via inheritance, as the role has two senior roles), $Andy$ can assign Bob to the role Cashier. In our discussion in Appendix A, we consider the more complex issue of whether Bob can become a member of both Cashier and Loan Officer simultaneously, as a result of actions taken by some maximum number of colluding administrators. We observe there that the collusion of at least three administrators is needed for an undesirable state (a state in which Bob is a member of both Cashier and Loan Officer) to be reachable.

3 Computational Complexity

In this section we study the computational complexity of URA-SAP. In particular, we show that URA-RC-SAP is **PSPACE**-complete. We also relate the computational complexity of AITAP to SAP.

The main source of the complexity of SAP is the fact that the state space that needs to be explored is potentially large. We would like to understand how different features in URA97 affect this search space; therefore, we consider special cases of URA-SAP that result from restricting the URA scheme in various ways. Answers to the following questions affect the computational complexity of URA-SAP.

- *What queries are considered?* If queries are allowed to contain conjunctions and disjunctions of roles, then URA-SAP is likely to be intractable. For example, in [35], queries such as $((r_1 \cup r_2) \cap r_3) \subseteq ((r_1 \cup r_2) \cap (r_2 \cup r_3))$ can be posed. The intractability results in [35] are consequences of the fact that these sophisticated queries can encode propositional formulas to show $\mathbf{NP}$-hardness. In this section, we focus on the simplest kind of queries, i.e., whether a user u is a member of a role r, to better understand the complexity caused by features within URA97. In other words, we consider URA-RC-SAP.

- *Do the preconditions involve only conjunctions?* Each tuple in CA has a precondition. It is conceivable that if the precondition involves arbitrary conjunction, disjunction, and negation of roles, then this could make the problem intractable; however, such a result would be less insightful and of less practical interest. In practical systems, one would not expect the precondition to be an arbitrary logical formula in CNF with many conjuncts. Furthermore, we show below that for the general case allowing complex preconditions does not affect the computational complexity. For these reasons, in this paper we focus on the special case in which each precondition is a conjunction of roles or their negations.

- *Is negation allowed in preconditions in CA?* When preconditions in CA may contain negation, one needs to consider the revocation of a users' role memberships in order to satisfy the precondition and be assigned to a new role.

- *Are SMER constraints allowed, i.e., is $CO = \{\ \}$?* When constraints are allowed, one may need to consider revocations in order to be able to assign a user to a new role.

- *Are revocations allowed, i.e., whether $CR = \{\ \}$?* One may want to consider the special case that role memberships cannot be revoked.

We summarize the variations we consider in this paper in Figure 2. The main results of this paper are stated in the following theorem. These results are also summarized in Figure 3.

Theorem 1 *The computational complexity for URA-RC-AITAP, URA-RC-SAP, and its various subcases are as shown in Figure 3.*

Some subcases of the problem are not listed in Figure 3, because they are special cases of the two cases that are known to be in $\mathbf{P}$; thus they are solvable in polynomial time as well. We make several observations from Theorem 1 and Figure 3.

- URA-RC-AITAP has the same space and time-complexity as URA-RC-SAP (they are both $\mathbf{PSPACE}$-complete). This follows a general result that AITAP can be solved using essentially the same space as SAP (Lemma 2).

- Whether we allow only conjunctive preconditions or allow arbitrary preconditions does not change the computational complexity of URA-RC-SAP in general. The problem is $\mathbf{PSPACE}$-complete with or without the conjunctive restriction.

- There are three cases in which the problem's complexity changes from $\mathbf{PSPACE}$-complete to $\mathbf{NP}$-complete. All three result from making CR empty. The reason is that if CR is empty, then one only needs to consider role assignments. Any role assignment sequence can have length at most polynomial in the size of the problem instance. This makes the problem in $\mathbf{NP}$. On the other hand, there may

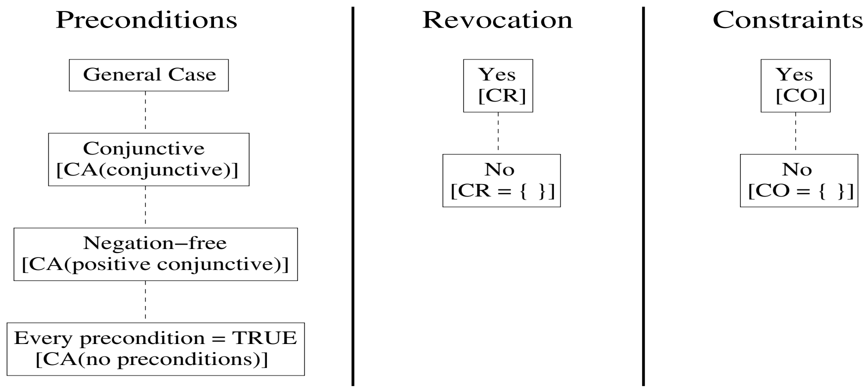

Figure 2: The possible variations in the features we consider for the preconditions in CA, revocation and constraints. A dotted line connects a case with a subcase. For example, negation-free preconditions ([CA (positive conjunctive)]) is a subcase of conjunctive preconditions ([CA (conjunctive)]). Various combinations from the three columns are possible. For example, we can consider the analysis problem with negation-free preconditions, with revocation, but without constraints, which corresponds to URA-RC-SAP[CA (positive conjunctive), CR, $CO = \{\ \}$].

be exponentially many such possible assignment sequences; thus the problem remains $\mathbf{NP}$-complete. However, when CR is not empty, the sequence necessary for entering a user into a role may be of exponential length.

- The effect of non-empty CO is identical to the effect of negation in preconditions of CA from the standpoint of computational complexities. The reason is that the effect of a constraint in CO can be "simulated" using negative preconditions in CA, and vice versa. We use this fact, for example, in the proofs for Lemmas 5 and 8.

The rest of this section proves the results in Theorem 1.

URA-RC-AITAP, URA-RC-SAP, and three subcases are $\mathbf{PSPACE}$-complete: We first show that SAP and AITAP can be solved in the same space complexity. We then show that the general case of URA-RC-SAP is in $\mathbf{PSPACE}$. Finally, we show that two subcases URA-RC-SAP[CA(positive conjunctive), CR, CO], and URA-RC-SAP[CA(conjunctive), CR, $CO = \{\ \}$] are $\mathbf{PSPACE}$-hard. These results together prove the five $\mathbf{PSPACE}$-completeness result in Theorem 1. In Appendix B, we present background information related to Turing machines that we use to establish these results.

Lemma 2 *For any access control scheme, if SAP has space complexity $O(f(n))$ then AITAP has space complexity $O(f(n) + n)$.*

Proof: Given an algorithm that solves SAP, and an AITAP instance, $\langle A_T, A_I, k, \gamma, \psi, q\rangle$, we enumerate through all size-k subsets of A_I, and for each such subset S we solve the SAP instance $\langle A_T \cup (A_I - S), \gamma, \psi, q\rangle$. In addition to the space used by the algorithm to solve SAP, we need to remember

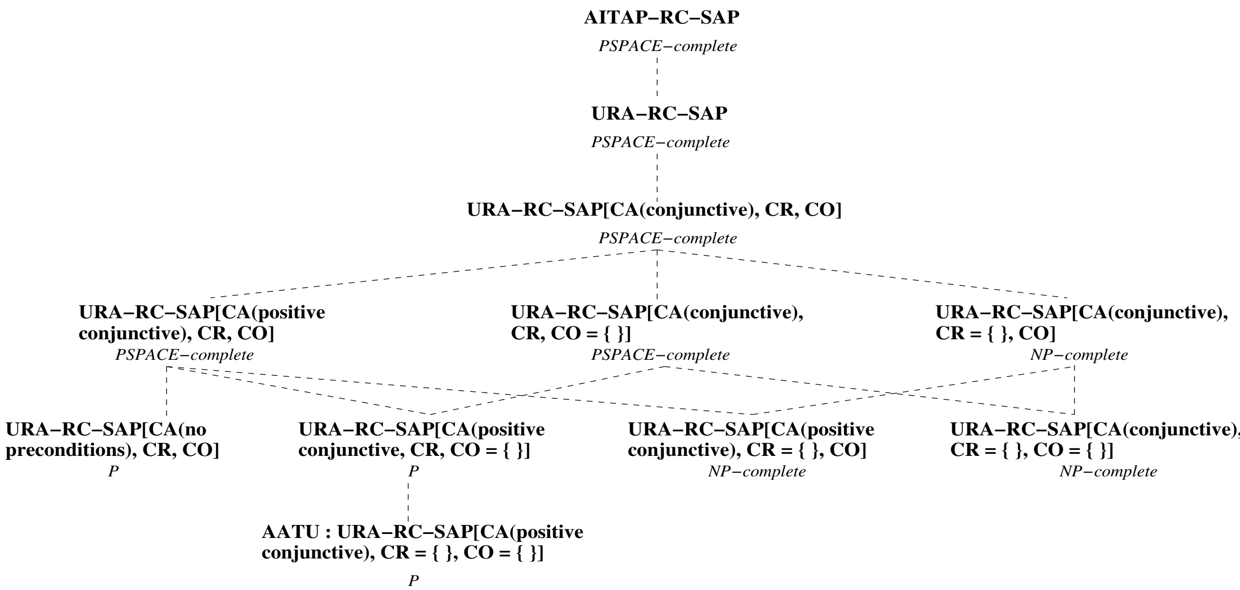

Figure 3: The summary of computational complexity results for various cases of URA-RC-SAP. A dotted line links a case to a subcase. For example, URA-RC-SAP[CA (positive conjunctive), CR, CO] is a subcase of URA-RC-SAP[CA (conjunctive), CR, CO].

the AITAP instance and a marker denoting which subsets of A_I have been checked. This takes space linear in the size of the AITAP instance.

Lemma 3 URA-RC-SAP is in $\mathbf{PSPACE}$.

Proof: Given a URA-RC-SAP instance, let $\gamma = \langle UA, PA, RH, CA, CR, CO \rangle$, let U_T be a set of trusted users, and let $u \in r$ be the query. Notice that the only component of the state that changes is UA. Furthermore, we only need to maintain u's role memberships (the number of roles does not change from the start state). It takes polynomial space to represent the u's role memberships. Recall that in URA97, administrative roles (i.e., roles that appear in the first component of a tuple in CA and CR) are not affected by CA and CR. Therefore, we do not need to consider memberships of users other than u because these role memberships do not affect whether u can be assigned to a role or not. Observe that in order to determine whether u can be added to a role, the precondition is only about u's role memberships.

Observe that if we relax the restriction that administrative roles are not affected by CA and CR, then we need to maintain role memberships of all users, which can still be done in polynomial space. Therefore, URA-RC-SAP is in $\mathbf{PSPACE}$ even without this restriction.

We describe a Non-Deterministic Turing Machine (NDTM) NM to solve this problem. Initially, NM sets initial state γ_0 to be equal to γ. Given that that NM is in state γ_i, it continues its computation as follows:

- If $\gamma_i \vdash q$, NM stops and outputs yes.
- Assume that $\gamma_i \not\vdash q$. NM guesses a next state γ_{i+1} that changes the user u's role memberships. NM ensures that such a change conforms to CA and CR and satisfies the constraints in CO. NM ensures also that the only users that effect an assign or revoke action are the ones do not belong to U_T.

The construction given above proves that URA-RC-SAP is in $NSPACE(O(n))$, where n is the space needed to represent the input URA-RC-SAP instance. Using Savitch's theorem we can conclude that URA-RC-SAP is in $DSPACE(O(n^2))$. $\Box$

Lemma 4 URA-RC-SAP [CA (positive conjunctive), CR, CO] is $\mathbf{PSPACE}$-hard.

The proof is given in Appendix C. The proof is by a reduction from the membership problem for linear bounded automata (LBA), which is known to be $\mathbf{PSPACE}$-complete. A LBA is a restricted form of a Turing machine. It differs from a Turing machine in that while the tape is initially considered infinite, only a finite contiguous portion whose length is a linear function of the length of the initial input can be accessed by the read/write head.

Lemma 5 URA-RC-SAP [CA (conjunctive), CR, $CO = \emptyset$] is $\mathbf{PSPACE}$-hard.

The proof for the above lemma is similar to the proof for Lemma 4. The proof for Lemma 4 uses 2-2 SMER constraints. We simulate the use of such constraints in the proof for Lemma 5 using negative preconditions in CA. For example, if a 2-2 SMER constraint $\mathsf{smer}\langle \{r_1, r_2\}, 2\rangle$ is used in the proof for Lemma 4, we can add $\neg\, r_1$ to the precondition of each rule that assigns to r_2 and add $\neg\, r_2$ to the precondition of each rule that assigns to r_1.

URA-RC-SAP [CA (conjunctive), $CR = \{\ \}$, CO] and its two subcases are $\mathbf{NP}$-complete: We first demonstrate that disallowing revocations in URA-RC-SAP causes the problem to be in $\mathbf{NP}$. We then demonstrate that two subcases are both $\mathbf{NP}$-hard. The first subcase is when we disallow negative preconditions in CA. The second subcase is when we disallow constraints. These results together prove the three $\mathbf{NP}$-completeness results in Theorem 1.

Lemma 6 URA-RC-SAP [CA (conjunctive), $CR = \{\ \}$, CO] is in $\mathbf{NP}$.

Proof. We need to demonstrate that if an instance of URA-RC-SAP [CA (conjunctive), $CR = \{\ \}$, CO] is true, then there exists an evidence of size polynomial in the problem that can be efficiently verified. Let the query q in the problem instance be about user u's membership in the role r. As the evidence, we use the shortest state-change sequence from the initial state γ_0 to a state γ_n such that $\gamma_n \vdash q$. Each state-change in this sequence is the assignment of u to a role of which he is not already a member. There can be at most $|R|$ such assignments, where R is the set of roles in the system. (See the proof of Lemma 3 about why we only need to consider assignment of user u. Also observe that even if considering assignment of all users, the total number of such assignments is still polynomial in the size of the instance.) Therefore, the state-change sequence of length at most $|R|$, which is polynomial in the input and can certainly be verified in polynomial time. ∎

Lemma 7 URA-RC-SAP [CA (conjunctive), $CR = \{\ \}$, $CO = \{\ \}$] is $\mathbf{NP}$-hard.

The proof is in Appendix D; it uses a reduction from the 3SAT problem.

Lemma 8 URA-RC-SAP [CA (positive conjunctive), $CR = \{\ \}$, CO] is $\mathbf{NP}$-hard.

The proof is in Appendix E. This result should not be surprising given Lemma 7; as discussed earlier, the effects of SMER constraints and negation in preconditions in CA are very similar.

Two subcases that are in $\mathbf{P}$: As we show above, either negation in preconditions or SMER constraints is sufficient to make URA-RC-SAP intractable. However, URA-RC-SAP [CA (positive conjunctive), CR, $CO = \{\ \}$], that is, when neither negations in preconditions nor SMER constraints are allowed, the problem can be solved in linear time. The reason is that to determine whether u can be a member of a role r in some future state, there is no need to consider revocation as there is no negation in preconditions in CA and there are no SMER constraints. A straightforward quadratic algorithm is to try each tuple in CA and see whether u can be assigned to more roles. As the number of roles that can be assigned according to CA is bounded by the size of CA, this algorithm takes at most quadratic time. A linear time algorithm can be obtained by reducing this to the Horn-SAT problem, which can be solved in linear time [13]. Each rule in CA can be viewed as a Horn rule; for example, if one such rule says $r_1 \cap r_2$ is the precondition for r_3, then this can be translated into a Horn clause "$r_3 \longleftarrow r1, r2$". Each initial role membership of the user can be translated into a Horn clause. The query can be translated into a Horn query clause.

Another tractable subcase is URA-RC-SAP [CA (no preconditions), CR, CO], which can be solved in quadratic time. In this subcase, every pre-condition in CA is "true", but we allow revocations and SMER constraints. The algorithm is to first check whether the user u is already a member of the role r (where u and r comprise the query). If not, we revoke u from as many roles as possible using entries from CR. We then check whether there exists an entry in CA that we can exercise to cause u to become a member of r, while not violating any entry in CO. If yes, the algorithm returns "true" and otherwise, it returns "false." This is linear in the sizes of CA, CR and CO.

We observe that in some RBAC schemes in practical systems, such as the RBAC scheme in the Oracle database, there is no precondition in role assignment. Security analysis there thus falls under the above tractable case.

Note that for more sophisticated queries, URA-SAP even in these basic versions may not be tractable. For example, the computational complexity when considering queries that ask whether members of a role r_1 is always a subset of the role r_2 is still open.

4 Experimental Results

The fact that URA-RC-SAP and several of its subcases are intractable (**PSPACE**-complete or **NP**-complete) means that there exist difficult problem instances. In this section, we describe our experiences using logic programming and model checking tools for some realistic instances of URA-RC-SAP. Our goals for performing these experiments are two-fold. First, we would like to see whether security analysis instances of nontrivial sizes can be solved in reasonable amounts of time. Our experimental results show that the answer is positive. Second, we would like to compare the effectiveness of model checking and logic programming in security analysis. Our results demonstrate that logic programming outperforms model checking in smaller instances; however, model checking appears to scale better than logic programming.

The Logic Programming Approach: Logic programming is a declarative, relational style of programming based on first-order logic. A logic program is composed of a set of facts and a number of rules which specify how to derive new facts from known ones. We use XSB [20], a Prolog-variant logic-programming system developed at SUNY, Stony Brook. XSB uses SLG resolution [8], which can correctly evaluate many recursive logic programs that would cause SLD-resolution-based Prolog systems to fail to terminate.

Our implementation is a natural reduction from instances of URA-RC-SAP to logic programs. Recall that an instance of URA-RC-SAP consists of an RBAC state $\langle UA, PA, RH, CA, CR, CO \rangle$ and a query of the form $u \in r$, where $u \in \mathcal{U}$ is a user and $r \in \mathcal{R}$ is a role. Our logic program defines a predicate over states that is true when the state is reachable. Each tuple in CA and CR is represented as a rule, while RH and CO are incorporated in the rules representing CA entries. The initial role memberships of the user u is given as a fact, and the evaluation's goal is to find a state in which the answer to the query $u \in r$ is true.

The Model Checking Approach: Model checking is a technique for determining whether a formal model M of a system satisfies a temporal-logic property p. A model M can be represented as a 4-tuple (S, R, s_0, L), where S is a finite set of states, $R \subseteq (S \times S)$ is a transition relation, $s_0 \in S$ is an initial state, and $L : S \rightarrow 2^{AP}$ is a labeling of states with propositional formulas from AP (given a state s, $L(s)$ denotes the atomic propositions in AP that are true in s). We express safety property p in *Computation Tree Logic* (*CTL*) with the form $\mathbf{AG}f$ (i.e., $p = \mathbf{AG}f$, where f is a formula in propositional logic). ($\mathbf{AG}f$ means that *always globally* the atomic proposition f is true, or in other words f is true in every state reachable from the initial state s_0.) If the model M satisfies the property p, a model checker reports true. If M does not satisfy p, a model-checker produces a counter-example that shows an execution that leads to a violation of the property. A thorough treatment of model checking is provided in [9].

The model checker we used is NuSMV [42]. We implemented a program that reads an instance of URA-RC-SAP and then generates a NuSMV program for the instance. Encoding an instance of URA-RC-SAP as a model in NuSMV is straightforward, e.g., states correspond to user assignments to roles and transitions correspond to rules in CA and CR.

Preprocessing: We observe that given a URA-RC-SAP instance, many rules in CA and CR may be irrelevant to the query. We use a preprocessing stage to remove these rules. Our experimental data shows that preprocessing can be very effective. Given a query $u \in r$ and an RBAC state, our preprocessing does the following two kinds of pruning:

- *Forward pruning:* We remove rules that will never be successfully executed. We first compute R_{lo}, the set of roles in the initial state that cannot be revoked by rules in CR. We then compute R_{up}, the set of roles that may be assigned to the user u, and A, the set of assignment rules that may be successfully applied. To do this, we initialize R_{up} with I, the initial set of roles that u is a member of, and A with $\emptyset$. For each assignment rule α in CA, if the target role of α is not in R_{lo}, the positive precondition of α is satisfied by R_{up}, and the negative precondition of α does not contain any role in R_{lo}, we add the target role of α to R_{up} and add α to A. We repeat this process of iterating through CA until R_{up} does not grow. Only assignment rules in A and revocation rules that revoke roles in R_{up} are kept after the pruning. Letting $|CA|$ be the number of rules in CA, the computation of R_{up} and A requires $O(|CA|^2)$ rule-consideration steps, because each pass through CA adds at least one $\alpha \in CA$ to A and each such α needs not be considered thereafter.

- *Backward pruning:* Some roles may be irrelevant to assigning the role r in the query. The backward pruning removes assignment and revocation rules about those roles. We compute two sets of roles: R_{po} is the set of roles that r positively depends on, and R_{ne} is the set of roles that r negatively depends on. We remove assignment rules that assign roles outside R_{po} and revocation rules that revoke roles outside R_{ne}. R_{po} is the smallest set that satisfies the following three conditions: (1) $r \in R_{po}$; (2) if $r_p \in R_{po}$, then any role that dominates r_p is also in R_{po}; (3) if $r_p \in R_{po}$, then any role that appears in the positive precondition of a CA entry assigning to r_p is also in R_{po}. R_{ne} is the smallest set that satisfies the following conditions: (1) if $r_p \in R_{po}$, then any role that appears (or dominates a role that appears) in the negative precondition of a CA entry assigning to r_p is in R_{ne}; (2) if $r_p \in R_{po}$, then any role that is (or dominates a role that is) mutually exclusive with r_p is in R_{ne}.

The preprocessing takes time at most cubic in the size of the instance.

Experimental results: We performed experiments using two kinds of instances. Manually crafted instances are designed to "hide" an unsafe state after a long sequence of transitions. These instances forced the analysis tools to search deep in the state space. Randomly generated instances contain a relatively large number of roles and transition rules.

Experiments were performed on a workstation with an Intel P4 3G processor and 512M of memory, running Windows XP Professional. We tested the performance of NuSMV and XSB on several instances. Table 1 presents results for seven instances, two of which were manually generated and five of which were randomly generated.

Experimental results show that the number of rules is a crucial factor in determining runtime. Therefore, the preprocessing step plays an important role in improving the efficiency of both logic programming and model checking implementations. Many instances, such as Rand2, that cannot be solved within 30 minutes without preprocessing are solved within a few seconds with preprocessing.

	Man1	Man2	Rand1	Rand2	Rand3	Rand4	Rand5
Num. of Roles	12	16	15	100	40	30	25
Num. of Rules	31	40	45	250	92	88	79
Num. of Rules *(AP)*	22	29	34	20	27	37	57
Transition Length	15	22	1	2	3	2	6
Total States	1.70E+7	1.76E+9	6.60E+8	NA	NA	NA	NA
Reachable States	1.31E+5	2.69E+6	7.54E+5	NA	NA	NA	NA
XSB Runtime	0.55s	14.22s	109.64s	NA	NA	NA	NA
NuSMV Runtime	0.188s	1.78s	2.10s	NA	NA	NA	NA
Total States*(AP)*	1.22E+7	1.29E+9	5.02E+8	1.08E+49	3.40E+20	7.82E+15	4.90E+13
Reachable States *(AP)*	3456	6000	6720	75264	7.31E+5	3.11E+6	3.98E+7
XSB Runtime *(AP)*	0.02s	0.05s	0.06s	0.55s	5.70s	19.94s	NA
NuSMV Runtime *(AP)*	0.11s	0.13s	0.13s	0.94s	1.27s	5.44s	72.94s

Table 1: Experimental data on URA-RC-SAP instances using XSB and NuSMV. Instances with names beginning with Man were manually crafted, while those beginning with Rand were randomly generated. Statistics on total states and reachable states is for NuSMV only. Rows marked *(AP)* present results after preprocessing. NA indicates that, for the NuSMV cases, the program did not finish running within 30 minutes, or, for the XSB case, ran out of the memory.

Both XSB and NuSMV are efficient in cases that have a small number of transition rules. For the example in Section 2.4, XSB uses 0.016 seconds and NuSMV uses 0.125 seconds. When tested on a manually crafted instance Man2 with 16 roles and 40 rules (29 left after preprocessing) that requires a sequence of at least 22 transitions before reaching an unsafe state, XSB uses 0.05 seconds and NuSMV uses 0.13 seconds. When preprocessing is effective, such as instance Rand2 with 100 roles and 250 rules (20 rules left after preprocessing), XSB uses 0.55 seconds and NuSMV executes in 0.94 seconds.

It appears that using XSB does not scale as well as using NuSMV. For example, when it came to the randomly generated instance Rand5, with 25 roles and 79 rules (57 left after preprocessing), XSB ran out of memory after 29 minutes, while NuSMV returned with an answer within 73 seconds. An observation is that the runtime of the XSB grows linearly with the number of reachable states.[1] However, NuSMV uses binary decision diagrams (BDDs) to represent its state space, so its runtime depends on the regularity of the state space. A point worth mentioning is that XSB consumes memory quickly. For the instance Rand4 with 30 roles and 88 rules (27 left after preprocessing), XSB uses more than 400MB of memory. The high demand on memory impairs the scalability of our XSB program. In contrast, the BDD-based NuSMV requires less memory than XSB.

Both model checking and logic programming have been used in network-vulnerability analysis [23, 53]. Recently, Ou et al. [43] showed that, in the context of network-vulnerability analysis, logic programming is much more scalable than model checking. Our experimentation data show that for URA-RC-SAP scalability of logic programming is worse than model checking. This is because in network vulnerability analysis, one can make the monotonicity assumption, i.e., if an attacker gains a privilege, it never loses it. However, in security analysis, because of negative preconditions and mutual exclusion constraints, the monotonicity assumption does not hold, and one has explore the state space.

[1]The statistics on total states and reachable states in Table 1 are actually for NuSMV, but the statistics for XSB should be similar.

In real-word large-scale RBAC systems, even though the number of roles in the whole system may be large, we expect that the roles that are relevant for any given query will be only a small portion of all roles. Therefore, we conjecture that our approach of combining preprocessing with existing tools such as NuSMV will be able to handle many queries.

5 Related Work

In their landmark paper [22], Harrison et al. formalized the safety analysis problem in the access matrix model [32, 19]; the problem determines whether a protection system can reach a state in which a particular right is leaked. They show that safety analysis is undecidable in their scheme [22]. Since then, safety analysis has attracted considerable attention in the research community. Safety analysis in monotonic versions of the HRU scheme has been studied in [21]. Jones et al. introduced the Take-Grant scheme [24, 36], in which safety can be decided in linear time. Sandhu et al. introduced the Schematic Protection Model [45], the Extended Schematic Protection Model [1, 2], and the Typed Access Matrix model [46]. Budd [7] and Motwani et al. [40] studied grammatical protection systems. Soshi et al. [55, 56] studied safety analysis in the Dynamic Typed Access Matrix model. These models all have subcases where safety is decidable. Solworth and Sloan [54] introduced a discretionary access control model in which safety is decidable. This thread of research has produced many new access control schemes but has had limited impact on access control systems used in practice, probably because the proposed schemes are either too simplistic to be useful or too arcane to be usable. In this paper, we focus on policy analysis problems in RBAC, which was invented not for the purpose of safety analysis, but for meeting the access control need of real-world applications.

The notion of roles was first introduced in access control in the context of database security [6, 58] as a means to group permissions together to ease security administration. Influential works on RBAC include the pioneering work by Ferraiolo et al. [14, 15] and the widely cited RBAC96 family of formal RBAC models developed by Sandhu et al. [50]. Recently, a standard for RBAC has been proposed and adopted as an ANSI Standard [4, 17]. Administration of RBAC is about controlling who can update the various relations in an RBAC system. The most well-known work on administration of RBAC is ARBAC97, developed by Sandhu et al. [47, 48, 49]. Recently, Crampton and Loizou [11, 12] introduced the notion of administrative scope and an RBAC administration scheme based on it.

An administrative scheme in conjunction with the representation for an RBAC state naturally lends itself to the safety question in RBAC. The work that is closest to this paper is such work on safety and security analysis in RBAC. Li and Tripunitara [35] studied security analysis for two particular RBAC schemes derived from ARBAC97 [49]: AATU (Assignment And Trusted Users) and AAR (Assign And Revocation), both of which are sub-schemes of the URA97 scheme [49]. The main results in [35] are that security analysis in AATU and AAR are intractable ($\mathbf{NP}$-hard) in general, but can be solved in polynomial time for semi-static queries. The intractability results there are consequences of the fact that a query may be able to encode an arbitrary boolean formula. The techniques used to establish tractable results was to reduce the problem to security analysis in the RT family of trust-management languages [34]. We observe that neither AATU nor AAR allows negative preconditions or constraints. We have shown that URA-RC-SAP with these restrictions are solvable in quadratic time and given direct algorithm for solving them. We point out, even though the role-containment queries are special cases of semi-static queries, our two tractable cases do not follow from results in [35], because AATU does not allow revocation and AAR does not allow trusted users. In essence, the results in [35] deal with very simple state transition rules but sophisticated queries. In this paper, we consider simple queries, but sophisticated state transitions. Since RT [33] is monotonic, it is unclear how to extend the techniques in [35] to deal with negative preconditions or constraints. Li and Tripunitara [35] explicitly mentioned dealing with negative preconditions and constraints as an open problem.

Koch et al. have proposed an RBAC scheme based on a graph-based formalism [29, 31] and have demonstrated that safety is decidable in a sub-scheme [30]. However, the decidable fragment of the graph-based formalism [30] does not allow negative application conditions, which are used to specify negative preconditions in assignment rules in the graph-based formalism for RBAC [29, 31]. Therefore, the decidability result applies only to the subcase without negative preconditions or mutual exclusion constraints. Furthermore, in [30], it has only been shown that safety is decidable in this case; no concrete computational complexity result is given in [30]. The proof shows that the search space is finite; however, searching the space likely takes exponential time. We show that for the case that can be modeled in the decidable fragment, namely, without negative precondition or constraints, URA-RC-SAP is decidable in quadratic time. For cases with negative preconditions and/or constraints, we have given precise computational complexities for them.

Some work related to safety in access control (e.g., [54]) refers to the work by Crampton [10] and Munawer and Sandhu [41] to claim that safety is undecidable in the ARBAC97 scheme. We point out that the undecidability results in Crampton [10] and Munawer and Sandhu [41] are not about the ARBAC97 scheme. The scheme considered by Crampton [10] adds two new features to ARBAC97. One is to allow changes to the CA and CR relations. (Sandhu et al. [49] state specifically that it is assumed that in an ARBAC97 system, these relations are static and may be changed only by (a trusted) chief security officer.) The other is to allow a state-change rule to include an arbitrary command specified using a construct similar to that proposed by Harrison et al. [22]. Such constructs do not exist in ARBAC97. Munawer and Sandhu [41] present a simulation of the Augmented Typed Access Matrix (ATAM) scheme [3] in a particular RBAC scheme that has similar features as those in Crampton [10].

6 Conclusion and Future Work

We have formalized classes of security analysis and insider threat assessment problems in the context of RBAC. We have shown that the URA-SAP and URA-AITAP are $\mathbf{PSPACE}$-complete in the general case and that a number of special cases of the problems are $\mathbf{NP}$-complete. We have also shown that model checking is a promising approach to solve these problems. In the future we plan to look at SAP and AITAP with more sophisticated queries and other administration schemes.

References

[1] P. Ammann and R. S. Sandhu. Safety analysis for the extended schematic protection model. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 87–97, May 1991.

[2] P. Ammann and R. S. Sandhu. The extended schematic protection model. *Journal of Computer Security*, 1(3-4):335–383, 1992.

[3] P. Ammann and R. S. Sandhu. Implementing transaction control expressions by checking for absence of access rights. In *Proceedings of the Eighth Annual Computer Security Applications Conference (ACSAC)*, Dec. 1992.

[4] ANSI. American national standard for information technology – role based access control. ANSI INCITS 359-2004, Feb. 2004.

[5] R. Awischus. Role based access control with the security administration manager (SAM). In *Proceedings of the second ACM workshop on Role-based access control table of contents (RBAC '97)*, pages 61–68, 1997.

[6] R. W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 116–132, May 1990.

[7] T. Budd. Safety in grammatical protection systems. *International Journal of Computer and Information Sciences*, 12(6):413–430, 1983.

[8] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, Jan. 1996.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[10] J. Crampton. *Authorizations and Antichains*. PhD thesis, Birbeck College, University of London, UK, 2002.

[11] J. Crampton and G. Loizou. Administrative scope and role hierarchy operations. In *Proceedings of Seventh ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 145–154, June 2002.

[12] J. Crampton and G. Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security*, 6(2):201–231, May 2003.

[13] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[14] D. F. Ferraiolo, J. A. Cuigini, and D. R. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th Annual Computer Security Applications Conference (ACSAC'95)*, Dec. 1995.

[15] D. F. Ferraiolo and D. R. Kuhn. Role-based access control. In *Proceedings of the 15th National Information Systems Security Conference*, 1992.

[16] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, Apr. 2003.

[17] D. F. Ferraiolo, R. S. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, Aug. 2001.

[18] M. R. Garey and D. J. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[19] G. S. Graham and P. J. Denning. Protection — principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429. AFIPS Press, May 16–18 1972.

[20] T. X. R. Group. The XSB programming system. http://xsb.sourceforge.net/.

[21] M. A. Harrison and W. L. Ruzzo. Monotonic protection systems. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 461–471. Academic Press, Inc., 1978.

[22] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, Aug. 1976.

[23] S. Jha, O. Sheyner, and J. M. Wing. Two formal analysis of attack graphs. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002)*, June 2002.

[24] A. K. Jones, R. J. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *17th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 33–41, October 1976.

[25] G. Karjoth. The authorization model of Tivoli Policy Director. In *Proceedings of the 17th Annual Computer Security Applications Conference*, pages 319–328, Dec. 2001.

[26] A. Kern. Advanced features for enterprise-wide role-based access control. In *Proceedings of the 18th Annual Computer Security Applications Conference*, pages 333–343, Dec. 2002.

[27] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett. Observations on the role life-cycle in the context of enterprise security management. In *Proceedings of the Seventh ACM symposium on Access control models and technologies (SACMAT 2002)*, pages 43–51, 2002.

[28] A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 3–11, June 2003.

[29] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A formal model for role-based access control using graph transformation. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 122–139, Oct. 2000.

[30] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Decidability of safety in graph-based models for access control. In *Proceedings of the Seventh European Symposium on Research in Computer Security (ESORICS 2002)*, pages 229–243. Springer, Oct. 2002.

[31] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A graph-based formalism for RBAC. *ACM Transactions on Information and System Security*, 5(3):332–365, Aug. 2002.

[32] B. W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM Operating Systems Review, 8(1):18-24, Jan 1974.

[33] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[34] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM*, 52(3):474–514, May 2005. Preliminary version appeared in *Proceedings of 2003 IEEE Symposium on Security and Privacy*.

[35] N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 126–135, June 2004.

[36] R. J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[37] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[38] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[39] D. McPherson. Role-based access control for multi-tier applications using authorization manager. *http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/management/athmanwp.mspx*.

[40] R. Motwani, R. Panigrahy, V. A. Saraswat, and S. Ventkatasubramanian. On the decidability of accessibility problems (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 306–315. ACM Press, May 2000.

[41] Q. Munawer and R. S. Sandhu. Simulation of the augmented typed access matrix model (ATAM) using roles. In *Proceedings of INFOSECU99 International Conference on Information and Security*, 1999.

[42] NuSMV: a new symbolic model checker. *http://afrodite.itc.it:1024/ nusmv/*.

[43] X. Ou, S. Govindavajhala, and A. W. Appel. MulVAL: A logic-based network security analyzer. In *Proceedings of the 14th USENIX Security Symposium*, Aug. 2005.

[44] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.

[45] R. S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating systems. *Journal of the ACM*, 35(2):404–432, 1988.

[46] R. S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 122–136. IEEE Computer Society Press, May 1992.

[47] R. S. Sandhu and V. Bhamidipati. Role-based administration of user-role assignment: The URA97 model and its Oracle implementation. *Journal of Computer Security*, 7, 1999.

[48] R. S. Sandhu, V. Bhamidipati, E. Coyne, S. Ganta, and C. Youman. The ARBAC97 model for role-based administration of roles: preliminary description and outline. In *Proceedings of the Second ACM workshop on Role-based access control (RBAC 1997)*, pages 41–50, Nov. 1997.

[49] R. S. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based aministration of roles. *ACM Transactions on Information and Systems Security*, 2(1):105–135, Feb. 1999.

[50] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[51] A. Schaad, J. Moffett, and J. Jacob. The role-based access control system of a European bank: A case study and discussion. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 3–9. ACM Press, 2001.

[52] F. Schneider, editor. *Trust in Cyberspace*. National Academy Press, 1999.

[53] O. Sheyner, J. W. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, 2002.

[54] J. A. Solworth and R. H. Sloan. A layered design of discretionary access controls with decidable safety properties. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 2004.

[55] M. Soshi. Safety analysis of the dynamic-typed access matrix model. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 106–121. Springer, Oct. 2000.

[56] M. Soshi, M. Maekawa, and E. Okamoto. The dynamic-typed access matrix model and decidability of the safety problem. *IEICE Transactions on Fundamentals*, E87-A(1):190–203, Jan. 2004.

[57] N. S. Telecommunications and I. S. S. Committee. The insider threat to U.S. government information systems, July 1999.

[58] T. C. Ting. A user-role based data security approach. In C. Landwehr, editor, *Database Security: Status and Prospects. Results of the IFIP WG 11.3 Initial Meeting*, pages 187–208. North-Holland, 1988.

A Details of the Example from Section 2.4

This appendix gives details of the example that we discussed in Section 2.4. We use RH_{branch} to denote the role-hierarchy, and CA_{branch}, CR_{branch} and CO_{branch} to denote the administrative relations for role assignment, revocation and mutually exclusive role constraints respectively (as shown in Figure 4). The `Loans` department has the Loan Officer role, for which we have two specializations, Commercial Loan Officer and Personal Loan Officer. As the entry in CO_{branch} indicates, a user cannot be a member of both Commercial Loan Officer and Personal Loan Officer. The `Retail` department has the roles Cashier and Teller, and the role Retail Manager is senior to both those roles. The role Accountant does not fall in either department. The roles Admin_H, Admin_L and Admin_R are the administrative roles for the `Human Resources`, `Loans` and `Retail` department respectively. As the entry in CO_{branch} indicates, a user can be a member of at most one of those administrative roles. An additional entry in CO_{branch} asserts that a user can be a member of at most two out of the four roles, Cashier, Teller, Accountant and Loan Officer. Thereby, the bank limits the maximum number of (functional) roles of which a user can be a member. This provides some added guarantee that a user will not have unrestricted access to resources.

The bank prefers that a user, other than someone that is assigned to Retail Manager, not be a member of more than one functional roles. However, during peak hours, the `Retail` department has a heavy work-load, and therefore members of Accountant may occasionally act as cashiers or tellers. An entry in CA_{branch} permits an administrator that is a member of Admin_R to assign someone that is a member of Accountant, but not Loan Officer, to any of the roles in the `Retail` department. The bank also allows some cashiers to act as personal loan officers. However, this must be controlled tightly to avoid wrongdoing on the part of employees. Consider the query, q_{branch} = "$Bob \in \{\text{Cashier, Personal Loan Officer}\}$", which we use to denote that the user Bob is a member of both the Cashier and Personal Loan Officer roles. We would now like to ensure that a user can become a member of both Cashier and Personal Loan Officer only

$$RH_{branch} = \{(\mathsf{Accountant}, \mathsf{Employee}), (\mathsf{Cashier}, \mathsf{Employee}), (\mathsf{Teller}, \mathsf{Employee}), (\mathsf{Retail\ Manager}, \mathsf{Cashier}), (\mathsf{Retail\ Manager}, \mathsf{Teller}) (\mathsf{Commercial\ Loan\ Officer}, \mathsf{Loan\ Officer}), (\mathsf{Personal\ Loan\ Officer}, \mathsf{Loan\ Officer})\}$$

$$CA_{branch} = \{\langle \mathsf{Admin\_H}, true, \mathsf{Employee}\rangle, \langle \mathsf{Admin\_H}, \mathsf{Employee} \wedge \neg \mathsf{Loan\ Officer}, \mathsf{Accountant}\rangle, \langle \mathsf{Admin\_R}, \mathsf{Accountant} \wedge \neg \mathsf{Loan\ Officer}, \{\mathsf{Cashier}, \mathsf{Teller}, \mathsf{Retail\ Manager}\}\rangle, \langle \mathsf{Admin\_L}, \mathsf{Employee} \wedge \neg \mathsf{Accountant}, \{\mathsf{Commercial\ Loan\ Officer}, \mathsf{Personal\ Loan\ Officer}, \mathsf{Loan\ Officer}\}\rangle\}$$

$$CR_{branch} = \{\langle \mathsf{Admin\_H}, \{\mathsf{Employee}, \mathsf{Accountant}\}\rangle, \langle \mathsf{Admin\_L}, \{\mathsf{Commercial\ Loan\ Officer}, \mathsf{Personal\ Loan\ Officer}, \mathsf{Loan\ Officer}\}\rangle, \langle \mathsf{Admin\_R}, \{\mathsf{Retail\ Manager}, \mathsf{Cashier}, \mathsf{Teller}\}\rangle\}$$

$$CO_{branch} = \{\mathsf{smer}\langle\{\mathsf{Commercial\ Loan\ Officer}, \mathsf{Personal\ Loan\ Officer}\}, 2\rangle, \mathsf{smer}\langle\{\mathsf{Cashier}, \mathsf{Teller}, \mathsf{Accountant}, \mathsf{Loan\ Officer}\}, 3\rangle, \mathsf{smer}\langle\{\mathsf{Admin\_H}, \mathsf{Admin\_L}, \mathsf{Admin\_R}\}, 2\rangle\}$$

$$UA_{\gamma} = \{(Alice, \mathsf{Admin\_H}), (Adam, \mathsf{Admin\_L}), (Andy, \mathsf{Admin\_R})\}$$

$$\sigma = assign\,(Alice, Bob, \mathsf{Employee}), assign\,(Alice, Bob, \mathsf{Accountant}), assign\,(Andy, Bob, \mathsf{Cashier}), revoke\,(Alice, Bob, \mathsf{Accountant}) assign\,(Adam, Bob, \mathsf{Personal\ Loan\ Officer})$$

Figure 4: Relations in the example. RH_{branch} gives the role hierarchy. UA_{γ} gives the user-role assignment in the current state. The user Bob is not assigned to any role in γ. The sequence of actions, σ, is the minimal sequence necessary to have Bob be assigned to both the Cashier and the Personal Loan Officer roles.

with the involvement of at least three administrators. Our assignment of users to the three administrative roles Admin_H, Admin_L and Admin_R is shown in Figure 4.

We let $A_T = \emptyset$, $A_I = \{Alice, Adam, Andy\}$ and $k = 2$. Then, the URA-AITAP instance $\langle A_T, A_I, k, \gamma, q_{branch} \rangle$, captures our intent to check whether at least three administrators are needed before someone can become a member of the two roles Cashier and Personal Loan Officer. which are in two different departments. The instance is false, as desired. However, if we set $k = 3$, then the URA-AITAP instance is true, as the three (administrative) users $Alice$, $Adam$ and $Andy$ can cause Bob to become a member of both Cashier and Personal Loan Officer. We present a corresponding sequence of actions as σ in Figure 4.

B Turing Machines

A *Turing Machine* is denoted as $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where Q is the finite set of *states*, Γ is the finite set of allowable *tape symbols*, $B \in \Gamma$ is the *blank* symbol, $\Sigma \subseteq \Gamma - \{B\}$ is the set of *input symbols*, δ is the *next move function* and is a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$, $q_0 \in Q$ is the *start state*, and $F \subseteq Q$ is the set of *final states*. A *nondeterministic Turing machine (NDTM)* allows a finite number of choices for its next move, i.e., δ is a function from $Q \times \Gamma$ to the power set of $Q \times \Gamma \times \{L, R\}$. The first definition describes a deterministic Turing machine (DTM).

The *language accepted* by M (denoted by $L(M)$) is the set of words in $\Sigma^\star$ that cause M to enter a final state when placed justified at the left on the tape of M, with M in state q_0, and the tape head of M at the leftmost cell. A language L is accepted by a DTM if and only if it is accepted by a NDTM. A language L is said to be in $DSPACE(S(n))$ if there exists a DTM M accepting L that takes at most $S(n)$ space, where n is the size of the input. Similarly, L is said to be in $NSPACE(S(n))$ if there exists a NDTM M accepting L that takes at most $S(n)$ space. A language L is said to be in $\mathbf{PSPACE}$ if and only if it is in $DSPACE(p(n))$ for some polynomial p (L is accepted by a DTM that takes space polynomial in the size of the input). We refer the reader to [44] for more details on these and related concepts.

C Proof of Lemma 4

We show that URA-RC-SAP is $\mathbf{PSPACE}$-hard by a reduction from the membership problem for linear bounded automata (LBA), which is known to be $\mathbf{PSPACE}$-complete. A LBA is a restricted form of a Turing machine. It differs from a Turing machine in that while the tape is initially considered infinite, only a finite contiguous portion whose length is a linear function of the length of the initial input can be accessed by the read/write head.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be any LBA that uses at most $p(n)$ space, where n is the size of the input and p is a degree-1 polynomial. (We assume that the polynomial $p(n)$ is know. However, it suffices that there exists a polynomial time algorithm to compute $p(n)$ given n.) We construct an RBAC system whose start state is $\gamma = \langle UA, PA, RH, CA, CR, CO \rangle$ and a query $u \in r$ corresponding to the DTM M so that there is an accepting computation in M on an input x if and only if in the RBAC system there is a sequence of RBAC states from γ (which corresponds to input x) to γ' in which $u \in r$ is true.

In our construction there are two users u_0 and u. There is a special role ra (the reader can think of this as the administrative role) and $(u_0, ra) \in UA$. All the assigning and revoking of roles will be performed by u_0. The special user u_0 can remove users from all roles, i.e. CR is equal to $\{\langle ra, R \rangle\}$, where R consists of all the roles introduced in the construction except for ra.

Encoding TM configurations. For each state $q \in Q$ and $1 \leq i \leq p(n)$ we introduce a role $r_{i,a}$. For each state $q \in Q$, $1 \leq i \leq p(n)$, and symbol a in Σ we introduce a role $r_{q,i,a}$. These roles are used to represent the configuration of the TM machine M. Additional roles will be introduced later to simulate transitions in M. If u is assigned to the role $r_{i,a}$ then the i-th cell contains a, and the tape head is not on cell i. For $1 \leq i \leq p(n)$, u assigned to role $r_{q,i,a}$ indicates that the tape head is on cell i, M is in state q, and the i-th cell contains a. We add the following mutual-exclusion constraints to CO to maintain the integrity of the encoding. We only add a polynomial number of constraints. We use (r_1, r_2) as a shorthand for the mutual-exclusion constraint that a user cannot be members of of both r_1 and r_2 (this is equivalent to $\mathsf{smer}\langle\{r_1, r_2\}\rangle, 2\rangle$ in our earlier notation).

- M can only be in one state at a time. We add mutual exclusion constraints of the form $(r_{q,i,a},\ r_{q',i,a})$ for all $q \neq q'$, i in the range $[1, \cdots, p(n)]$ and $a \in \Sigma$.
- The tape head can only point to one location. We add mutual-exclusion constraints of the form $(r_{q,i,a},\ r_{q,j,a'})$ for all $i \neq j$, and for all q, a, a'.
- One location cannot both have the head and not have the head at the same time. We also add mutual-exclusion constraints of the form $(r_{i,a},\ r_{q,i,a})$.
- Each tape cell can only contain one symbol. Therefore, we add constraints of the form $(r_{i,a}, r_{i,a'})$ and $(r_{q,i,a},\ r_{q,i,a'})$ for all i, $a \neq a'$, and $q \in Q$.

Encoding the initial configuration. Assume that M starts in initial state q_0, the first n cells of the tape contain $a_1, \cdots, a_n$ (where $a_i \in \Gamma$), the rest of tape cells consist of blank symbols B, and the tape head points to the first cell. The initial state of the RBAC system has user u in role $r_{q_0,1,a_1}$, r_{i,a_i} for $2 \leq i \leq n$, and $r_{j,B}$ for $n+1 \leq j \leq p(n)$.

Encoding the halting states. We use one role r_{target} to be used in the query, and add the following tuples to CA:

- For each i in the range $[1, \cdots, p(n)]$, $a \in \Sigma$, and accepting state q, add $\langle ra,\ r_{q,i,a},\ \{r_{target}\}\rangle$

This ensures that the TM M enters an accepting state if and only u can be assigned to r_{target}. The query in the URA-RC-SAP instance we are constructing is $u \in r_{target}$.

Encoding the next-move function: During each transition in M, two revocations and two assignments need to be done to ensure that the next configuration is correctly represented by the role memberships. We need to make these changes transactional. Therefore, we need to introduce some control roles. We will first present the construction and then explain how a state transition occurs.

We first introduce two roles r_b and r_c. Initially, u is assigned to r_b but not r_c. The following tuple is added to CA:

- $\langle ra,\ r_c,\ \{r_b\}\rangle$

Suppose $\delta(q, a)$ is equal to (q', a', L) (the case when $\delta(q, a) = (q', a, R)$ is similar). The transition $\delta(q, a) = (q', a', L)$ is modeled by doing the following.

- For each $2 \leq i \leq p(n)$, add two roles $r^{lb}_{i,q',a'}$ and $r^{lc}_{i,q',a'}$. Initially, u is not assigned to any of these roles. We add the following mutual exclusion constraints:
  - For each $2 \leq i \leq p(n)$, add constraints of the form $(r_b,\ r^{lc}_{i,q',a'})$.
  - For each $2 \leq i \leq p(n)$, add constraints of the form $(r^{lb}_{i,q',a'},\ r_c)$.
- For each $2 \leq i \leq p(n)$, and each $a_\ell \in \Sigma$, add a role r^{ld}_{i,q',a_ℓ}. We add the following mutual exclusion constraints:
  - For each $a \leq i \leq p(n) - 1$, add constraints of the form $(r^{ld}_{i,q',a'},\ r_b)$.
- We then add the following tuples to the relation CA:
  1. For all $2 \leq i \leq p(n)$ we add $\langle ra,\ r_{q,i,a} \cap r_b,\ \{r^{lb}_{q',i,a'}\}\rangle$.
  2. For all $2 \leq i \leq p(n)$ we add $\langle ra,\ r^{lb}_{q',i,a'},\ \{r^{lc}_{q',i,a'}\}\rangle$.
  3. For all $2 \leq i \leq p(n)$ we add $\langle ra,\ r^{lc}_{q',i,a'},\ \{r_{i,a'}\}\rangle$.
  4. For all $2 \leq i \leq p(n)$ for each $a_\ell \in \Sigma$ we add $\langle ra,\ r^{lc}_{q',i,a'} \cap r_{i,a'} \cap r_{i-1,a_\ell},\ \{r^{ld}_{q',i,a_\ell}\}\rangle$.
  5. For all $2 \leq i \leq p(n)$ for each $a_\ell \in \Sigma$ we add $\langle ra,\ r^{ld}_{q',i,a_\ell},\ \{r_{q',i-1,a_\ell}\}\rangle$.
  6. For all $2 \leq i \leq p(n)$ for each $a_\ell \in \Sigma$ we add $\langle ra,\ r^{ld}_{q',i,a'} \cap r_{q',i-1,a_\ell},\ \{r_c\}\rangle$.

Transitions in the RBAC system: Suppose that at one point of computation of M, the tape head is at position i, the i'th cell contains a, and the current state is q. Then u is in the role $r_{q,i,a}$. The only way to proceed in the RBAC system is use CA tuples added in step 1 to assign u to $r^{lb}_{q',i,a'}$. This can succeed only when u is in r_b. Initially, u is in r_b; and we will show that u can be assigned to r_b after a transition in M has been simulated.

Once u is assigned to r_b and then to $r^{lb}_{q',i,a'}$, the only way to make progress in the RBAC system is to use CA tuples added in step 2 to assign u to $r^{lc}_{q',i,a'}$; however, because of the constraints, one has to revoke u from r_b first.

Once u is assigned to r_b and then to $r^{lc}_{q',i,a'}$, u can be assigned (using tuples added in step 3) to $r_{i,a'}$ (after revoking u from from $r_{q,i,a}$ first, due to the constraints added for tape integrity). To update roles corresponding to the $i-1$'th cell, let a_ℓ be the symbol on the $i-1$'th cell, u can be assigned (using tuples added in step 4) to r^{ld}_{q',i,a_ℓ}. Using tuples in step 5, u can be assigned to $r_{q',i-1,a_ell l}$. Finally, after the tape representation roles have been updated, u can be assigned to r_c. Before doing this, u has to be revoked from all roles of the form $r^{lb}_{i,q',a'}$ because of the constraints.

To make the next state transition, u must be assigned to r_b, which requires u to be revoked from all roles of the form $r^{lc}_{i,q',a'}$ and r^{ld}_{i,q',q_ℓ}, clearing all the intermediate roles used in the simulation.

Summary: An *instantaneous description (ID)* of a TM M is given by the contents of the tape, the position of the head, and the state of M. Given two IDs ID_1 and ID_2, we say that $ID_1 \rightarrow_M ID_2$ if ID_2 follows from ID_1 by one move of the M. Given an input x, let $ID_0, \cdots, ID_n$ be a sequence of IDs such that ID_0 corresponds to the input x, $ID_i \rightarrow_M ID_{i+1}$, and ID_n has an accepting state (the sequence is an accepting corresponding to the string). Let γ_0 be the RBAC-state that corresponds to ID_0. Each move of the Turing machine can be emulated by the RBAC scheme in a number of steps; let this constant be c. Therefore, there exists a sequence of RBAC-states $\gamma_0, \cdots, \gamma_{cn}$ such that γ_{ci} encodes ID_i, $\gamma_i \rightarrow \gamma_{i+1}$, and in γ_{cn} u is in a role $r_{q,i,a}$ such that $q \in F$. And then u can be assigned to r_{target}. Hence an accepting computation in M has a corresponding sequence in the RBAC system.

Our discussion of state transitions in the RBAC system above show that state changes in the RBAC systems correspond to computations. If a user u is assigned role $r_{q,i,a}$ in some state in the RBAC system, there is a corresponding computation in the TM M that reaches state q and has the head on the i'th position with the symbol a in the i'th position.

Therefore, there is an accepting computation in TM on an input x if and only if in there is a sequence of RBAC-states from γ (which corresponds to input x) to γ_m, where u is in role r_{target}. This proves the result. □

D Proof for Lemma 7

Proof. We reduce 3SAT to the URA-RC-SAP, which proves that URA-RC-SAP is NP-hard. Let $f = c_1 \wedge \cdots \wedge c_m$ be an instance of 3SAT, and let $p_1, \ldots, p_n$ be the propositional variables in f. We associate a role, r_f, with the formula f. We now construct an instance of URA-RC-SAP [CA (conjunctive), $CR = \{\ \}$, $CO = \{\ \}$] so that a user becomes a member of r_f if and only if f is satisfiable.

In addition to f, $\mathcal{R}$ contains roles $c_1, \ldots, c_m$ corresponding to the clauses and roles $p_1, \ldots, p_n$ corresponding to the propositional variables, plus a role t that will be used for signaling, as will be explained shortly. We construct CA so as to allow u to be assigned to any combination of the p_i's. Once this is done, CA will permit u to be assigned to role t, signaling that a second phase has begun in which u can be added to role c_j just in case u's assignment to the p_i's represents a truth assignment that makes clause c_j true. More precisely, $CA = \{\langle a, \neg t, \{p_1, \ldots, p_n\}\rangle, \langle a, \mathit{true}, \{t\}\rangle, \langle a, c_1 \cap \cdots \cap c_m, f\rangle\} \cup \{\langle a, t \cap p_i, \{c_j \mid 1 \leq j \leq m \wedge p_i \text{ appears positively in } c_j\}\rangle \mid 1 \leq i \leq n\} \cup \{\langle a, t \cap \neg p_i, \{c_j \mid 1 \leq j \leq m \wedge p_i \text{ appears negatively in } c_j\}\rangle \mid 1 \leq i \leq n\}$ and $CR = \emptyset$.

We now consider an instance of URA-RC-SAP in which A_T, UA, PA and RH are empty, and show that it is true if and only if f is satisfiable. If the formula f is satisfiable, it is easy to see that a can add u to role f by first adding u to the role p_i if the propositional variable p_i is true in the solution to f, then adding u to t, and then adding u to each c_j and finally to f.

Conversely, if the problem instance is true, then at some point u must be added to role t. Since u cannot be removed from t, u's assignment to the p_i roles at that time enables u to be added to each role c_j. By defining the propositional variables p_i to be true if and only if the role p_i contains u at that time, we get an assignment that makes at least one literal true in each clause c_j. ∎

E Proof for Lemma 8

Proof. We reduce monotone 3SAT to the problem. Monotone 3SAT is a special case of 3SAT where all literals in a clause are either all positive or all negative; monotone 3SAT is known to be **NP**-complete [18].

Let $e = c_1 \wedge \ldots \wedge c_l \wedge \overline{c_{l+1}} \wedge \ldots \wedge \overline{c_n}$ be an instance of monotone 3SAT, where $c_1, \ldots, c_l$ are the clauses with only positive literals, and $\overline{c_{l+1}}, \ldots, \overline{c_n}$ are the clauses with only negative literals. Let $p_1, \ldots, p_k$ be all the propositional variables that appear in e, and each $c_i = p_{i_1} \vee p_{i_2} \vee p_{i_3}$ and each $\overline{c_j} = \neg p_{j_1} \vee \neg p_{j_2} \vee \neg p_{j_3}$. We produce a corresponding URA-RC-SAP instance for $RBAC_{assign,nonegation}$ as follows.

Corresponding to each propositional variable, p_i, we have a role, r_i. We also have a role, r_{c_i}, corresponding to each positive clause c_i in e. In addition, we have the roles r and a. We assign a user u_0 to

a, that is, $\langle u_0, a\rangle \in UA$. The role a is an administrative role, and appears as the first component of every entry in CA. We first add the tuple $\langle a, r_{c_1} \wedge \ldots \wedge r_{c_l}, r\rangle$ to CA. Corresponding to each positive clause $c_i = p_{i_1} \vee p_{i_2} \vee p_{i_2}$, we add the three entries $\langle a, r_{i_1}, r_{c_i}\rangle$, $\langle a, r_{i_2}, r_{c_i}\rangle$, $\langle a, r_{i_3}, r_{c_i}\rangle$ to CA. If c_i has fewer than 3 literals, then we only add such entries to CA that correspond to the literals in c_i. Clearly, the CA so constructed has no negation in the preconditions of its entries, and each precondition is a conjunction of roles. We capture the negative clauses in e using entries in CO. For each negative clause $\overline{c_j} = \neg p_{j_1} \vee \neg p_{j_2} \vee \neg p_{j_3}$, we add the constraint $\mathsf{smer}\langle\{r, r_{j_1}, r_{j_2}, r_{j_3}\}, 4\rangle$. If $\overline{c_j} = \neg p_{j_1} \vee \neg p_{j_2}$ (that is, has only 2 literals), then we add the constraint $\mathsf{smer}\langle\{r, r_{j_1}, r_{j_2}\}, 3\rangle$ to CO, and if $\overline{c_j} = \neg p_{j_1}$ (has only one literal), then we add the constraint $\mathsf{smer}\langle\{r, r_{j_1}\}, 2\rangle$ to CO.

We ensure that the user u that is specified in the query q is not a member of r or any r_{c_i} in the start-state γ. We now assert that there exists a reachable state in which u is a member of r if and only if e is satisfiable. The reason is that the only way u can become a member of r is by u_0 successfully exercising the only entry in CA that has r as the target role (last component of the tuple in CA). This is possible if only if u already satisfies the role-memberships as specified in the precondition, and assigning u to r does not violate any of the entries in CO. More formally, for the "if" part, assume that e is satisfiable. Then there is some truth-assignment that makes e true. We use the truth-assignment as the user-role assignment in γ for u. That is, if the propositional variable p_i is true in the truth-assignment that makes e true, then $\langle u, r_i\rangle \in UA$. Now, u_0 will be able to assign u to a role r_{c_i} if and only if $\langle u, r_{i_1}\rangle \in UA$, $\langle u, r_{i_2}\rangle \in UA$ or $\langle u, r_{i_3}\rangle \in UA$, where $c_i = p_{i_1} \vee p_{i_2} \vee p_{i_3}$, and none of the entries in CO is violated. An entry in CO is violated if and only if u is a member of all roles other than r in the set of the roles in a constraint. This is the case if and only if the corresponding negative clause is false, which is impossible given the assumption that e is satisfiable.

For the "only if" part, assume that there exists a reachable state in which u is a member of r. We use the role-memberships of u in the roles $r_1, \ldots, r_k$ in the start-state γ as our truth-assignment for e. That is, if $\langle u, r_i\rangle \in UA$, then we set the corresponding propositional variable, p_i, to be true. Given that u can eventually be assigned to r, we know that u can be assigned to every r_{c_i} in γ. Therefore, each positive clause is true. Furthermore, given that in the final action, we can assign u to r, we know that no entry in CO is violated. Consider the constraint $smer\langle\{r, r_{i_i}, r_{i_2}, r_{i_3}\}, 4\rangle$. This constraint would disallow u_0 from assigning u to r if and only if u is already a member of all of r_{i_1}, r_{i_2} and r_{i_3}. As this is not the case, we know that every negative clause evaluates to is true. ∎