Policy Analysis for Administrative Role Based Access Control*

Amit Sasturkar
Yahoo! Inc.

Ping Yang
Binghamton University

Scott D. Stoller
Stony Brook University

C.R. Ramakrishnan
Stony Brook University

Abstract

Role-Based Access Control (RBAC) is a widely used model for expressing access control policies. In large organizations, the RBAC policy may be collectively managed by many administrators. Administrative RBAC (ARBAC) models express the authority of administrators, thereby specifying how an organization's RBAC policy may change. Changes by one administrator may interact in unintended ways with changes by other administrators. Consequently, the effect of an ARBAC policy is hard to understand by simple inspection. In this paper, we consider the problem of analyzing ARBAC policies. Specifically, we consider reachability properties (e.g., whether a user can eventually be assigned to a role by a group of administrators), availability properties (e.g., whether a user cannot be removed from a role by a group of administrators), containment properties (e.g., every member of one role is also a member of another role) satisfied by a policy, and information flow properties. We show that reachability analysis for ARBAC is PSPACE-complete. We also give algorithms and complexity results for reachability and related analysis problems for several categories of ARBAC policies, defined by simple restrictions on the policy language. Some of these results are based on the connection we establish between security policy analysis and planning problems in Artificial Intelligence.

1 Introduction

Background. Role-Based Access Control (RBAC) [SCFY96] is a well known and widely used model for expressing access control policies. At a high level, an RBAC policy specifies the roles to which each user has been assigned (the user-role assignment) and the permissions that have been granted to each role (the permission-role assignment). Users may perform multiple roles in an organization. For instance, in a university setting, a teaching assistant (TA) for a course may be enrolled in other courses at the same time. That person has at least two distinct roles in the university: TA and student. Permissions are associated with these roles; for example, a student can access only her assignments and grades, while a TA can access assignments submitted by students in the course. Expressing access control policy using roles eases specification and management of policies, especially in large organizations.

*Most of this work was done while all of the authors were at Stony Brook University. Contact author's address: Ping Yang, Dept. of Computer Science, Binghamton University, Binghamton, NY, 13902. Email: pyang@cs.binghamton.edu. Phone: (607)-777-5697. Fax: (607) 777-4729 This work was supported in part by NSF under Grants CCR-0205376 and CCR-0311512, and by ONR under grant N00014-04-1-0722.

The RBAC policy in a large organization may be collectively managed by many administrators. For instance, a department manager may have authority to determine who is a TA, while the registrar's office determines who is a student. Thus, there is a need to specify the authority of each administrator. Administrative RBAC'97 (ARBAC97) [SBM99] is a model for expressing such policies. At a high level, an ARBAC policy is specified by sets of rules, including *can_assign* rules that specify the roles to which an administrator may assign a user, and under what conditions, and *can_revoke* rules that specify the roles from which an administrator may remove a user, and under what conditions. The conditions may be positive or negative. For instance, a *can_assign* rule can be used to specify that a department chair may appoint a user as a TA only if the user is already a student and is not a research assistant. In short, an ARBAC policy defines administrative roles, and specifies how members of each administrative role can modify the RBAC policy.

The Problem. It is often hard to understand the effect of an ARBAC policy by simple inspection. For instance, consider a *can_assign* rule for a department manager that specifies that (1) only students may be appointed as TAs, and (2) a student in a class cannot be appointed as a TA of the same class. Thus assignment of a user to the TA role is governed by both a positive precondition (1), and a negative precondition (2). At first glance it appears as though this ensures that a student in a class cannot be the TA for that class. However, this desired condition may not hold: the registrar's policy for assigning a student role in a course might check only the student's registration status and not include conditions regarding TA-ship. This policy would allow the registrar to add someone to a class after that person's appointment as a TA for that class by the department manager. This example illustrates that changes to the RBAC policy by one administrator may interact in unintended ways with changes by other administrators. The ARBAC policy should be designed to prevent such unexpected interactions. In large organizations with many roles (*e.g.*, [SMJ01] describes a European bank's policy with over 1000 roles) and many administrative domains, understanding the ARBAC policy's implications for such interactions may be difficult.

Analysis of security policies has been long recognized as an important problem, *e.g.*, [HRU76, LS77, San88, San92, SM02, JR04, LT06, LMW05]. In a role-based policy framework, a natural analysis problem is to check potential role membership. The *reachability* (or *safety* [HRU76]) problem asks whether a given user u is a member of a given role r in any policy reachable from the initial (i.e., current) policy by actions of a given set of administrators. The *availability* [LMW05] problem asks whether a given user u is a member of a given role r in all policies reachable from the initial policy by actions of a given set of administrators. Another natural analysis problem is *containment* [LMW05]. For example, role-role containment problem asks whether every member of a given role r_1 is also a member of a given role r_2 in some (or all) reachable policies.

Contributions. We define miniARBAC, an ARBAC model based closely on ARBAC97 [SBM99]; the main differences are that miniARBAC allows preconditions for revocation and allows explicit specification of static mutually-exclusive role (SMER) constraints (also called static separation of duty constraints). We show that the reachability problem (abbreviated as "RE") for miniARBAC is PSPACE-complete by relating it to a PSPACE-complete planning problem in Artificial Intelligence (AI). To the best of our knowledge, our work is the first to study the connection between security policy analysis and the well-studied area of planning in AI. This complexity result motivates us to consider restrictions on the policies and two variants of the reachability problem. Our goals are to better understand the intrinsic complexity of the problem and to identify tractable cases of practical interest.

We consider the following restrictions on policies:

1. $\overline{N}$: no negative preconditions.
2. $\overline{EN}$: no explicit negative preconditions; negative preconditions for role assignment may occur only in the form of static mutually-exclusive role (SMER) constraints [LTB07] (sometimes called separation of duty constraints), each of which specifies that a user cannot simultaneously be a member of two given roles;
3. $\overline{CR}$: every role can be revoked unconditionally; and
4. $\overline{D}$: disjunction is not used in the policy's overall conditions for assignment or revocation of a role. The precondition in a single rule never contains explicit disjunction, so this restriction actually requires that there is at most one assignment rule and one revocation rule per role.

We expect that most ARBAC policies satisfy one of these restrictions on negation. Moreover, while role assignments typically have preconditions, role revocations typically do not, and considering unconditional revocation of all roles is sufficient for analysis of policies designed primarily to ensure safety (as opposed to availability). The restriction on disjunction is motivated by results for AI planning that show that this restriction (called post-uniqueness in the planning literature), in combination with other restrictions, can reduce the complexity of planning [BK91, BN95].

We also consider two variants of the Reachability problem. One is Bounded Reachability (BRE): is the goal of adding the given user to the given role reachable using at most a given number of administrative operations? The other is Existence of a Polynomial-size Plan (EPP) for a class of policies: is every reachable goal reachable using a sequence of operations whose length is polynomial in the size of the policy? We consider these variants of reachability analysis primarily to shine some indirect light on the difficulty of reachability analysis for problem classes for which the complexity of reachability analysis itself is unknown.

Restricting attention to plans or paths of bounded length is a common idea in bounded-length planning and bounded model checking [BCC$^+$03]. Generally, the bounded version of the problem is not of intrinsic interest, but it provides a way to partially or indirectly attack the original (unbounded) problem when the latter is too difficult. This is the case in our work as well. We establish complexity results for BRE for some problem classes for which there are no known complexity results for reachability.

Existence of a polynomial-size plan is a natural property to consider. In addition to the intuitive connection that reachability is easier when the plans are shorter (and the plans, regarded as potential attacks, are more practical), some specific complexity results for reachability follow from complexity results for EPP. In particular, if EPP for a problem class is true, then Reachability (*i.e.*, the reachability decision problem) for that problem class is in NP; and if EPP is false for a problem class, then the plan generation problem for that problem class is not in P.

We explore the complexity of reachability analysis, the above variants of it (BRE and EPP), and availability analysis under combinations of these restrictions on policies. Some of our results are corollaries of existing results for AI planning, but most of our results are new. In many cases, the restricted analysis problem still has high computational complexity. Perhaps the most revealing of these results is the non-existence of polynomial-size plans for a number of classes of ARBAC policies. This reflects the difficulty of understanding the implications of ARBAC policies.

In summary, the main contributions of this paper are to:

- establish that reachability analysis for general miniARBAC policies is PSPACE-complete;
- determine the computational complexity of user-role reachability analysis and the above variants of it (namely, BRE and EPP) for several categories of miniARBAC policies;
- give algorithms for cases where the analysis problem is solvable in polynomial time;
- give algorithms and complexity results for several other analysis problems, including permission-role reachability, user-permission reachability, availability, role-role containment, permission-role containment, and information flow analysis.
- give complexity result for reachability analysis of a variant of miniARBAC that does not require disjointness of administrative roles and regular roles.

Sections 2 and 3 provide an overview of RBAC and ARBAC policies. Sections 4, 5 and 7 formally define the analysis problems and provide algorithms and complexity results for user-role reachability analysis. Other analysis problems are considered in Section 6. The related work and concluding remarks appear in Sections 8 and 9, respectively.

2 Role Based Access Control (RBAC)

The central notion of RBAC is that users are assigned to appropriate roles and roles are assigned appropriate permissions. Thus, a role serves as an intermediary in correlating users with permissions. In this paper, we study policy analysis only for models of RBAC based on [Ame04]. Since the policy analysis queries we support are independent of sessions, we consider simplified ("mini") models that do not support sessions.

The miniRBAC model is based on the core RBAC model [Ame04].

Definition 1: miniRBAC. A *miniRBAC policy* γ is a tuple (U, R, P, UA, PA) where

- U, R and P are finite sets of users, roles, and permissions, respectively. A permission represents authorization to invoke a particular operation on a particular resource.
- $UA \subseteq U \times R$ is the user-role assignment relation. $(u, r) \in UA$ means that user u is a member of role r.
- $PA \subseteq P \times R$ is the permission-role assignment relation. $(p, r) \in PA$ means that members of role r are granted the permission p.

Given $\gamma = (U, P, R, UA, PA)$, define $users_\gamma(r) = \{u \in U : (u, r) \in UA\}$ and $perms_\gamma(r) = \{p \in P : (p, r) \in PA\}$.

Our miniHRBAC model, based on Hierarchical RBAC [Ame04], extends miniRBAC with *role hierarchies*, which are a natural means for structuring roles to reflect an organization's lines of authority and responsibility.

Definition 2: miniHRBAC. A *miniHRBAC policy* γ_h is a tuple $(U, R, P, UA, PA, \succeq)$ where

- U, R, P, UA and PA are as in miniRBAC.
- $\succeq \subseteq R \times R$ is a partial order, called the *role hierarchy*.

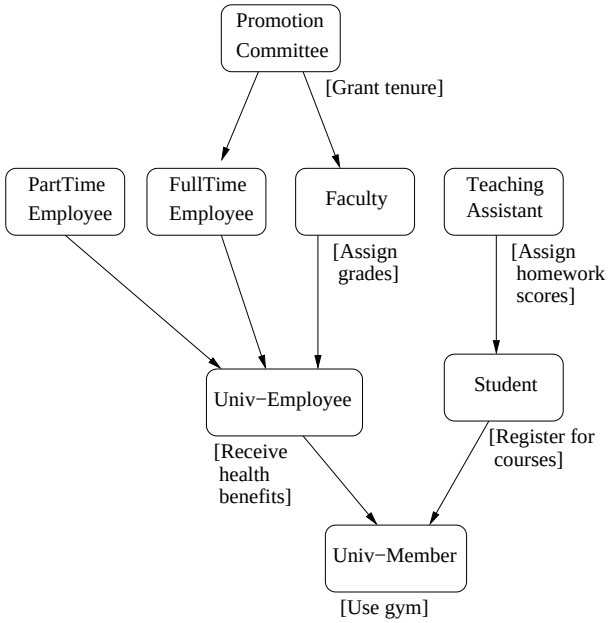

Figure 1: Example of miniRBAC and miniHRBAC policy.

$r_1 \succeq r_2$ means r_1 is senior to r_2; *i.e.*, every member of r_1 is also a member of r_2, and every permission assigned to r_2 is also available to members of r_1. Thus, r_2 inherits all the users of r_1, and r_1 inherits all the permissions of r_2.

Given $\gamma_h = (U, P, R, UA, PA, \succeq)$, we extend the $users_\gamma$ and $perms_\gamma$ functions to account for role hierarchy: $users_{\gamma_h}(r) = \{u \in U : \exists r' \in R.r' \succeq r \wedge (u, r') \in UA\}$, and $perms_{\gamma_h}(r) = \{p \in P : \exists r' \in R.r \succeq r' \wedge (p, r') \in PA\}$. We define $Senior(r) = \{r' \in R : r' \succeq r\}$.

Figure 1 gives a simple example of a miniRBAC and a miniHRBAC policy with 8 roles. Consider the roles `Univ-Member`, `Univ-Employee`, and `Student`. The permissions for each role are shown below the role. Users in the `Univ-Member` role are members of the University and have permission to use University facilities like the gym. The roles `Univ-Employee` and `Student` are senior to the `Univ-Member` role. Thus, members of these roles are also implicitly members of the `Univ-Member` role, and inherit the permission to use the gym.

3 Administrative Role Based Access Control (ARBAC)

Administration of (*i.e.*, changes to) RBAC policies must be carefully controlled. RBAC policies for large organizations may have over a thousand roles and tens of thousands of users. For scalability, it is necessary to distribute the task of administering such large policies, by giving each administrator authority to make specified kinds of changes to specified parts of the policy. This is an access control policy that, for scalability and ease of administration, can profitably be expressed in a role-based manner.

ARBAC97 ("Administrative RBAC") is a model for decentralized administration of RBAC policies [SBM99]. Changes to the ARBAC policy (*e.g.*, granting permissions to administrative roles) are not considered in the ARBAC97 model. This is justified by assuming that only a small

group of fully trusted administrators are allowed to modify the ARBAC policy.

In typical ARBAC policies, there is a single top level administrator role, called the Senior Security Officer (SSO) which is the principal administrator of the RBAC policy and which establishes the ARBAC policy. The SSO partitions the organization's RBAC policy into different security domains, each of which is administered by a different Junior Security Officer (JSO). For example, there may be a JSO role for each department. The ARBAC policy specifies the permissions assigned to each JSO role; for example, to which normal roles and under what conditions can members of a JSO role assign users. SSOs can design ARBAC policies that enforce global constraints on the RBAC policy by allowing JSOs to make only changes that are consistent with the constraints.

There are three main parts in an ARBAC97 policy: the user-role administration (URA) policy, the permission-role administration (PRA) policy, and the role-role administration (RRA) policy. They control changes to the user-role assignment UA, the permission-role assignment PA, and the role hierarchy respectively. In this paper, we consider a slightly modified version of ARBAC97, which we call miniARBAC. A miniARBAC policy consists of URA, PRA, and RRA policies, defined below.

URA policy. The URA policy controls changes to the user-role assignment UA. Its specification uses *preconditions* (called prerequisite conditions in [SBM99]) which are conjunctions of *literals*, where each literal is either r or $\neg r$ for some role r. Given a miniRBAC state γ and a user u, u satisfies a precondition $\wedge_i l_i$, denoted $u \models_\gamma \wedge_i l_i$, iff for all i, either l_i is a role r and $u \in users_\gamma(r)$, or l_i is a negated role $\neg r$ and $u \notin users_\gamma(r)$.

Permission to assign users to roles is specified by the $can\_assign \subseteq AR \times C \times R$ relation, where AR is a finite set of administrative roles and C is the set of all preconditions on R. ARBAC97 requires that $AR \cap R = \emptyset$. A $UserAssign(r_a, u, r)$ action specifies that an administrator who is a member of the administrative role r_a adds user u to role r. This action is enabled in state $\gamma = (U, P, R, UA, PA)$ iff there exists $(r_a, c, r) \in can\_assign$ and $u \models_\gamma c$. Upon executing the action, γ is transformed to the state $\gamma' = (U, P, R, UA \cup \{(u, r)\}, PA)$. Note that preconditions are not invariants; if $(r_a, r_1, r_2) \in can\_assign$, then a user u in r_1 and r_2 remains a member of r_2 even if an administrator removes u from r_1. If some role r does not appear in the last component of any tuple in can_assign, then no administrator can add users to r, but r could still appear in the initial role assignment.

Permission to revoke users from roles is specified by the $can\_revoke \subseteq AR \times C \times R$ relation, where AR and C are as above. [SBM99] mentions the option of including preconditions in can_revoke but does not include them in the basic ARBAC97 model. A $UserRevoke(r_a, u, r)$ action specifies that an administrator who is a member of the administrative role r_a removes user u from the membership of role r. This action is enabled in state $\gamma = (U, P, R, UA, PA)$ iff there exists $(r_a, c, r) \in can\_revoke$ and $u \models_\gamma c$. Upon executing the action, γ is transformed to the state $\gamma' = (U, P, R, UA \setminus \{(u, r)\}, PA)$.

PRA policy. The PRA policy controls changes to the permission-role assignment PA. Assignment of a permission p to a role r by an administrator in administrative role r_a is achieved by the $PermAssign(r_a, p, r)$ action and is controlled by the can_assign_p relation. Similarly, revocation of a permission p from a role r by an administrator in administrative role r_a is achieved by the $PermRevoke(r_a, p, r)$ action and is controlled by the can_revoke_p relation. These relations are defined in the same way as the can_assign and can_revoke relations above, except that users are replaced with permissions.

RRA policy. The RRA policy controls changes to the role hierarchy. Actions that modify the role

hierarchy include *CreateRole*(r, r_p, r_c), which creates a new role r with parent r_p and child r_c (the parent and child must be existing roles), *DeleteRole*(r), which deletes role r, *InsertEdge*(r_1, r_2), which inserts an edge from r_1 to r_2 (i.e., makes r_1 senior to r_2), and *DeleteEdge*(r_1, r_2), which deletes the edge from r_1 to r_2. Permissions to execute these actions are controlled by the *can_modify* $\subseteq AR \times 2^R$ relation. The subsets of R are specified using range notation, similar to standard mathematical notation for numeric ranges. A role range has the form (r_1, r_2), $(r_1, r_2]$, $[r_1, r_2)$, or $[r_1, r_2]$, where $r_2 \succeq r_1$. A parenthesis indicates that the endpoint is not included in the range; a square bracket indicates that the endpoint is included in the range. For example, $[r_1, r_2)$ denotes the $\{r \in R \mid r_2 \succ r \succeq r_1\}$, where $r_1 \succ r_2 = r_1 \succeq r_2 \wedge r_1 \neq r_2$. For an administrative role r_a and role range Y of the form (r_1, r_2), $(r_a, Y) \in$ *can_modify* specifies that members of r_a can perform *CreateRole*(r, r_p, r_c), *DeleteRole*(r), *InsertEdge*(r_1, r_2), and *DeleteEdge*(r_1, r_2) provided that r, r_p, r_c, r_1, and r_2 are in Y and that some additional restrictions hold that ensure the changes to the role hierarchy are sensible and safe; for example, deletion of roles that are referenced explicitly in relations in URA, PRA, or RRA (e.g., roles that appear in preconditions in *can_assign*) is prohibited. We adopt these restrictions but omit details of them, because they do not affect our results. Note that the RRA policy makes sense only in the presence of role hierarchy, i.e., when we are considering changes to miniHRBAC policies not miniRBAC policies.

Static Mutually Exclusive Roles (SMER) constraints. miniARBAC also includes a set of 1-2 SMER constraints [LTB07] which are used to enforce separation of duty [CW87]. SMER constraints are called Static Separation of Duty (SSoD) constraints in [Ame04]; we follow [LTB07] in referring to them as SMER constraints. A 1-2 SMER constraint is an unordered pair of roles $s = \{r_1, r_2\}$ and is satisfied in a state γ, denoted $\gamma \vdash s$, iff $users(r_1) \cap users(r_2) = \emptyset$; *i.e.*, the roles r_1 and r_2 do not have any users in common in the RBAC policy γ. γ is said to be valid for a set of SMER constraints S iff $\forall s \in S : \gamma \vdash s$. SMER constraints specifying disjointness of permissions assigned to two roles could also be allowed, but it is unclear whether such constraints would be useful in practice.

Note that a SMER constraint $\{r_1, r_2\}$ can be expressed by including $\neg r_1$ in the precondition of all *can_assign* rules for r_2 and roles senior to r_2, and vice versa. We choose to explicitly represent SMER constraints (and not force them to be specified in the URA model using negation), because this allows us to investigate whether policy analysis is easier if negation is used only in the form of SMER constraints, which is a common case.

Definition 3: miniARBAC A *miniARBAC policy* is a tuple $\psi = (AR, \succeq_a, can\_assign, can\_revoke, can\_assign\_p, can\_revoke\_p, can\_modify, SMER)$, where AR is the set of administrative roles, the administrative role hierarchy $\succeq_a$ is a partial order on AR, and the other five relations are as defined above.

In summary, the differences between miniARBAC and ARBAC97, aside from notation, are that, in miniARBAC: (1) *can_revoke* and *can_revoke_p* are extended to allow preconditions for revocation, (2) SMER constraints can be explicitly specified, and (3) the last component of *can_assign*, *can_revoke*, *can_assign_p*, and *can_revoke_p* is a single role, instead of a role range.

A miniARBAC policy specifies a transition relation between miniRBAC policies, which we refer to as "states". We denote a transition by $\gamma \xrightarrow{act}_{\psi} \gamma'$ where *act* is one of the administrative actions specified above (namely, *UserAssign*, *UserRevoke*, *PermAssign*, *PermRevoke*, *CreateRole*, *DeleteRole*, *InsertEdge*, or *DeleteEdge*), and γ and γ' satisfy the SMER constraints in ψ.

Examples. We present a few example miniARBAC policies that illustrate features of miniARBAC. Consider the miniHRBAC policy of Figure 1.

- **Positive preconditions:** A user can be made member of the `Teaching-Assistant (TA)` role by an administrator in role r_a only if she is already a member of the `Student` role. This policy can be specified by the rule $(r_a, \texttt{Student}, \texttt{TA}) \in \textit{can\_assign}$.
- **Conjunction in preconditions:** A user who is a member of both `Faculty` and `FullTime-Employee` roles can serve on the `Promotion-Committee`. This policy can be specified by the rule $(r_a, \texttt{Faculty} \wedge \texttt{FullTime-Employee}, \texttt{Promotion-Committee}) \in \textit{can\_assign}$, where r_a is an appropriate administrative role.
- **1-2 SMER constraints:** A user can be a member of at most one of the `Faculty` and `Student` roles. This policy can be specified by the constraint set $\textit{SMER} = \{\{\texttt{Faculty}, \texttt{Student}\}\}$.
- **Negative preconditions:** Negative preconditions in the *can_revoke* relation can be used to force role revocations to occur in a particular order. We might have a policy that says that a user can be made member of the `TA` role only if he is already a member of the `Student` role. The policy also requires that when a user ceases to be a `Student` he also ceases to be a `TA`. This policy can be enforced with the following rules : $(r_a, \texttt{Student}, \texttt{TA}) \in \textit{can\_assign}$, and $(r_a, \neg\texttt{TA}, \texttt{Student}) \in \textit{can\_revoke}$. The second rule forces an administrator to revoke the user's `TA` role before revoking his `Student` role.
- **Conditional role revocation:** Recall that miniARBAC, unlike ARBAC97 [SBM99], allows preconditions in role revocation. The policy with negative preconditions described above is also an example of a policy that requires conditional role revocation.

4 Policy Analysis Problems

A miniARBAC policy ψ defines a transition relation between miniRBAC policies and therefore defines a transition graph. Each vertex of the transition graph is a miniRBAC policy, and each edge is a transition $\gamma \xrightarrow{act}_{\psi} \gamma'$. Usually we are interested in analyzing or restricting the power of a given set A of administrative roles, so we discard edges labeled with actions by administrative roles not in A (recall that the administrative role is the first argument of every action), and ask the following kinds of queries about the resulting graph. Note that A is an implicit parameter of all these queries.

- **User-Role Reachability Analysis:** Given a role r and a user u not in r, can u be added to r (by actions of administrators in administrative roles in A)?
- **Permission-Role Reachability Analysis:** Given a role r and a permission p not granted to r, can p be granted to r?
- **User-Permission Reachability Analysis:** Given a user u and a permission p, does there exist a role r such that p can be granted to r and u can be added to r (*i.e.*, p is granted to u)? (If these two administrative operations can be done at all, they can be done simultaneously,

because miniARBAC does not include any constraints that relate the user-role assignment and the permission-role assignment.)

- **User-Role Availability Analysis [LMW05]:** Given a role r and a member u of r, can u be removed from r ?
- **Permission-Role Availability Analysis [LMW05]:** Given a role r and a permission p granted to r, can p be revoked from r?
- **Role-Role Containment analysis:** Given two roles r_1 and r_2, are the members of r_1 always (*i.e.*, in the policy at every node) a subset of the members of r_2
- **Permission-Role Containment analysis:** Given two roles r_1 and r_2, are the permissions of r_1 a subset of the permissions of r_2 ?

4.1 Simplifying the Problem

These analysis problems can be simplified by eliminating irrelevant aspects of the RBAC and ARBAC policies. This reduces clutter but does not change the algorithmic complexity of the problem. This section details our simplifications for user-role reachability analysis without role hierarchy (i.e., for miniARBAC controlling changes to miniRBAC policies, not miniHRBAC policies). Similar simplifications are possible for other analysis problems.

A *user-role reachability query* Q has the form: Given a user u, a set A of administrative roles, a set *goal* of roles, an initial miniRBAC policy γ, and a miniARBAC policy ψ, can administrators in administrative roles in A, using the administrative permissions granted to those roles by ψ, transform γ to another policy γ' such that u is a member of all roles in *goal* in γ'?

We simplify the problem as follows.

1. **Ignoring permissions:** miniARBAC does not include any constraints that relate the user-role assignment and the permission-role assignment, so the answer to Q is affected only by the user-role assignment relation and the *can_assign*, *can_revoke* and *SMER* components of ψ. The other components of γ and ψ can be ignored when answering Q. This also implies that only the *UserAssign* and *UserRevoke* actions are relevant.

2. **Implicit administrative role:** Administrative roles not in A are assumed to be inactive while administrators in administrative roles in A are trying to reach the goal. Thus, administrative roles not in A have no effect on the answer to Q, so we can remove from ψ every administrative role that is not in A and is not junior to an administrative role in A. We also remove all *can_assign* and *can_revoke* rules for the removed administrative roles. Then Q asks about reachability under actions by all of the (remaining) administrative roles. Thus, there is no need to distinguish these roles from each other, so we can delete their names. In other words, we can assume that there is a single implicit administrative role, and we simplify *can_assign* and *can_revoke* to have the type $C \times R$ (instead of $AR \times C \times R$).

3. **Single user:** The preconditions for $UserAssign(r_a, u, r)$ and $UserRevoke(r_a, u, r)$ depend only on the current role memberships of the target user u. These conditions are independent of the role memberships of other users. Therefore, when answering a query Q about user u, we can remove all other users from the policy. Thus, we can assume there is a single implicit

user, and we can simplify UA to be a subset of R, where $r \in UA$ means that the implicit user is a member of r.

With these simplifications, a (simplified) miniRBAC policy γ is a pair (R, UA) where $UA \subseteq R$, an action is $UserAssign(r)$ or $UserRevoke(r)$, and a (simplified) miniARBAC policy ψ is a triple $(can\_assign, can\_revoke, SMER)$ where can_assign, $can\_revoke \subseteq C \times R$. Note that can_modify is omitted because we are not considering role hierarchy. A reachability query for the simplified policy can be represented as a set $goal$ of roles (since u and A are now implicit). A goal set $goal$ is satisfied in an RBAC policy state $\gamma = (R, UA)$, denoted $\gamma \vdash goal$, iff $goal \subseteq UA$.

Definition 4: Reachability Analysis Problem Instance. A (simplified) user-role reachability analysis problem instance is a 3-tuple $I = (\gamma, goal, \psi)$ where γ is a miniRBAC policy, ψ is a miniARBAC policy, and $goal \subseteq R$ is a goal set.

Given a reachability analysis problem instance, one natural question is whether an RBAC policy state satisfying the goal can be reached starting from the initial state of the instance. A closely related problem is to find a sequence of RBAC policy changes that lead to a state satisfying the goal. Such a sequence of changes is called a *plan*, defined formally below.

Definition 5: Plan. Let $I = (\gamma, goal, \psi)$ be a reachability analysis problem instance. A sequence of actions $act_1, act_2, \ldots, act_n$ where each $act_i \in \{UserAssign(r), UserRevoke(r) : r \in R\}$ is called a *plan* or *solution* for I if $\gamma \overset{act_1}{\rightarrow}_\psi \ \ldots \overset{act_n}{\rightarrow}_\psi \ \gamma'$ and $\gamma' \vdash goal$. The length of the plan is the length of the sequence.

The reachability problem can be expressed in terms of plan existence:

Definition 6: Reachability (RE). Given a problem instance I, the Reachability Problem (RE) is to determine whether or not there exists a plan for I.

Note that we allow the goal to be a set of roles. It might seem sufficient to take the goal to be a single role, because reachability of a goal $\{g_1, \ldots, g_n\}$ can be reduced to reachability of a goal containing a single role g by adding $(g_1 \wedge \ldots \wedge g_n, g)$ to the can_assign relation. However, this transformation cannot always be used, because it adds a can_assign rule with multiple positive preconditions to the policy, and some of our results in Section 5 depend on restrictions on the number of preconditions or the number of positive preconditions in each rule. Also note that the approach of separately checking reachability of $g_1, \ldots, g_n$ does not work, because even if these roles are reachable individually, they might not be reachable simultaneously, *e.g.*, if two of them are involved in a SMER constraint.

The *Reachability* problem is PSPACE-complete in general, as shown below in Section 5. To understand the problem better and identify efficiently solvable cases of practical interest, we consider several subclasses of problem instances by imposing various structural restrictions on the miniARBAC policy and the query. For several of these classes, we can tightly characterize the complexity of the *Reachability* problem. However, there remain a number of classes for which the precise complexity of the problem remains unknown. For such classes, it is interesting to consider a variant of the problem, known as the *Bounded Reachability* problem, defined below.

Definition 7: Bounded Reachability (BRE). Given a problem instance I and an integer k, the Bounded Reachability Problem (BRE) is to determine whether or not there exists a plan for I of length at most k.

Note that BRE might be harder than RE for some problem classes—intuitively, this is because finding a short plan might be more difficult than finding an arbitrary plan. Below, we show that RE is no harder than BRE:

Lemma 1 For any problem class, BRE is at least as hard as RE.

PROOF SKETCH: The proof is by reduction from BRE to RE. Let I be a problem instance of RE. We construct an instance I, k of BRE by choosing a bound k such that there is a plan for I iff there is a plan of length $\leq k$ for I. Let $|R|$ be the number of roles in I. It suffices to choose k to be $2^{|R|}$ because this is an upper bound on the number of reachable states. It is easy to see that there is a plan for I iff there is a plan for I that goes through each state at most once. $O(|R|)$ bits are needed to represent this value of k which is linear in the number of bits in the representation of I. □

When the complexity of BRE and RE are not known with sufficient precision, we can characterize the difficulty of problem instances by considering the lengths of plans for them.

Definition 8: Existence of Polynomial-size Plan (EPP). Given a set S of problem instances, we say that S has the property of existence of a polynomial-size plan (equivalently, we say that EPP is `true` for S) if and only if there is a polynomial f such that for all problem instances $I \in S$, if I has a plan, then I has a plan with length at most $f(|I|)$.

Definition 9: Size of a Problem Instance. The size $|I|$ of a problem instance I is the sum of the sizes of all the sets in it.

When EPP is `false` for a problem instance in a class $\mathcal{C}$, the plan-generation problem for $\mathcal{C}$ is not in P (i.e., not solvable in polynomial time). Note that RE for $\mathcal{C}$ may still be solvable in polynomial time, because RE is a decision problem which does not require explicit generation of a plan. When EPP is `true` for a problem class $\mathcal{C}$, then RE for $\mathcal{C}$ is in NP, because a non-deterministic Turing machine can guess the plan and verify it in polynomial time.

4.2 Classification of Problem Instances

Due to the intractability of reachability analysis for ARBAC in the general case, we consider a variety of restrictions on reachability problem instances. This section defines those restrictions.

We consider four categories of restrictions on ψ. The acronyms for them are summarized in Figure 2.

- **Restricting negation:** We say that ψ uses explicit negation if a negative literal appears in *can_assign* or *can_revoke*, and ψ uses implicit negation if ψ contains a SMER constraint (*i.e.*, $SMER \neq \emptyset$).
  - **No negation ($\overline{N}$):** ψ satisfies the $\overline{N}$ restriction if ψ does not use explicit or implicit negation.
  - **No explicit negation ($\overline{EN}$):** ψ satisfies the $\overline{EN}$ restriction if ψ does not use explicit negation. This restriction is interesting because SMER constraints are more common than other uses of negation.
- **No disjunction ($\overline{D}$):** ψ satisfies the $\overline{D}$ restriction if for every role $r \in R$, there is at most one rule in ψ for assigning/revoking r.

- **Restricting revocation:**
  - **No revocation ($\overline{R}$):** ψ satisfies the $\overline{R}$ restriction if *can_revoke* $= \emptyset$. This implies that once a user is assigned to a role, the user cannot be revoked from the role.
  - **No conditional revocation ($\overline{CR}$):** ψ satisfies the $\overline{CR}$ restriction if for every role $r \in R$, $(\texttt{true}, r) \in$ *can_revoke*. In other words, every role in R can be unconditionally revoked. This restriction is reasonable, especially when considering powerful administrative roles, because revocation is generally not controlled as tightly as role assignment, for example, recall that ARBAC97 does not support preconditions on revocation.
- **Size restrictions:** ψ satisfies $|pre| \leq k$ if $\forall (c, r) \in$ *can_revoke*, the number of literals in c is bounded by k and $\forall (c, r) \in$ *can_assign*, the number of literals in c is bounded by $k - |\{r' : \{r, r'\} \in SMER\}|$ (if $\{r_1, r_2\}$ is a SMER constraint, then $\neg r_1$ is counted as part of every precondition for r_2 in *can_assign*, and vice versa). ψ satisfies $|ppre| \leq k$ if $\forall (c, r) \in$ *can_assign* $\cup$ *can_revoke*, the number of positive literals in c is bounded by k. ψ satisfies $|SMER(r)| \leq k$ if $\forall r \in R$, $|\{r' : \{r, r'\} \in SMER\}| \leq k$. ψ satisfies $|goal| \leq k$ if the size of the goal set is at most k (thus, for a set of problem instances satisfying $|goal| \leq k$, the size of every goal is bounded by k, independent of the number of roles in each problem instance). As we show below, enforcing one or more of these restrictions greatly simplifies the reachability analysis problem.

We also consider a restriction *EI* (empty initial state) on problem instances. A problem instance $(\gamma, goal, \psi)$ satisfies *EI* if the user assignment in γ is the empty set.

A set of restrictions defines a class of reachability analysis problems. For example, the class $[\overline{R}, \overline{D}, |pre| \leq 1]$ includes all problems $(\gamma, goal, \psi)$ where ψ satisfies the $\overline{R}$, $\overline{D}$ and $|pre| \leq 1$ restrictions. When a class has the $\overline{EN}$ restriction (allow SMER constraints, but not explicit negation), the $|ppre| \leq k$ restriction is used instead of the $|pre| \leq k$ restriction, since preconditions contain only positive literals. For each class we consider the RE and BRE problems and check whether EPP is `true` for every problem instance in the class.

5 Complexity Results for User-Role Reachability Analysis Without Role Hierarchy

Figure 2 summarizes our complexity results for user-role reachability analysis without role hierarchy (role hierarchy is considered in Section 7.1). The problem classes, represented by boxes, are divided into four groups, separated by double lines, based on the complexity of RE and BRE. The problem classes are arranged in a hierarchy. An edge from class $\mathcal{C}_1$ to $\mathcal{C}_2$ indicates that $\mathcal{C}_2$ is a specialization of $\mathcal{C}_1$. Thus, every hardness result for $\mathcal{C}_2$ also applies to $\mathcal{C}_1$, and every algorithm for $\mathcal{C}_1$ can be used to solve $\mathcal{C}_2$. When multiple top-level problem classes contain the same restriction, the figure emphasizes this structure using an ellipse representing the restriction with edges to those problem classes. Different line styles are used for outgoing edges of different ellipses, just to increase the visual distinction between them. The notation $Th\langle n \rangle$ means that the result is proved in Theorem n; for example, *Th9* refers to Theorem 9.

Some observations follow.

1. The restriction $|goal| \leq k$ is relevant only in classes that also have the restriction $|pre| \leq 1$. If a problem class $\mathcal{C}$ has the former restriction but not the latter, then given a problem instance

$I = (\gamma, goal, \psi)$ with $|goal| > k$, we can rewrite I to an instance $I' = (\gamma', goal', \psi')$ with $|goal'| = 1$, by introducing new roles in γ and adding rules to ψ for modifying them. For example, if $goal = \{r_1, r_2\}$ and $\mathcal{C}$ has the restriction $|goal| \leq 1$ but not the restriction $|pre| \leq 1$, then introduce a new role r_g, add the rule $(r_1 \wedge r_2, r_g)$ to *can_assign*, and take $goal' = \{r_g\}$. The new problem instance is equivalent to the old instance but satisfies $|goal| \leq 1$ and is still in $\mathcal{C}$.

2. The restriction $\overline{D}$ (no disjunction) makes the *Reachability* problem easier; *Reachability* for $[\overline{R}]$ is NP-complete whereas for $[\overline{D}, \overline{R}]$ it is solvable in polynomial time. Not allowing disjunction in preconditions reduces the number of possible plans for a problem instance, thereby reducing the complexity of the *Reachability* problem.

3. The restriction $\overline{CR}$ (only unconditional revocation) affects what we can say about the *Existence of Polynomial-size Plan* (EPP) problem. EPP for $[\overline{D}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$ is `false` and hence for $[\overline{EN}, |ppre| \leq 1, |goal| \leq k]$ is `false`, implying that a polynomial time algorithm for generating a plan for this problem class does not exist. EPP for $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$ is `true`.

4. The restriction $\overline{R}$ (no revocation) ensures that the answer to the *Existence of Polynomial-size Plan* (EPP) problem is `true`. When role revocation is not allowed, the user can be assigned to a role at most once in any plan. Thus, the length of a plan is at most the number of roles.

5. For most problem classes (*i.e.*, sets of restrictions) we considered, adding the $\overline{EN}$ restriction (allow SMER constraints but not explicit negation) neither lowered the worst-case complexity of RE or BRE nor changed EPP from `false` to `true`. Thus, in general, SMER constraints do not seem to be easier to analyze than explicit use of negation. Intuitively, this may reflect the fact that when negation is allowed, even in restricted forms, the analysis problem loses its monotonic nature. However, there are problem classes for which the effect of adding or removing the $\overline{EN}$ restriction remains unknown. For example, we showed that RE is solvable in polynomial time for the class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$, but the worse-case complexity of RE for the class $[\overline{CR}, |ppre| \leq 1, |goal| \leq k]$ is unknown.

The proof of Theorem 6 provides a polynomial time algorithm for solving *Reachability* (RE) for a problem class that is still general enough to be interesting in practice. Theorems 29 and 19 show that even when three or four restrictions are applied simultaneously, reachability may remain a hard problem, not solvable in polynomial time.

The rest of this section contains proofs for the results in Figure 2. Section 5.1 contains proofs for PSPACE-completeness results. Section 5.2 contains proofs and algorithms for problem classes for which user-role reachability analysis can be solved in polynomial time. Sections 5.3 and 5.4 present proofs for problem classes for which user-role reachability analysis is NP-complete and NP-hard, respectively.

Reachability has been studied in a variety of settings, such as static program analysis and model checking, as well as planning. However, work on propositional planning in AI is particularly relevant to ARBAC policy analysis, because operators considered in planning are similar to RBAC administrative operations. As a result, some complexity results for ARBAC policy analysis, such as Theorems 5 and 14 below, can be derived in a fairly direct way from results for planning, by reductions from planning to policy analysis and *vice versa*.

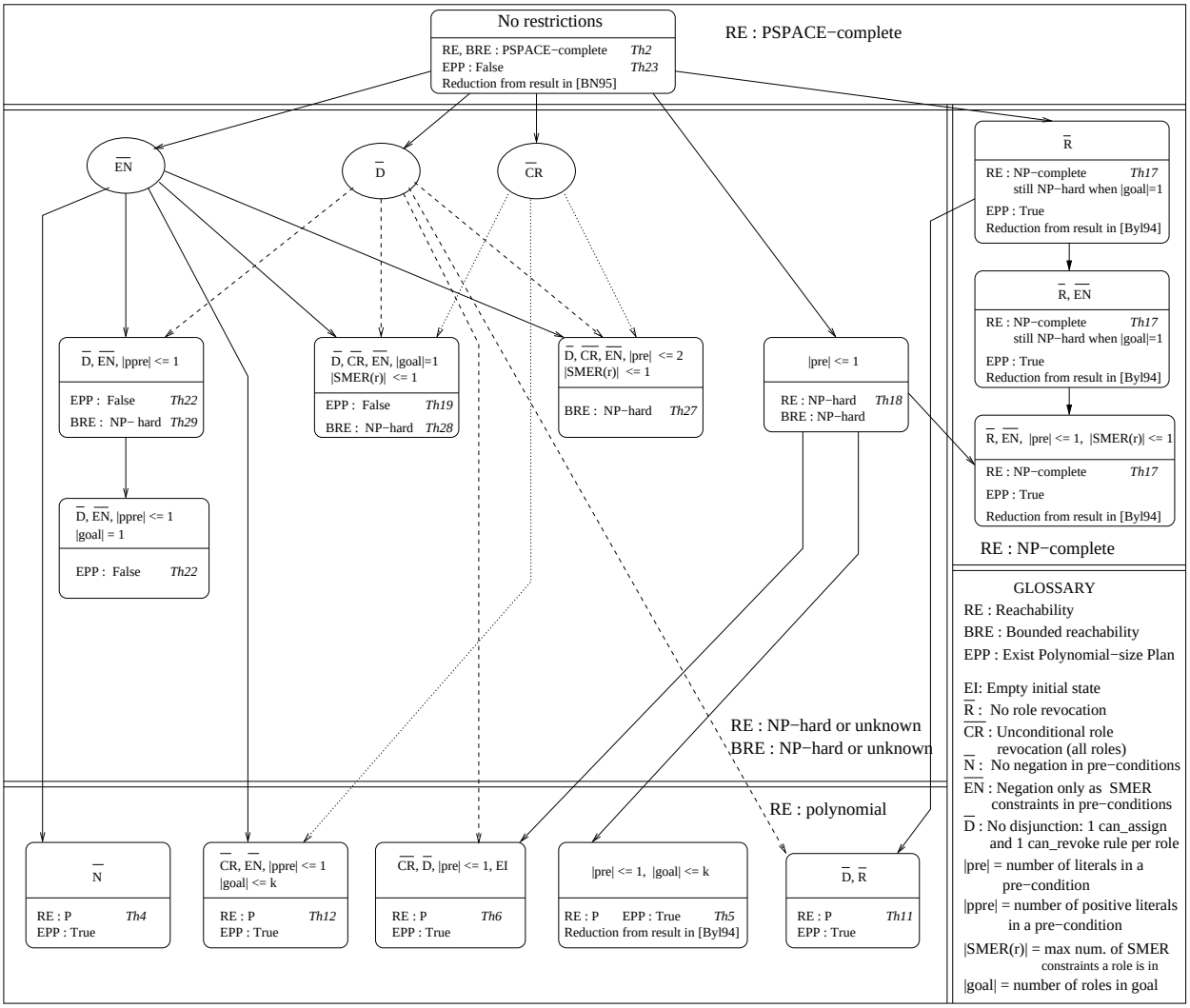

Figure 2: Complexity of Reachability Analysis

5.1 Proofs for PSPACE-Completeness of Reachability Analysis

Theorem 2 is based on complexity results for SAS$^+$ planning described in [BN95]. Below, we describe the SAS$^+$ planning model of [BN95].

Definition 10 An instance of the SAS$^+$ planning problem is given by the tuple $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_*)$ with components defined as follows.

- $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$ is a set of state variables. Each variable v has an associated domain D_v, which implicitly defines an extended domain $D_v^+ = D_v \cup \{u\}$ where u denotes the **undefined** value. Further, the total state space $S = D_{v_1} \times D_{v_2} \times \ldots \times D_{v_m}$ and the partial state space $S^+ = D_{v_1}^+ \times D_{v_2}^+ \times \ldots \times D_{v_m}^+$ are implicitly defined. We write $s[v]$ to denote the value of variable v in state s.

- $\mathcal{O}$ is a set of operators of the form $(pre, post, prv)$ where $pre, post, prv \in S^+$ denote the **pre**, **post**, and **prevail** conditions, respectively. Prevail conditions of an operator o are preconditions of o that remain unchanged by the execution of o. Operators are subject to the following restrictions: for every operator $(pre, post, prv)$ in $\mathcal{O}$,

  (R1) $\forall v \in \mathcal{V} : pre[v] \neq u \rightarrow pre[v] \neq post[v] \neq u$.

  (R2) $\forall v \in \mathcal{V} : post[v] = u$ or $prv[v] = u$.

- $s_0 \in S_{\mathcal{V}}^+$ and $s_* \in S_{\mathcal{V}}^+$ denote the initial and goal states respectively.

Restriction R1 says that a state variable can never become undefined once it has been defined by some operator. Restriction R2 says that the prevail condition of an operator must never define the variable being modified by the operator. If $o = (pre, post, prv)$ is an operator, we write $pre(o)$ to denote pre, *etc.*

A plan is a sequence of operators in $\mathcal{O}$. Let $\alpha.o$ denote the sequence α extended with element o. The state $result(s, \alpha)$ that results from executing plan α from state s is defined recursively by

$$\begin{aligned} result(s, \langle\rangle) &= s \\ result(s, \alpha.o) &= \begin{cases} result(s, \alpha) \oplus post(o) & \text{if } (pre(o) \sqcup prv(o)) \sqsubseteq result(s, \alpha) \\ s & \text{otherwise (this is an arbitrary choice)} \end{cases} \end{aligned}$$

where $s \oplus t$ (state s updated by state t) is defined by

$$(s \oplus t)[v] = \begin{cases} t[v] & \text{if } t[v] \neq u \\ s[v] & \text{otherwise} \end{cases}$$

and $s \sqcup t$ (join of s and t) is defined by

$$(s \sqcup t)[v] = \begin{cases} s[v] & \text{if } t[v] = u \\ t[v] & \text{if } s[v] = u \\ \text{undefined} & \text{otherwise} \end{cases}$$

and $s \sqsubseteq t$ (state s is subsumed by state t) is defined by $s \sqsubseteq t = (\forall v \in \mathcal{V}.\ s[v] = u \vee s[v] = t[v])$.

An operator o is admissible in a state s iff $(pre(o) \sqcup prv(o)) \sqsubseteq s$. A plan $\alpha = \langle o_1, \ldots, o_n \rangle$ is admissible in state s iff either α is empty or o_k is admissible in $result(s, \langle o_1, \ldots, o_{k-1} \rangle)$ for all $1 \leq k \leq n$.

Definition 11 The Plan Existence Problem for SAS$^+$ planning is to determine, given a problem instance $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_*)$, whether there exists a plan α such that α is admissible in s_0 and $s_* \sqsubseteq \mathit{result}(s_0, \alpha)$.

A SAS$^+$ planning instance $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_*)$ satisfies the

- **Binary** (B) restriction iff for all $v \in \mathcal{V}$, $|D_v| = 2$. Thus, every variable is boolean variable under the B restriction.
- **Unary** (U) restriction iff for all operators $o \in \mathcal{O}$, $\mathit{post}(o)[v] \neq u$ for exactly one $v \in \mathcal{V}$. Thus, an operator can change the state of a single variable under the U restriction.

Theorem 2 *Reachability* (RE) and *Bounded Reachability* (BRE) for the problem class without any restrictions are PSPACE-complete.

PROOF: [BN95] shows that Plan Existence for SAS$^+$ planning under the U and B restrictions is PSPACE-complete. Informally, the U restriction requires actions to have a single effect and the B restriction requires the effects of every action to be binary. The actions that we consider here—$\mathit{UserAssign}(r)$ and $\mathit{UserRevoke}(r)$—are binary actions that have a single effect, since they either add the user to r or revoke the user from r. We can encode Plan-Existence for a SAS$^+$ planning problem instance that satisfies the U and B restrictions as a miniARBAC*Reachability* problem instance. This establishes that solving *Reachability* for unrestricted miniARBAC policies is PSPACE-hard. By Lemma 1, BRE is at least as hard as RE, so BRE is also PSPACE-hard.

RE and BRE for unrestricted ARBAC policies are in NPSPACE, because a non-deterministic Turing Machine can guess a plan one step at a time, storing at each step only the current state (*i.e.*, current miniRBAC policy), whose size is polynomial in the size of the problem instance. These problems are therefore also in PSPACE, because Savitch's Theorem implies NPSPACE=PSPACE. Thus, RE and BRE for unrestricted ARBAC policies are PSPACE-complete.

□

5.2 Proofs for Polynomial-Time Reachability Analysis

Lemma 3 Let $I = (\gamma, \psi, \mathit{goal})$ be a reachability analysis problem instance where $\gamma = (R, UA)$ and $\psi = (\mathit{can\_assign}, \mathit{can\_revoke}, \mathit{SMER})$. If $\mathit{can\_revoke} = \emptyset$ then if I has a plan, I has a plan of length at most $|R|$.

PROOF: Since there is no revocation, a plan for I contains only role assignment actions. It is not useful for a plan to assign the user to the same role multiple times. More precisely, for each role to which the user is assigned by the plan, keep the first assignment of the user to that role and delete subsequent assignments (if any) of the user to that role. The length of the resulting plan is bounded by $|R|$. □

Theorem 4 RE for the problem class $[\overline{N}]$ can be solved in polynomial time. In addition, EPP for $[\overline{N}]$ is `true`.

PROOF: Let $I = (\gamma, \psi, \mathit{goal})$ be a reachability analysis problem instance in the problem class $[\overline{N}]$ where $\gamma = (R, UA)$ and $\psi = (\mathit{can\_assign}, \mathit{can\_revoke}, \mathit{SMER})$. Then, $\mathit{SMER} = \emptyset$. Since the preconditions in *can_assign* do not contain $\neg$, revoking a role does not help satisfy any precondition,

so we can assume *can_revoke* $= \emptyset$ for reachability analysis. From Lemma 3 it follows that if I has a plan, then I has a plan of length at most $|R|$. Thus, EPP for the problem class $[\overline{N}]$ is `true`.

RE for this problem class can be solved by a simple fixed-point algorithm. The algorithm computes the set S of all reachable roles by starting with S equal to the initial state UA and repeatedly adding roles to S using any tuple in *can_assign* whose precondition is satisfied by S. It is easy to see that this algorithm takes polynomial time.

Li and Tripunitara proved a very similar result, namely, that simple safety analysis for the Assignment and Trusted Users (AATU) problem class is solvable in polynomial time [LT06]. That result makes the additional assumption that all roles that can be assigned can be unconditionally revoked (similar to our $\overline{CR}$ restriction). □

We recall the definition of STRIPS planning and the PLANSAT decision problem [Byl94], and then prove the next theorem by reduction to PLANSAT.

Definition 12: Propositional STRIPS Planning [Byl94]. An instance of *propositional STRIPS planning* is specified by a tuple $(\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$ where:

- $\mathcal{P}$ is a finite set of ground atomic formulas, called *conditions*.
- $\mathcal{O}$ is a finite set of operators, where each operator o has the form $Pre \Rightarrow Post$
  - Pre is a satisfiable conjunction of positive and negative conditions, respectively called the positive preconditions o^+ and the negative preconditions o^- of the operator.
  - $Post$ is a satisfiable conjunction of positive and negative conditions, respectively called the positive postconditions o_+ and the negative postconditions o_- of the operator.
- $\mathcal{I} \subseteq \mathcal{P}$ is the initial state.
- $\mathcal{G}$ is a satisfiable conjunction of positive and negative goals, respectively called the positive goals $\mathcal{G}_+$ and the negative goals $\mathcal{G}_-$.

A state S is represented as a set containing the conditions that are true in the state. If the preconditions of an action $Pre \Rightarrow Post$ are satisfied in a state S, then executing this action in state S leads to the state $S \cup Post$.

Definition 13: PLANSAT [Byl94]. PLANSAT is the decision problem: given a propositional STRIPS planning problem $\phi = (\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$, determine whether there exists a sequence of operators that leads from $\mathcal{I}$ to a state that satisfies the goal $\mathcal{G}$. PLANSAT^1 is PLANSAT under the restriction that the precondition of every action contains at most one condition. PLANSAT$_+$ is PLANSAT under the restriction that all postconditions in ϕ are positive.

Theorem 5 For the problem class $[|pre| \leq 1, |goal| \leq k]$, RE can be solved in polynomial time and EPP is `true`.

Proof: The proof is based on a reduction from miniARBAC reachability analysis to STRIPS planning [Byl94]. The reduction allows us to use a planning algorithm to solve the reachability analysis problem.

Let $I = (\gamma, goal, \psi)$ be in the problem class $[|pre| \leq 1, |goal| \leq k]$ where $\gamma = (R, UA)$ and $\psi = (can\_assign,\ can\_revoke, SMER)$. Without loss of generality, we can assume $SMER = \emptyset$, because SMER constraints can be transformed into negative preconditions, as described in Section 3. Then, $|goal| \leq k$, and for each $(c, r) \in can\_assign \cup can\_revoke$, $|c| \leq 1$. Construct an instance $\phi = (\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$ of the PLANSAT^1 problem as follows.

- $\mathcal{P} = R$.
- $\mathcal{G} = goal$. Thus $|\mathcal{G}| \leq k$.
- $\mathcal{I} = UA$.
- $\mathcal{O} = \mathcal{O}^+ \cup \mathcal{O}^-$.
- $\mathcal{O}^+ = \{ UserAssign(r) : (c, r) \in can\_assign$.
  - $pre(UserAssign(r)) = c$.
  - $post(UserAssign(r)) = r$.
- $\mathcal{O}^- = \{ UserRevoke(r) : (c, r) \in can\_revoke$.
  - $pre(UserRevoke(r)) = c$.
  - $post(UserRevoke(r)) = \neg r$.

For every operator $o \in \mathcal{O}$, $pre(o) \leq 1$ since for each $(c, r) \in can\_assign \cup can\_revoke$, $|c| \leq 1$. Thus ϕ is in PLANSAT^1 and the number of goals in ϕ is limited to a constant k. In addition, $|\phi| = O(|I|)$. It is easy to see that a plan for I is also a plan for ϕ and vice versa.

From Theorem 3.8 in [Byl94] the PLANSAT^1 problem limited to constant number of goals is solvable in polynomial time. Thus, RE for the problem class $[|pre| \leq 1, |goal| \leq k]$ can be solved in polynomial time and EPP is `true`. □

Theorem 6 *Reachability* (RE) for the problem class $[\overline{CR}, \overline{D}, |pre| \leq 1, EI]$ (only unconditional role revocation, no disjunction, at most one precondition, empty initial state) is solvable in polynomial time.

PROOF: Let $I = (\gamma, goal, \psi)$ be a problem instance in the problem class $[\overline{CR}, \overline{D}, |pre| \leq 1, EI]$, where $\gamma = (R, UA)$, $\psi = (can\_assign, can\_revoke, SMER)$, and $goal$ is a set of roles. Then, for every role $r \in R$, (1) there is at most one state change rule $(c, r) \in can\_assign$, (2) $|c| \leq 1$, and (3) $(\texttt{true}, r) \in can\_revoke$. Also, since negative preconditions are allowed, we can assume without loss of generality that $SMER = \emptyset$. Note that the definition of the size restriction $|pre| \leq 1$ counts SMER constraints, so transforming SMER constraints into negative preconditions preserves that restriction.

Construct a graph $G_\psi = (V_\psi, E_\psi)$ as follows. The set of vertices V_ψ is the set of roles R. There are two kinds of edges in E_ψ, positive and negative. For each $(r', r) \in can\_assign$, $e = (r', r) \in E_\psi$, and $label(e) =$ `pos`. For each $(\neg r', r) \in can\_assign$, $e = (r, r') \in E_\psi$ and $label(e) =$ `neg`. Note that `neg` edges have reverse direction as the `pos` edges. Intuitively, edges in E_ψ indicate the order in which roles must be assigned and revoked; if $(r, r') \in E_\psi$, then $UserAssign(r)$ must occur before $UserAssign(r')$. Next, we prune the graph G_ψ by removing vertices for which there is no assignment rule and vertices that are reachable through a sequence of positive incoming edges from such vertices. These vertices are not reachable from the empty initial state because each vertex in G_ψ has at most one positive incoming edge (this follows from $|pre| \leq 1$ and $\overline{D}$). A cycle is called a `pos` cycle if it is composed of only `pos` edges; `neg` cycles are defined similarly.

Lemma 7 Let Y be a cycle in G_ψ. Then Y is either a `pos` cycle or a `neg` cycle.

PROOF: Suppose Y contains a `pos` edge and a `neg` edge. Then there exist vertices r, r_1 and r_2 in Y such that (r, r_1) is a `pos` edge and (r_1, r_2) is a `neg` edge. This means that there are two assignment rules for r_1, $(r, r_1) \in$ *can_assign* and $(\neg r_2, r_1) \in$ *can_assign*, which violates the $\overline{D}$ restriction. □ Lemma 7

Lemma 8 RE for $I = (\gamma, goal, \psi)$ where $\gamma = (R, \emptyset)$ is `false` if and only if either (`C1`) G_ψ contains a `pos` cycle Y such that $Y \cap goal \neq \emptyset$, (`C2`) G_ψ contains a `neg` cycle Y such that $Y \subseteq goal$, or (`C3`) $goal \not\subseteq G_\psi$.

PROOF: First, we show that if one of the conditions `C1`, `C2`, and `C3` holds, then RE for I is `false`.

Case 1: Suppose `C1` is true, *i.e.*, G_ψ contains a `pos` cycle Y and $r \in Y \cap goal$. Then, since we start in the empty state γ and since ψ satisfies the $\overline{D}$ restriction, the only way to derive the goal r is to first derive all roles in the cycle Y, including r itself. Thus, deriving role r entails deriving itself first, and hence no plan for deriving r exists. Therefore RE for I is `false`.

Case 2: Suppose `C2` is true, *i.e.*, G_ψ contains a `neg` cycle Y and $Y \subseteq goal$. Let $Y = (r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_k \rightarrow r_1)$. Then, $(\neg r_1, r_k) \in$ *can_assign*, and for $1 \leq i \leq k-1$, $(\neg r_{i+1}, r_i) \in$ *can_assign*. Let P be a plan for I; *i.e.*, $\gamma \xrightarrow{P} \gamma'$ and $\forall 1 \leq i \leq k : r_i \in \gamma'$. Since we start in the empty state γ, $\forall 1 \leq i \leq k : UserAssign(r_i) \in P$. Without loss of generality assume that $UserAssign(r_k)$ is the last action in P. Since the precondition for $UserAssign(r_k)$ is $\neg r_1$, it follows that $r_1 \notin \gamma'$ which contradicts the assumption that P is a plan for I. Thus, RE for I is `false`.

Case 3: If C3 is true, then there is no assignment rule for at least one of the goals and hence *goal* is not reachable.

Next, we show that if RE for I is `false`, then `C1` or `C2` or `C3` is `true`. We prove the contrapositive, *i.e.*, we assume conditions `C1`, `C2` and `C3` are false, and we show that RE for I is `true`, by giving an algorithm that constructs a plan for I. If all of `C1`, `C2` and `C3` are `false`, then G_ψ contains all goals and one of the following cases holds: (1) G_ψ is acyclic, in which case the topological-sort ordering of G_ψ gives the order in which roles must be assigned to reach *goal*; (2) G_ψ contains a `neg` cycle Y such that there exists $s \in Y$ and $s \notin goal$; or (3) G_ψ contains a `pos` cycle Y such that $Y \cap goal = \emptyset$. We break cycles in (2) by deleting each `neg` edge $e = (r, s)$ such that $r \in goal$ and $s \notin goal$. Since e is a `neg` edge, we know that $(\neg s, r) \in$ *can_assign*. Thus, in a plan for I, $UserRevoke(s)$ occurs between $UserAssign(s)$ and $UserAssign(r)$. We need to ensure that every $UserAssign(s')$ that has a precondition s occurs before $UserRevoke(s)$ and hence before $UserAssign(r)$ in the plan. We add edge (s', r) to G_ψ to ensure this. Regarding case (3), we can simply delete all cycles that do not contain any goal. With the above transformations, the resulting graph G'_ψ is acyclic, and we can generate a plan for I by assigning roles (to the user) in the topological-sort order of G'_ψ. The algorithm is given in Algorithm 1. □ Lemma 8

Constructing graph G_ψ takes polynomial time, and $|G_\psi| = |I|$. Validity of `C1` can be checked by restricting G_ψ to only `pos` edges. Since ψ satisfies the $\overline{D}$ (no disjunction) restriction, in this restricted graph each vertex has at most one incoming edge. This implies that all cycles in the graph are disjoint and we can use a simple Depth-First Search to find all cycles and check whether any cycle contains a role not in *goal*. Validity of condition `C2` can be checked by restricting G_ψ to vertices in *goal* and to `neg` edges, and checking whether the restricted graph contains a cycle; a simple Depth-First Search can accomplish this. Validity of condition `C3` can be checked by traversing G_ψ at most once. Hence `C1`, `C2` and `C3` can be checked in polynomial time. Transforming G_ψ to an acyclic graph G'_ψ takes polynomial time, and $|G'_\psi|$ is $O(|G_\psi|^2)$, since for each `neg` edge in G_ψ, at most $|G_\psi|$ new edges may be added. Topologically sorting G'_ψ takes polynomial time. Thus, *Reachability* for this problem class can be solved in polynomial time. □ Theorem 6

Algorithm 1 Plan generation for problem class $[\overline{CR}, \overline{D}, |pre| \leq 1, EI]$

Input: Problem instance $I = ((R, \emptyset), goal, \psi)$
Output: Returns the plan if a plan exists for I, else returns `false`

1: **if** `C1`, `C2` or `C3` is `true` for G_ψ **then**
2: return `false`
3: **end if**
4: Construct graph $G'_\psi = (V'_\psi, E'_\psi)$
5: Construct graph $G''_\psi = comp(G'_\psi, goal)$
6: Topologically sort G''_ψ.
7: Plan $P = \langle\rangle$
8: $UA_{new} = \emptyset$
9: **for all** $r \in V''_\psi$ in topologically sorted order **do**
10: $(c, r) \in can\_assign$
11: **if** $c = \neg s \wedge s \in UA_{new}$ **then**
12: $P = P.\langle UserRevoke(s)\rangle$
13: $UA_{new} = UA_{new} \setminus \{s\}$
14: **end if**
15: $P = P.\langle UserAssign(r)\rangle$
16: $UA_{new} = UA_{new} \cup \{r\}$
17: return P
18: **end for**

The problem class to which Theorem 6 applies can be expanded by reducing problem instances that do not satisfy the *EI* (empty initial state) restriction to problem instances that satisfy *EI*. The next two lemmas express such reductions.

Lemma 9 Suppose ψ satisfies $\overline{CR}$. RE for $((R, UA), goal, \psi)$ is `true` if RE for $((R, \emptyset), goal, \psi)$ is `true`.

PROOF: Since ψ allows unconditional revocation of all roles, we can revoke all roles $r \in UA$ to transform the initial RBAC state to the empty state $(R, \emptyset)$. We then check if $(R, goal)$ is reachable from $(R, \emptyset)$. Thus, if a state (R, UA_1) is reachable from the empty state, then it is reachable from any state. □

Lemma 10 RE for $((R, UA), goal, \psi)$ is `false` if RE for $((R, \emptyset), UA, \psi)$ is `true` and RE for $((R, \emptyset), goal, \psi)$ is `false`.

PROOF: Let I, I_1, and I_2 denote the three problem instances in the statement of the lemma, in the order they appear. The proof is by contradiction. Suppose for contradiction that RE for I is `true`, i.e., there is a path P from (R, UA) to $(R, goal)$. Since I_1 is `true`, there is a path P_1 from $(R, \emptyset)$ to (R, UA). The concatenation of P_1 and P is a plan for I_2, contradicting the hypothesis that RE for I_2 is `false`. □

Theorem 11 For the problem class $[\overline{D}, \overline{R}]$, RE is solvable in polynomial time and EPP is `true`.

PROOF: Let $I = (\gamma, goal, \psi)$ be a problem instance that satisfies the $[\overline{D}, \overline{R}]$ restriction where $\gamma = (R, UA)$, $\psi = (can\_assign, \emptyset, SMER)$. Without loss of generality, we assume $SMER = \emptyset$.

For $(c, r) \in$ *can_assign*, let $post((c, r)) = r$, $ppre((c, r)) =$ positive literals in c, and $npre((c, r)) =$ negative literals in c. We will sometimes refer to elements of *can_assign* as "rules".

For a role r, let $rule(r)$ be the unique element of *can_assign* with postcondition r, if any, otherwise $\perp$. For convenience, define $ppre(\perp) = \perp$.

Let *goals* be the limit of the following monotonically increasing sequence of subsets of R: $goals_0 = goal$ and $goals_{i+1} = (goals_i \cup (\cup_{r \in goals_i} ppre(rule(r)))) \setminus UA$. The $\overline{D}$ assumption implies that in every plan for I, all of the roles in *goals* must be added to the state. If $\perp \in goals$, then return false (i.e., there is no plan for I), because there is no rule to add some role in *goals*.

Let $S = \cup_{r \in goals} rule(r)$. The $\overline{D}$ assumption implies that in every plan for I, every rule in S must be fired at least once. The $\overline{R}$ assumption implies that it suffices to fire each rule at most once. Thus, there exists a plan for I if and only if some linearization of S is a plan for I.

If $UA \cap (\cup_{r \in goals} npre(r)) \neq \emptyset$, then return false. This is because some rule in S has a negative precondition that cannot be satisfied, regardless of the order in which rules are fired.

Now consider constraints on the order of firing the rules in S. A rule ρ in S must fire before another rule ρ' in S, denoted $\rho < \rho'$, if $post(\rho) \in ppre(\rho')$ or $post(\rho') \in npre(\rho)$. At this point in the algorithm (i.e., if the algorithm has not already returned false), every plan satisfying these ordering constraints is feasible. This means that there exists a plan for I iff the graph $G = (S, <)$ is acyclic.

Constructing G and testing acyclicity takes polynomial time in $|I|$, so reachability for this problem class is decidable in polynomial time. If G is acyclic, then any topological sort of G is a plan for I, and its length is polynomial in $|I|$. Thus EPP for this problem class is true. □

Theorem 12 RE for the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$ (where k is a constant) is solvable in polynomial time. In addition, EPP for the class is `true`.

PROOF: Let $I = (\gamma, goal, \psi)$ be a problem instance in the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$, where $\gamma = (R, UA)$, $\psi =$ (*can_assign*, *can_revoke*, *SMER*). Then, for each $r \in R$, $(\texttt{true}, r) \in$ *can_revoke* (*i.e.*, every role can be unconditionally revoked), and if $(c, r) \in$ *can_assign*, then $|c| \leq 1$ and c is `true` or a positive literal.

Algorithm 2 checks plan existence for this problem class. The algorithm starts with the state *goal*. At each step it picks an unexplored node and explores it. During the process of exploration, new states are generated that are then recursively explored. Furthermore, a directed graph $G_\psi = (V_\psi, E_\psi)$ is constructed during the process of exploration such that V_ψ contains all generated states and E_ψ contains explored transitions between those states. Let UA_0 be the initial state. When exploring a state UA, if there exists some role $r \notin UA_0$ in the state such that the precondition of every *can_assign* rule for r contains a role that is mutually exclusive with r, then r is not reachable from UA_0. If a state $UA \subseteq UA_0$ is reached, the algorithm returns the graph G_ψ and UA. The $\overline{CR}$ restriction implies that the state UA is reachable from UA_0 simply by revoking all roles in $UA_0 \setminus UA$.

If plan existence for I is `true`, then a plan can be generated from G_ψ and UA as follows. Let $P = \langle e_1, e_2, \ldots, e_n \rangle$ be a path in G_ψ from UA to the final state $UA_{final} = goal$. Let $\pi = A_1.A_2.\ldots.A_n$ where $label(e_i) = UserAssign(r_i)$, $S_i = \{r' \in R : (r_i, r') \in SMER\}$, and $A_i = \{UserRevoke(s) : s \in S_i\}.UserAssign(r_i)$. Note that A_i consists of the indicated *UserRevoke* actions in arbitrary order, followed by the indicated *UserAssign* action. It is easy to see that π is a plan for I.

Each state in G_ψ contains at most k roles, because the search starts with the state *goal*, and the assignments on lines 15 and 17 ensure that $|UA_{new}| \leq |UA|$. To bound the number of states, note

Algorithm 2 *Reachability* for the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$

Input: Problem instance $I = ((R, UA_0), goal, \psi)$

Output: Returns a directed graph and an initial state if a plan exists for I, else returns `false`.

if $\exists\ r_1, r_2 \in goal : (r_1, r_2) \in SMER$ **then**
  return `false`
end if
Vertex set $V_\psi = \{goal\}$
Edge set $E_\psi = \emptyset$
while There exists an unexplored node in V_ψ **do**
  Pick an unexplored node UA from V_ψ.
  Mark UA as explored
  if $UA \subseteq UA_0$ **then**
    return (V_ψ, E_ψ) and UA
  else
    for all $r \in UA$ **do**
      for all $(s, r) \in can\_assign$ **do**
        if $s ==$ `true` **then**
          $UA_{new} = UA \setminus \{r\}$
        else
          $UA_{new} = (UA \cup \{s\}) \setminus \{r\}$
        end if
        $S = \{r' \in R : (r, r') \in SMER\}$
        if $UA_{new} \cap S \neq \emptyset$ **then**
          continue
        end if
        if $UA_{new} \notin V_\psi$ **then**
          Add UA_{new} to V_ψ and mark it unexplored
        end if
        Add edge $e = (UA_{new}, UA)$ to E_ψ and set $label(e) = UserAssign(r)$
      end for
    end for
  end if
end while
return `false`

that there are $\binom{|R|}{k}$ different sets containing exactly k roles, and there are 2^k subsets of each of them, so the total number of states in G_ψ is $O(2^k \times \binom{|R|}{k})$, which is $O(|I|^k)$ (recall that k is considered constant). Since the number of explored states is polynomial in $|I|$, it is easy to see that Algorithm 2 runs in time polynomial in $|I|$. Thus, RE for the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$ can be solved in polynomial time.

A plan for I is at most the length of the longest path in G_ψ times the maximum length of an A_i. The former length is $O(|I|^k)$, and the latter length is $O(k)$, so EPP for the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$ is `true`. □

5.3 Proofs for NP-Completeness of Reachability Analysis

Theorem 13 For the problem class $[\overline{R}]$, RE is in NP and EPP is `true`.

PROOF: Let $I = (\gamma, goal, \psi)$ be a reachability analysis problem in the problem class $[\overline{R}]$ where $\gamma = (R, UA)$ and $\psi = (can\_assign,\ can\_revoke, SMER)$. Since ψ satisfies the $\overline{R}$ restriction, *can_revoke* $= \emptyset$. From Lemma 3, it follows that if I has a plan, then I has a plan of length at most $|R|$, and the plan consists entirely of *UserAssign* actions. Thus EPP for the problem class $[\overline{R}]$ is `true`.

Given a sequence P of $UserAssign(r_i)$ actions ($r_i \in R$) with $|P| \leq |R|$, a Turing Machine can verify whether P is a plan for I by (1) executing P on γ and transforming it to a state $\gamma' = (R, UA')$, and (2) checking that $goal \subseteq UA'$. Both these operations take time polynomial in $|I|$. Thus, RE for the problem class $\overline{R}$ is in NP. □

Theorem 14 RE for the problem classes $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER(r)| \leq 1]$ and $[\overline{R}, \overline{EN}, |goal| = 1, |SMER(r)| \leq 1]$ are NP-hard.

PROOF: First, we show that RE for the problem class $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER(r)| \leq 1]$ is NP-hard. The proof follows the proof of NP-hardness of the PLANSAT$_+$ problem [Byl94, (Theorem 3.5)]. We reduce 3-SAT to RE for the above problem class. Consider a 3-SAT formula $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ containing propositional variables in $V = \{x_1, \ldots, x_m\}$, where $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, and each literal l_{ij} is a variable in V or the negation of one. Construct a problem instance $I = (\gamma, goal, \psi)$ as follows.

1. The set of roles R is defined as follows. For each literal $x_i \in V$ there are two roles t_i and f_i in R. For each clause C_i, there is a role c_i in R.
2. $\gamma = (R, \emptyset)$. $goal = \{c_1, c_2, \ldots, c_n\}$.
3. $\psi = (can\_assign,\ can\_revoke, SMER)$ is defined as follows.
   (a) For $1 \leq i \leq m$: $(\texttt{true}, t_i) \in can\_assign$.
   (b) For $1 \leq i \leq m$: $(\texttt{true}, f_i) \in can\_assign$.
   (c) For each clause $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, for $1 \leq j \leq 3$, if l_{ij} is the literal x_k then $(t_k, c_i) \in can\_assign$, else if l_{ij} is the literal $\neg x_k$ then $(f_k, c_i) \in can\_assign$.
   (d) $can\_revoke = \emptyset$.
   (e) $SMER = \{(t_i, f_i) : x_i \in V\}$.

Because $can\_revoke = \emptyset$, I satisfies the $\overline{R}$ restriction. Also, the *can_assign* rules do not contain $\neg$, and t_i appears only with f_i in *SMER* (and vice versa). Thus the policy satisfies the $\overline{EN}$ and $|SMER(r)| \leq 1$ restrictions. Furthermore, *can_assign* also satisfies the $|pre| \leq 1$ restriction.

Lemma 15 If RE for I is `true` then $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ is satisfiable.

PROOF: From Theorem 13 we know that I has a plan $P = (a_1, a_2, \ldots, a_m)$ where each a_i is a *UserAssign* action and $m \leq |R|$. Construct an assignment π to literals in V as follows.

1. If $UserAssign(t_i) \in P$ then $\pi(x_i) =$ `true`.
2. If $UserAssign(f_i) \in P$ then $\pi(x_i) =$ `false`.

Note that only one of $UserAssign(t_i)$ or $UserAssign(f_i)$ can occur in P; $(t_i, f_i) \in SMER$ and since $can\_revoke = \emptyset$, once either of $UserAssign(t_i)$ or $UserAssign(f_i)$ occurs in P, the other action cannot occur in P. Thus π is a well-formed assignment.

Let $\gamma \xrightarrow{a_1} \gamma_1 \xrightarrow{a_2} \gamma_2 \ldots \xrightarrow{a_u} \gamma_u$, where $\forall 1 \leq i \leq u : \gamma_i = (R, UA_i)$. Since P is a plan, $\forall\ 1 \leq i \leq n : c_i \in UA_u$. Thus, for each c_i, $\exists\ j \leq u : a_j = UserAssign(c_i)$. Since $UserAssign(c_i)$ is enabled in γ_{j-1}, there exists a state change rule $(X, c_i) \in can\_assign$ such that $X \in UA_{j-1}$. Thus, $\exists\ v < j-1 : a_v = UserAssign(X)$. From the construction of I it follows that if $X = t_k$ then $x_k \in C_i$ and $\pi(x_k) =$ `true`, and if $X = f_k$ then $\neg x_k \in C_i$ and $\pi(x_k) =$ `false`. Thus, $\pi(C_i) =$ `true`. Since this is true for every $1 \leq i \leq n$, $\pi(\phi) =$ `true`. Thus, ϕ is satisfiable. □ Lemma 15

Lemma 16 If ϕ is satisfiable then RE for I is `true`.

PROOF: Consider a satisfiable assignment π of ϕ. Construct a sequence of actions $P = P'.P''$ as follows. For all literals $x_i \in V$, if $\pi(x_i) =$ `true` then $UserAssign(t_i) \in P'$, and if $\pi(x_i) =$ `false` then $UserAssign(f_i) \in P'$. $P'' = \{UserAssign(c_i) : 1 \leq i \leq n\}$. We show that P is a plan for I.

Note that every action $a \in P'$ is enabled in γ. Let $\gamma \xrightarrow{P'} \gamma'$. For every clause $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, since $\pi(C_i) =$ `true`, $\exists\ 1 \leq j \leq 3 : \pi(l_{ij}) =$ `true`. If $l_{ij} = x_k$ then $\pi(x_k) =$ `true` and hence $UserAssign(t_k) \in P'$. Also, $(t_k, c_i) \in can\_assign$. Similarly, if $l_{ij} = \neg x_k$ then $\pi(x_k) =$ `false`, $UserAssign(f_k) \in P'$, and $(f_k, c_i) \in can\_assign$. Thus all actions $UserAssign(c_i) \in P''$ are enabled in γ'. Therefore $\gamma' \xrightarrow{P''} \gamma''$ where $\gamma'' = (R, UA'')$ and $\forall 1 \leq i \leq n : c_i \in UA''$. Thus $P = P'.P''$ is a plan for I. □ Lemma 16

From Lemmas 15 and 16 it follows that RE for the problem class $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER(r)| \leq 1]$ is NP-hard.

As discussed in Section 4, reachability of a goal containing multiple roles can be transformed to reachability of a singleton goal, and the transformation preserves all restrictions except $|pre| \leq k$ and $|ppre| \leq k$. Therefore, the above proof can directly be applied to show that reachability for the problem class $[\overline{R}, \overline{EN}, |goal| = 1, |SMER(r)| \leq 1]$ is NP-hard.

□ Theorem 14

Theorem 17 For the problem classes $[\overline{R}]$, $[\overline{R}, \overline{EN}]$, $[\overline{R}, \overline{EN}, |goal| = 1, |SMER(r)| \leq 1]$, and $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER(r)| \leq 1]$, RE is NP-complete and EPP is `true`.

PROOF: Follows immediately from Theorems 13 and 14. □

5.4 Proofs for NP-Hardness of Reachability Analysis

Theorem 18 *Reachability* (RE) and *Bounded Reachability* (BRE) for the problem class $[|pre| \leq 1]$ are NP-hard.

PROOF: NP-hardness of RE for this problem class is a corollary of Theorem 17, which shows that RE is NP-complete for a more restricted problem class. NP-hardness of BRE for this problem class follows from NP-hardness of RE and Lemma 1. □

Theorem 19 EPP for the complexity class $[\overline{D}, \overline{CR}, \overline{EN}, |goal| = 1, |SMER(r)| \leq 1]$ is `false`.

PROOF: Consider the problem instance $I_n = (\gamma, goal, \psi)$ where:

- the set of roles $R = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n\}$,
- $\gamma = (R, \emptyset)$
- $goal = \{u_n\}$
- $\psi = (can\_assign,\ can\_revoke, SMER)$ where
  - $SMER = \{(u_i, v_i) : 1 \leq i \leq n\}$
  - $\forall\ 1 \leq i \leq n : (\texttt{true}, v_i) \in can\_revoke$
  - $\forall\ 1 \leq i \leq n : (\texttt{true}, v_i) \in can\_assign$
  - $\forall\ 1 \leq i \leq n : (\texttt{true}, u_i) \in can\_revoke$
  - $(\texttt{true}, u_1) \in can\_assign$, $(u_1, u_2) \in can\_assign$ and $\forall\ 3 \leq i \leq n$ if $i = 2k + 1$ then $(v_1 \wedge v_2 \ldots \wedge v_{i-2} \wedge u_{i-1}, u_i) \in can\_assign$, else if $i = 2k$ then $(u_1 \wedge u_2 \wedge \ldots \wedge u_{i-1}, u_i) \in can\_assign$.

The *can_revoke* relation specifies that for every role $r \in R$, $UserRevoke(r)$ has a `true` precondition. Thus, I_n satisfies the $\overline{CR}$ restriction (every role can be unconditionally revoked). For every role $r \in R$, there is a unique state-change rule (c, r) in both *can_assign* and *can_revoke*. Thus I_n satisfies the $\overline{D}$ restriction. The preconditions in *can_assign* and *can_revoke* do not contain negation, and each role u_i appears only with v_i in *SMER*. Thus I_n satisfies the $\overline{EN}$ and $|SMER(r)| \leq 1$ restrictions.

Define the following sequences.[1]

- $\forall\ 1 \leq i \leq n : V_i = \langle UserAssign(v_1),\ UserAssign(v_2),\ \ldots,\ UserAssign(v_i)\rangle$. Given any state γ, $\gamma \xrightarrow{V_i} \gamma \cup \{v_{1\ldots i}\}$
- $V'_i = \langle UserRevoke(v_1), UserRevoke(v_2),\ \ldots,\ UserRevoke(v_i)\rangle$. Given any state γ, $\gamma \xrightarrow{V'_i} \gamma \setminus \{v_{1\ldots i}\}$
- $U'_i = \langle UserRevoke(u_1), UserRevoke(u_2),\ \ldots,\ UserRevoke(u_i)\rangle$. Given any state γ, $\gamma \xrightarrow{U'_i} \gamma \setminus \{u_{1\ldots i}\}$
- $U_{2i} = U_{2i-1}.\langle UserAssign(u_{2i})\rangle$, and $U_{2i+1} = U_{2i}.U'_{2i-1}.V_{2i-1}.\langle UserAssign(u_{2i+1})\rangle.V'_{2i-1}.U_{2i-1}$
- $\alpha_{2i} = U_{2i}$ and $\alpha_{2i+1} = U_{2i}.U'_{2i-1}.V_{2i-1}.\langle UserAssign(u_{2i+1})\rangle$

Lemma 20 α_n is the minimum size plan for I_n

[1] We use angle brackets to denote sequences. $x.y$ denotes the concatenation of sequences x and y.

PROOF: First we show that α_n is a plan for I_n. Define $\gamma_i = (R, \{u_1, u_2, \ldots, u_i\})$. Then $\gamma = \gamma_0$. It is easy to see that $\gamma_0 \stackrel{U_i}{\rightarrow} \gamma_i$: $\gamma_0 \stackrel{U_1}{\rightarrow} \gamma_1$ is `true`, and applying induction, if $\gamma_0 \stackrel{U_{i-1}}{\rightarrow} \gamma_{i-1}$, then

1. if $i = 2k$ then $\gamma_{2k-1} \stackrel{UserAssign(u_{2k})}{\rightarrow} \gamma_{2k}$. Thus $\gamma_0 \stackrel{U_{2k}}{\rightarrow} \gamma_{2k}$
2. if $i = 2k+1$ then $\gamma_{2k} \stackrel{U'_{2k-1}.V_{2k-1}.\langle UserAssign(u_{2k+1})\rangle.V'_{2k-1}}{\rightarrow} \gamma'$ where $\gamma' = (R, \{u_{2k}, u_{2k+1}\})$. Because $\gamma_0 \stackrel{U_{2k-1}}{\rightarrow} \gamma_{2k-1}$, and u_{2k} and u_{2k+1} does not affect this derivation, $\gamma' \stackrel{U_{2k-1}}{\rightarrow} \gamma_{2k+1}$. Thus $\gamma_0 \stackrel{U_{2k+1}}{\rightarrow} \gamma_{2k+1}$.

Also note that $u_n \in \gamma_n$. Therefore, U_n is a plan for I_n. Since α_n is a prefix of U_n and the last action in α_n is $UserAssign(u_n)$, α_n is a plan for I_n.

Next we argue that α_n is the minimum plan for reaching the goal u_n. The proof is by induction. $\alpha_1 = \langle UserAssign(u_1)\rangle$ and $\alpha_2 = \langle UserAssign(u_1), UserAssign(u_2)\rangle$ are trivially the minimal plans for reaching the goals u_1 and u_2 respectively. Suppose for $k < n$, $1 \leq i \leq k$, α_i is the minimum plan for reaching goal u_i. We show that α_{k+1} is the minimum plan for achieving the goal u_{k+1}.

Observation: If $i = 2j$ (*i.e.*, i is even), then since the precondition of $UserAssign(u_{2j})$ is $u_1, u_2, \ldots, u_{2j-1}$, it follows that any path to goal u_{2j} must go through the state $\nu_{2j} = (R, \{u_1, u_2, \ldots, u_{2j-1}, u_{2j}\})$. Similarly, if $i = 2j+1$ (*i.e.*, i is odd), then since the precondition of $UserAssign(u_{2j+1})$ is $v_1, v_2, \ldots, v_{2j-1}, u_{2j}$, it follows that any path to goal u_{2j+1} must go through the state $\nu_{2j+1} = (R, \{v_1, v_2, \ldots, v_{2j-1}, u_{2j}, u_{2j+1}\})$. Thus, if $\nu_0 = (R, \emptyset)$, then since α_i is the shortest path to u_i, it follows that $\nu_0 \stackrel{\alpha_i}{\rightarrow} \nu_i$, *i.e.*, α_i transforms the initial state ν_0 to ν_i.

Case 1 : $k = 2j$ for some j, *i.e.*, k is even. Any path to the goal u_{2j+1} must go through the state ν_{2j+1}. Thus, the shortest path to u_{2j+1} ends in state ν_{2j+1}. Let P be the shortest path to u_{2j+1}, and P ends in ν_{2j+1}. Since $u_{2j} \in \nu_{2j+1}$, it follows from the above observation that P goes through the state ν_{2j}. Thus, $P = \langle P_1, P_2\rangle$ where P_1 is the shortest path from ν_0 to ν_{2j}, and P_2 is the shortest path from ν_{2j} to ν_{2j+1}. By the induction hypothesis, P_1 is α_{2j}. It is clear that $P_2 = U'_{2j-1}.V_{2j-1}.\langle UserAssign(u_{2j+1})\rangle$. Thus, $P = \alpha_{2j}.U'_{2j-1}.V_{2j-1}.\langle UserAssign(u_{2j+1})\rangle = \alpha_{2j+1} = \alpha_{k+1}$. Therefore, α_{k+1} is the minimum plan for achieving the goal u_{k+1}.

Case 2 : $k = 2j-1$ for some j, *i.e.*, k is odd. Any path to goal u_{2j} must go through the state ν_{2j}. Thus, the shortest path to u_{2j} ends in state ν_{2j}. Let P be the shortest path to u_{2j}, and P ends in ν_{2j}. Since $u_{2j-1} \in \nu_{2j}$, it follows from the above observation that P goes through the state ν_{2j-1}. Thus, $P = P_1.P_2$ where P_1 is the shortest path from ν_0 to ν_{2j-1}, and P_2 is the shortest path from ν_{2j-1} to ν_{2j}. By the induction hypothesis, $P_1 = \alpha_{2j-1}$. It is clear that P_2 should first revoke all the $v_1, v_2, \ldots, v_{2j-3}$ roles before assigning the $u_1, u_2, \ldots, u_{2j-3}$ roles. After all the $v_1, v_2, \ldots, v_{2j-3}$ roles are revoked, P_2 must transform the state ν_0 to the state ν_{2j-2}. Since u_{2j-1} and u_{2j-2} are in ν_{2j-1}, and reachability of $u_1, \ldots u_{2j-3}$ using U_{2j-3} is not affected by carrying along u_{2j-1} and u_{2j-2} in the state, we conclude that $P = \alpha_{2j-1}.V'_{2j-3}.U_{2j-3}.\langle UserAssign(u_{2j})\rangle = U_{2j-1}.\langle UserAssign(u_{2j})\rangle = \alpha_{2j}$. Therefore α_{k+1} is the minimum plan for achieving the goal u_{k+1}.

From Cases 1 and 2 it follows that α_n is the minimum plan for achieving the goal u_n. □ Lemma 20

Lemma 21 $|\alpha_n| = \Omega(2^{p(n)})$ where $p(n)$ is a polynomial in n.

PROOF: We first show that $|U_n| = \Omega(2^n)$. Note that $|U_{2i+1}| = |U_{2i}| + |U_{2i-1}| + c_i$ where c_i is a constant. Also, $|U_{2i}| = |U_{2i-1}| + 1$. It follows that $|U_{2i+1}| = 2|U_{2i-1}| + c_i + 1$. This implies $|U_{2i+1}| \geq 2^i|U_1| = \Omega(2^i)$. Thus, $|U_n| = \Omega(2^{n/2})$. From the definition of α_n, we have $|\alpha_n| = \Omega(2^{n/2})$. □ Lemma 21

Note that $|I_n|$ is $O(n^2)$. Thus, from Lemmas 20 and 21, it follows that the minimum size plan for I_n is not polynomial in $|I_n|$. Therefore, EPP for the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |SMER(r)| \leq 1]$ is `false`. □ Theorem 19

Theorem 22 EPP for the complexity class $[\overline{D}, \overline{EN}, |ppre| \leq 1, |goal| = 1]$ is `false`.

PROOF: Consider the problem instance $I_n = (\gamma, goal, \psi)$ where:

- the set of roles $R = \{r_1, r_2, \ldots, r_n\}$.
- $\gamma = (R, \emptyset)$
- $goal = \{r_n\}$
- $\psi = (can\_assign,\ can\_revoke, SMER)$ where
  - $\forall 1 \leq i \leq n-2, i+2 \leq j \leq n : (r_i, r_j) \in SMER$
  - $(\texttt{true}, r_1) \in can\_assign$ and $\forall\ 2 \leq i \leq n : (r_{i-1}, r_i) \in can\_assign$
  - $(\texttt{true}, r_1) \in can\_revoke$ and $\forall\ 2 \leq i \leq n : (r_{i-1}, r_i) \in can\_revoke$

For each role r_i, the *can_assign* and *can_revoke* relations have precondition r_{i-1}. Define the following sequences.

- $\forall\ 1 \leq i \leq n :\ S_i = R'_i.R'_{i-1}...R'_1$
- $R'_1 = \langle UserRevoke(r_1) \rangle$ and $\forall\ 2 \leq i \leq n : R'_i = R_{i-1}.\langle UserRevoke(r_i) \rangle$
- $R_1 = \langle UserAssign(r_1) \rangle$, $R_2 = \langle UserAssign(r_1),\ UserAssign(r_2) \rangle$, and $\forall\ 3 \leq i \leq n : R_i = R_{i-1}.S_{i-2}.\langle UserAssign(r_i) \rangle$

It is easy to see that R_n is a plan for I_n. We can show that R_n is the minimum plan for I_n similar to the proof of Theorem 19. In addition, $|R_k| = |R_{k-1}| + |S_{k-2}| + 1 = |R_{k-1}| + 1 + \Sigma_{i=1}^{k-2}|R'_i| = k - 1 + |R_{k-1}| + \Sigma_{i=1}^{k-3}|R_i|$. Since $|R_k|$ is monotonic in k, it follows that $|R_k| > 2|R_{k-3}|$, and hence $|R_n| = \Omega(2^{n/3})$. Therefore, the minimum plan for I_n has size exponential in $|I_n|$. Thus, EPP for the problem class $[\overline{D}, \overline{EN}, |ppre| \leq 1, |goal| \leq 1]$ is `false` □

Theorem 23 EPP for the problem class with no restrictions is `false`.

PROOF: Follows immediately from Theorem 19 or Theorem 22. □

Lemma 24 *Bounded Reachability* (BRE) for the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |ppre| \leq 2, |SMER(r)| \leq 1]$ is NP-hard.

PROOF: The proof is by reduction from the `CLIQUE` problem, which is known to be NP-complete [Kar72]. The proof is based on the proof of Theorem 4.15 in [BN95], but our construction and proof are more complicated, because we are dealing with a more restricted problem class. Given a graph $G = (V, E)$ where $V = \{v_1, \ldots v_n\}$ and an integer k, the `CLIQUE` problem asks whether G has a clique of size k, i.e., whether there is a set $C \subseteq V$ of size k such that there is an edge between every pair of vertices in C. We construct a problem instance $I = (\gamma, goal, \psi)$ in the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |pre| \leq 2, |SMER(r)| \leq 1]$ such that G has a clique of size k if and only if I has a plan of size at most $n^2 + 13n - k$. The basic idea behind the reduction is to construct an ARBAC policy

such that the following hold. (1) If the policy has a plan, then the plan can be regarded as an interleaving of several sequences of actions, with one such sequence of actions corresponding to each node of G. If a node v_i is in a clique of size k or larger, then there are two feasible options—with different sizes—for the subsequence corresponding to v_i. If v_i is not in a clique of size k (or larger), then only one of the options—the longer one—is possible. (2) If the policy has a plan, the size of the shortest plan is $p(n) - mk$, where $p(n)$ is some polynomial in n, and m is an integer constant. The construction is designed to make the plan length anti-monotonic in (i.e., a non-increasing function of) k, because, if a graph contains a clique of size greater than k, then it also contains a clique of size k, so considering a larger clique must not make the length of the plan exceed the bound in the generated BRE problem instance.

Let $N_i = \{v \in V : (v, v_i) \notin E\}$. Note that $|N_i| = 0$ if there are edges between v_i and all other vertices in V. Define $I = (\gamma, goal, \psi)$ where

1. $\gamma = (R, UA)$, where $R = \bigcup_{1 \le i \le n} \{v_i, a_i, b_i, c_i, \bar{c}_i, d_i, \bar{d}_i, e_i\} \cup \bigcup_{1 \le i,j \le n} \{g_{ij}, h_{ij}\}$ and $UA = \{c_i, d_i : 1 \le i \le n\}$.

2. $\psi = (can\_assign,\ can\_revoke, SMER)$ where

   - *can_assign* is defined as
     (a) $\forall 1 \le i \le n : (\texttt{true}, v_i) \in can\_assign$
     (b) $\forall 1 \le i \le n : (\bar{c}_i \wedge d_i, a_i) \in can\_assign$
     (c) $\forall 1 \le i \le n : (c_i \wedge \bar{d}_i, b_i) \in can\_assign$
     (d) $\forall 1 \le i \le n : (\texttt{true}, \bar{c}_i) \in can\_assign$
     (e) $\forall 1 \le i \le n : (\texttt{true}, \bar{d}_i) \in can\_assign$
     (f1) $\forall 0 \le i \le n : (true, h_{i,0}) \in can\_assign$
     (f2) $\forall 1 \le i \le n, 0 \le j < |N_i| : (h_{i,j} \wedge N_{i,j+1}, h_{i,j+1}) \in can\_assign$, where $N_{i,j}$ is the jth element of N_i, taken in arbitrary order.
     (f3) $\forall 1 \le i \le n, |N_i| \le j < n : (h_{i,j}, h_{i,j+1}) \in can\_assign$.
     (g1) $\forall 1 \le i \le n : (\texttt{true}, g_{i,1}) \in can\_assign$
     (g2) $\forall 1 \le i \le n, \forall 1 \le j \le n-1 : (g_{i,j}, g_{i,j+1}) \in can\_assign$.
     (h) $\forall 1 \le i \le n : (v_i \wedge g_{i,n}, c_i) \in can\_assign$
     (i) $\forall 1 \le i \le n : (h_{i,n}, d_i) \in can\_assign$
     (j) $\forall 1 \le i \le n : (\texttt{true}, e_i) \in can\_assign$
   - *can_revoke* is defined as
     (a) $\forall r \in R : (\texttt{true}, r) \in can\_revoke$
   - *SMER* is defined as
     (a) $\forall 1 \le i \le n : (c_i, \bar{c}_i) \in SMER$
     (b) $\forall 1 \le i \le n : (d_i, \bar{d}_i) \in SMER$
     (c) $\forall 1 \le i \le n : (v_i, e_i) \in SMER$

3. $goal = \{a_i, b_i, \bar{c}_i, \bar{d}_i, e_i : 1 \le i \le n\}$.

b_i serves as a flag that (when present in the state) indicates that the plan passed through a state containing c_i and $\overline{d}_i$. a_i serves as a flag indicating that the plan passed though a state containing

$\overline{c}_i$ and d_i. $h_{i,n}$ serves as a flag indicating that a plan passed through a state containing all nodes v_j that are not neighbors of v_i. This means that if all v_j not in any clique of size k are reachable, then $h_{i,n}$ are reachable for all v_i in the clique. For example, suppose that $V = \{v_1, v_2, v_3, v_4, v_5\}$ and the maximum clique is $C = \{v_1, v_2, v_3\}$; then for every $s \in [4..5]$, there exists $t \in [1..3]$ such that $(v_s, v_t) \notin E$. From the definitions of $h_{i,j}$ and N_i, it follows that roles $h_{1,n}$, $h_{2,n}$, and $h_{3,n}$ can be added after roles v_4 and v_5 have been added. Roles $g_{i,j}$ are introduced for the purpose of constructing a plan with size anti-monotonic to k.

Note that, starting from the initial state, c_i must be revoked in order to add a_i, and d_i must be revoked in order to add b_i. This means that in order to reach the goal, 1) after a_i is added, c_i must be added in order to add b_i, and 2) after b_i is added, d_i must be added in order to add a_i. Since v_i and $g_{i,n}$ are the preconditions of c_i, and $h_{i,n}$ is the precondition of d_i, if there exists a plan, then $g_{i,n}$ and v_i must appear between $UR(c_i)$ and $UA(c_i)$ in the plan, and $h_{i,n}$ must appear between $UR(d_i)$ and $UA(d_i)$ in the plan, where *UA* and *UR* abbreviate *UserAssign* and *UserRevoke*, respectively.

Each role has a precondition of size at most 2, so I satisfies the $|pre| \leq 2$ restriction. Each role has at most one state change rule in *can_assign* and *can_revoke*, so I satisfies the $\overline{D}$ restriction. Each role is allowed to be unconditionally revoked, so I satisfies the $\overline{CR}$ restriction. Negation is specified only as SMER constraints, so I satisfies the $\overline{EN}$ restriction. Each role appears in at most one *SMER* constraint, so I satisfies the $|SMER(r)| \leq 1$ restriction. Thus, I is in the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |pre| \leq 2, |SMER(r)| \leq 1]$.

Lemma 25 Suppose $G = (V, E)$ has a clique of size k. Then I has a plan of size $n^2 + 13n - k$ or less.

PROOF: Let C be a clique in G of size k. Consider the sequence of actions $P = P_1.P_2.P_3.P_4.P_5.P_6.P_7.P_8$ where $UA(\langle r_1, \ldots, r_n \rangle)$ abbreviates $UA(r_1) \ldots UA(r_n)$.

1. P_1 is the concatenation of the sequences $\langle UR(c_i), UA(\bar{c}_i), UA(a_i) \rangle$ for $v_i \in V - C$. Note: $|P_1| = 3n - 3k$.
2. P_2 is the concatenation of the sequences $\langle UR(d_i), UA(\bar{d}_i), UA(b_i) \rangle$ for $v_i \in C$. Note: $|P_2| = 3k$.
3. P_3 is the concatenation of the sequences $\langle UA(v_i) \rangle$ for $v_i \in V - C$. Note: $|P_3| = n - k$.
4. P_4 is the concatenation of the sequences $\langle UA(H_i) \rangle$ for $v_i \in C$, where $H_i = \langle h_{i,0}, h_{i,1}, \ldots, h_{i,n} \rangle$. Note : $|H_i| = n + 1$, so, $|P_4| = (n + 1) \times k$.
5. P_5 is the concatenation of the sequences $\langle UR(\bar{d}_i), UA(d_i), UR(c_i), UA(\bar{c}_i), UA(a_i), UR(d_i), UA(\bar{d}_i) \rangle$ for $v_i \in C$. Note: $|P_5| = 7k$.
6. P_6 is the concatenation of the sequences $\langle UA(G_i) \rangle$ for $V_i \in V - C$, where $G_i = \langle g_{i,1}, g_{i,2}, \ldots, g_{i,n} \rangle$. Note : $|G_i| = n$, thus, $|P_6| = n \times (n - k)$.
7. P_7 is the concatenation of the sequences $\langle UR(\bar{c}_i), UA(c_i), UR(d_i), UA(\bar{d}_i), UA(b_i), UR(c_i), UA(\bar{c}_i) \rangle$ for $v_i \in V - C$. Note: $|P_7| = 7n - 7k$.
8. P_8 is the concatenation of the sequences $\langle UR(v_i) \rangle$ for $v_i \in V - C$. Note: $|P_8| = n - k$.
9. P_9 is the concatenation of the sequences $\langle UA(e_i) \rangle$ where $v_i \in V$. Note: $|P_9| = n$.

The order of P_1 and P_2 can be switched. From the definition of the *can_assign* relation for the roles $h_{i,j}$, it follows that for every $v_i \in C$, the sequence of actions $UA(H_i)$ can be executed in a state where $\forall v_m \in V - C : (v_i, v_m) \notin E$ is true. This means that $UA(H_i)$ for all $v_i \in C$ can be executed in a state where every $v_m \in V - C$ is `true`. The sequence of actions $UA(G_i)$ can be executed from any state. Thus, P_4 is a feasible subsequence of P. Finally, e_i are added to the plan for all $v_i \in V$, signaling the end of the plan. It is straightforward to verify that P is indeed a plan for I and $|P| = n^2 + 13n - k$. □ Lemma 25

Lemma 26 Suppose there exists a plan for I of length $n^2 + 13n - k$ or less. Then G has a clique of size k.

PROOF: Let P be a plan for I such that $|P| \leq n^2 + 13n - k$.

For all $1 \leq i \leq n$, P must contain $UA(e_i)$ and also, based on UA and the pre-conditions in the *can_assign* rules, either

$$\begin{aligned}&\langle UR(c_i), \ldots, UA(\bar{c}_i), \ldots, UA(a_i), \ldots, UR(\bar{c}_i), \ldots, UA(G_i), UA(c_i), \ldots, UR(d_i), \ldots,\\ &UA(\bar{d}_i), \ldots, UA(b_i), \ldots, UR(c_i), \ldots, UA(\bar{c}_i)\rangle\end{aligned} \tag{1}$$

or

$$\begin{aligned}&\langle UR(d_i), \ldots, UA(\bar{d}_i), \ldots, UA(b_i), \ldots, UR(\bar{d}_i), \ldots, UA(H_i), UA(d_i), \ldots, UR(c_i), \ldots,\\ &UA(\bar{c}_i), \ldots, UA(a_i), \ldots, UR(d_i), \ldots, UA(\bar{d}_i)\rangle.\end{aligned} \tag{2}$$

Let W be the set of vertices v_i such that P contains $UA(v_i)$. P must also contain $UR(v_i)$ in order to add e_i.

For $v_t \in V - W$, P does not contain $UA(v_t)$, so the precondition for $UA(c_t)$ in the sequence (1) is never satisfied, so P contains the sequence (2).

Let $v_s \in V - W$. If there is no edge between v_s and v_t (*i.e.*, $(v_s, v_t) \notin E$), then from the definition of *can_assign*, v_s is in the precondition of $UA(H_t)$. Since $UA(H_t) \in P$, it follows that $UA(v_s) \in P$ contradicting that $v_s \in V - W$. Thus, there is an edge $(v_s, v_t) \in E$. Since this is true for all v_s and v_t in $V - W$, $V - W$ is a clique.

The length of each sequence of the form (1) is $10+|V|$, because $|UA(c_i)| = |UA(d_i)| = |UA(\overline{c_i})| = |UA(\overline{d_i})| = |UA(a_i)| = |UA(b_i)| = |UR(c_i)| = |UR(d_i)| = |UR(\overline{c_i})| = |UR(\overline{d_i})| = 1$ and $|UA(G_i)| = |V|$.

The length of each sequence of the form (2) is $10 + (|V| + 1)$, because $|UA(c_i)| = |UA(d_i)| = |UA(\overline{c_i})| = |UA(\overline{d_i})| = |UA(a_i)| = |UA(b_i)| = |UR(c_i)| = |UR(d_i)| = |UR(\overline{c_i})| = |UR(\overline{d_i})| = 1$ and $|UA(H_i)| = |V| + 1$.

Let p denote the sum of the lengths of the above sequences that appear in P. P contains a sequence of type (2) for each vertex in $V - W$, and a sequence of type (1) or (2) for each vertex in W. To obtain a lower bound on $|P|$, we suppose that P contains the shorter of these two kinds of sequences (namely, the former) for each vertex in W.

Therefore, the sum of the lengths of these sequences in P is $(10+|V|)|W|+(10+|V|+1)|V-W|$. P must also contain a $UA(e_i)$ for each vertex in V, and $|UA(v_i)|$ and $|UR(v_i)|$ for each vertex in W, so $|P| \geq (10 + |V|)|W| + (10 + |V| + 1)|V - W| + |V| + 2|W|$. The right side can be simplified to (recall that $n = |V|$) $n^2 + 12n + |W|$, which can be rewritten as $n^2 + 13n - (n - |W|)$, which equals $n^2 + 13n - |V - W|$. Thus, we have $|P| \geq n^2 + 13n - |V - W|$.

By assumption, $|P| \leq n^2 + 13n - k$. Combining these two inequalities, we have $n^2 + 13n - |V - W| \leq |P| \leq n^2 + 13n - k$, so $|V - W| \geq k$. Thus, G contains a clique of size k. □ Lemma 26

From Lemmas 25 and 26 it follows that BRE for the class $[\overline{D}, \overline{CR}, \overline{EN}, |ppre| \leq 2, |SMER(r)| \leq 1]$ is NP-hard. □ Lemma 24

Theorem 27 *Bounded Reachability* (BRE) for the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |pre| \leq 2, |SMER(r)| \leq 1]$ is NP-hard.

PROOF: The proof is very similar to that of Lemma 24. The theorem can be easily proven by replacing the *can_assign* rules for v_i and c_i with the following rules:

- $\forall 1 \leq i \leq n : (g_{i,n}, v_i) \in$ *can_assign*
- $\forall 1 \leq i \leq n : (v_i, c_i) \in$ *can_assign*

□

Theorem 28 *Bounded Reachability* (BRE) for the problem class $[\overline{D}, \overline{EN}, \overline{CR}, |goal| = 1, |SMER(r)| \leq 1]$ is NP-hard.

PROOF: As discussed in Section 4, reachability of a goal containing multiple roles can be transformed to reachability of a singleton goal, and the transformation preserves all restrictions except $|pre| \leq k$ and $|ppre| \leq k$. Therefore, the proof of Lemma 24 can directly be applied to show that reachability for the problem class $[\overline{D}, \overline{EN}, \overline{CR}, |goal| = 1, |SMER(r)| \leq 1]$ is NP-hard. □

Theorem 29 *Bounded Reachability (BRE) for the problem class* $[\overline{D}, \overline{EN}, |ppre| \leq 1]$ *(no disjunction, SMER constraints allowed but no explicit negation, at most one positive literal in prerequisites) is NP-hard.*

PROOF: The proof is by reduction from the `CLIQUE` problem, which is known to be NP-complete [Kar72]. The proof is similar in structure to the proof of Theorem 4.15 in [BN95], but our construction and proof are significantly more complicated, because we are dealing with a significantly more restricted problem class. Given a graph $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ and an integer k, we construct a reachability problem instance I that is in the problem class $[\overline{D}, \overline{EN}, |ppre| \leq 1]$ and show that G has a clique of size k if and only if I has a plan of size at most $15n - 2k$.

Define $I = (\gamma, goal, \psi)$ where

1. $\gamma = (R, UA)$. Corresponding to each vertex $v_i \in V$ there is a role $v_i \in R$, and for each such role v_i there are additional roles $a_i, b_i, c_i, d_i, e_i, f_i, g_i, h_i$. Thus $R = \{v_i, a_i, b_i, c_i, d_i, e_i, f_i, g_i, h_i : 1 \leq i \leq n\}$, and $UA = \{c_i, d_i : 1 \leq i \leq n\}$.
2. $\psi = (can\_assign,\ can\_revoke, SMER)$ where
   - *can_assign* is defined as
     (a) $\forall 1 \leq i \leq n : (\texttt{true}, v_i) \in can\_assign$
     (b) $\forall 1 \leq i \leq n : (d_i, a_i) \in can\_assign$
     (c) $\forall 1 \leq i \leq n : (c_i, b_i) \in can\_assign$
     (d) $\forall 1 \leq i \leq n : (\texttt{true}, c_i) \in can\_assign$
     (e) $\forall 1 \leq i \leq n : (\texttt{true}, d_i) \in can\_assign$
     (f) $\forall 1 \leq i \leq n : (a_i, e_i) \in can\_assign$
     (g) $\forall 1 \leq i \leq n : (b_i, f_i) \in can\_assign$

(h) $\forall 1 \leq i \leq n : (\texttt{true}, g_i) \in \mathit{can\_assign}$
(i) $\forall 1 \leq i \leq n : (\texttt{true}, h_i) \in \mathit{can\_assign}$

- *can_revoke* is defined as
  (a) $\forall 1 \leq i \leq n : (\texttt{true}, v_i) \in \mathit{can\_revoke}$
  (b) $\forall 1 \leq i \leq n : (v_i, a_i) \in \mathit{can\_revoke}$
  (c) $\forall 1 \leq i \leq n : (v_i, b_i) \in \mathit{can\_revoke}$
  (d) $\forall 1 \leq i \leq n : (\texttt{true}, c_i) \in \mathit{can\_revoke}$
  (e) $\forall 1 \leq i \leq n : (\texttt{true}, d_i) \in \mathit{can\_revoke}$
- *SMER* is defined as
  (a) $\forall 1 \leq i \leq n : (d_i, b_i) \in \mathit{SMER}$
  (b) $\forall 1 \leq i \leq n : (a_i, c_i), (v_i, c_i) \in \mathit{SMER}$
  (c) $\forall 1 \leq i, j \leq n, (v_i, v_j) \notin E : (d_i, v_j) \in \mathit{SMER}$
  (d) $\forall 1 \leq i \leq n, (a_i, g_i) \in \mathit{SMER}$
  (e) $\forall 1 \leq i \leq n, (b_i, h_i) \in \mathit{SMER}$

3. $\mathit{goal} = \{v_i, e_i, f_i, g_i, h_i : 1 \leq i \leq n\}$.

Note that all the roles have positive preconditions of size at most 1, Thus, I satisfies the $|ppre| \leq 1$ restriction. Because each role $r \in R$ has at most one state change rule in *can_assign* and *can_revoke*, I satisfies the $\overline{D}$ restriction. Because negation is specified only as SMER constraints, I satisfies the $\overline{EN}$ restriction. Thus, I is in the problem class $[\overline{D}, \overline{EN}, |ppre| \leq 1]$.

Lemma 30 If G has a clique of size k, then I has a plan of size $15n - 2k$.

PROOF: Let C be a clique in G of size k. Consider the sequence of actions $P = P_1.P_2.P_3.P_4.P_5.P_6.P_7.P_8, P_9.P_{10}.P_{11}.P_{12}.P_{13}.P_{14}$ where *UA* and *UR* abbreviate *UserAssign* and *UserRevoke* respectively.

1. P_1 is the concatenation of the sequences $\langle UR(c_i), UA(a_i), UA(e_i)\rangle$ where $v_i \in V - C$. $|P_1| = 3n - 3k$.
2. P_2 is the concatenation of the sequences $\langle UR(d_i), UA(b_i), UA(f_i)\rangle$ where $v_i \in C$. $|P_2| = 3k$.
3. P_3 is the concatenation of the sequences $\langle UR(d_i)\rangle$ where $v_i \in V - C$. $|P_3| = n - k$.
4. P_4 is the concatenation of the sequences $\langle UR(c_i)\rangle$ where $v_i \in C$. $|P_4| = k$.
5. P_5 is the concatenation of the sequences $\langle UA(v_i)\rangle$ for all $1 \leq i \leq n$. $|P_5| = n$.
6. P_6 is the concatenation of the sequences $\langle UR(a_i)\rangle$ where $v_i \in V - C$. $|P_6| = n - k$.
7. P_7 is the concatenation of the sequences $\langle UR(b_i)\rangle$ where $v_i \in C$. $|P_7| = k$.
8. P_8 is the concatenation of the sequences $\langle UR(v_i)\rangle$ where $v_i \in V - C$. $|P_8| = n - k$.
9. P_9 is the concatenation of the sequences $\langle UA(c_i), UA(b_i), UA(f_i), UR(c_i)\rangle$ where $v_i \in V - C$. $|P_9| = 4n - 4k$.

10. P_{10} is the concatenation of the sequences $\langle UA(d_i), UA(a_i), UA(e_i), UR(d_i)\rangle$ where $v_i \in C$. $|P_{10}| = 4k$.

11. P_{11} is the concatenation of the sequences $\langle UA(v_i)\rangle$ where $v_i \in V - C$. $|P_{11}| = n - k$.

12. P_{12} is the concatenation of the sequences $\langle UR(a_i)\rangle$ where $v_i \in C$. $|P_{12}| = k$.

13. P_{13} is the concatenation of the sequences $\langle UR(b_i)\rangle$ where $v_i \in V - C$. $|P_{13}| = n - k$.

14. P_{14} is the concatenation of the sequences $\langle UA(g_i)UA(h_i)\rangle$ where $1 \leq i \leq n$. $|P_{14}| = 2n$.

It is easy to see that P is a plan for I and that $|P| = 15n - 2k$. □ Lemma 30

Lemma 31 Suppose I has a plan of length at most $15n - 2k$. Then G has a clique of size k.

PROOF: The proof is similar to the proof of Lemma 26. □ Lemma 31

From Lemmas 30 and 31 it follows that G has a clique of size k if and only if I has a plan of size at most $15n - 2k$. Thus, BRE for $[\overline{D}, \overline{EN}, |ppre| \leq 1]$ is NP-hard. □ Theorem 29

6 Other Analysis Problems

6.1 Reachability, Availability, and Containment Analysis Problems

In this section, we consider the following analysis problems: permission-role and user-permission reachability, user-role availability, role-role containment, and permission-role containment.

Permission-Role Reachability Analysis. Consider queries of the form "Can administrators in administrative roles in A assign a permission p to all roles in *goal*?". miniRBAC and miniARBAC specifications for the user-role and permission-role assignment relations are symmetrical, except that miniARBAC does not include an analogue of SMER constraints for permissions, so permission-role reachability analysis can be performed in exactly the same manner as user-role reachability analysis with $SMER = \emptyset$. Thus, the results of Section 5 apply directly.

We now consider a more general case: "Can administrators in administrative roles in A assign permissions $\{p_1, \ldots, p_n\}$ to all roles in *goal*?". Note that role assignments for different permissions are independent, just as role assignments for different users are independent (*cf.* the Single User simplification in Section 4.1). Thus, this analysis problem is true if and only if the administrators in administrative roles in A can assign every permission p_i to all roles in *goal*. All complexity results for permission-role reachability analysis with single permission carry over to permission-role reachability analysis with multiple permissions.

User-Permission Reachability Analysis. Consider queries of the form "Can administrators in administrative roles in A grant user u permission p?". Such a query can be answered by checking whether there exists a role r such that (1) user u is initially a member of r or the administrators can add u to r, and (2) permission p is already granted to r or administrators can grant p to r. Thus, the problem can be transformed into a polynomial number of user-role and permission-role reachability analysis problems that satisfy the same structural restrictions ($\overline{N}$, $\overline{D}$, *etc.*) as the original problem. This implies that results in Section 5 showing user-role reachability for a problem class is in P, NP, or PSPACE also apply to user-permission reachability (for a single permission)

for that problem class. To show that the NP-hardness and PSPACE-hardness results in Section 5 also carry over, we sketch a polynomial-time reduction from permission-role reachability analysis to user-permission reachability analysis (for a single permission). Recall from above that complexity results for user-role reachability carry over to permission-role reachability. Given a permission-role reachability query involving permission p, role r, and administrative roles in A, we add a new user u_0 to the policy, make u_0 a member of role r in the initial state, delete all the *can_assign* and *can_revoke* rules (keeping the *can_assign_p* and *can_revoke_p* rules unchanged), and ask whether permission p can be granted to user u_0 by administrators in roles in A. u_0 is always a member of r and of no other roles, so p can be granted to u_0 iff p can be granted to r.

We now consider a more general user-permission query with multiple permissions: "Can administrators in A give user u permissions $\{p_1, \ldots, p_n\}$?". To solve this problem, we first perform permission-role reachability analysis to compute, for $i = 1..n$, the set R_i of roles that can be granted permission p_i by administrators in A. Next, we perform user-role reachability analysis to check if there exist $g_1 \in R_1, \ldots, g_n \in R_n$ such that u can be assigned all roles in $\{g_1, \ldots, g_n\}$. The given set of permissions is reachable iff such $g_1, \ldots, g_n$ exist. Note that role assignments for different permissions are independent, just as role assignments for different users are independent (*cf.* the Single User simplification in Section 4.1), so each g_i can be selected independently from R_i. Thus, user-permission reachability analysis for n permissions can be transformed into $O(|R| \times n)$ permission-role reachability analysis problems and $O(|R|^n)$ user-role reachability analysis problems that satisfy the same structural restrictions as the original problem. It follows that the PSPACE-completeness result for user-role reachability in Section 5 carries over to user-permission reachability with multiple permissions.

Availability Analysis. User-role availability analysis checks whether a given user u is a member of a given role r in all policies reachable from the initial policy by actions of a given set of administrators A. The problem can be simplified in the same ways as user-role reachability (*cf.* Section 4.1). A simplified user-role availability analysis problem instance has the form $I = (\gamma, goal, \psi)$ where $\gamma = (R, UA)$ is a simplified miniRBAC policy, $\psi =$(*can_assign*, *can_revoke*, *SMER*) is a simplified miniARBAC policy, and *goal* is a set of roles. The answer to I is `true` iff in every state γ' reachable from γ via ψ (*i.e.*, $\gamma \rightarrow_\psi^* \gamma'$), the user is a member of at least one role in *goal* in state γ'. I can be solved as follows.

1. Suppose $goal \cap UA = \emptyset$; *i.e.*, no role in *goal* is in the initial state. Then the answer is `false`.
2. Suppose ψ satisfies the $\overline{CR}$ restriction (every role can be unconditionally revoked). The answer is `false`, because u's membership in every role in *goal* can be revoked.
3. Otherwise we transform the user-role availability analysis problem instance I to a user-role reachability analysis problem instance $I' = (\gamma', goal', \psi')$ as follows.
   - $goal' = \{\bar{r} : r \in goal\}$ where each $\bar{r}$ is a new role.
   - Let $\gamma' = (R', UA)$ where $R' = R \cup goal'$.
   - $\psi' =$(*can_assign*$'$, *can_revoke*$'$, *SMER*$'$) where
     (1) *can_assign*$' =$ *can_assign* $\cup\{(\texttt{true}, \bar{r}) : \bar{r} \in goal'\}$,
     (2) *can_revoke*$' =$ *can_revoke* $\cup\{(\texttt{true}, \bar{r}) : \bar{r} \in goal'\}$, and
     (3) *SMER*$' =$ *SMER* $\cup \{(r, \bar{r}) : r \in goal\}$.

We show that I and I' have opposite answers. Suppose the answer to I' is `true`. Then there exists a state $\gamma' = (R, UA')$ such that $\gamma \rightarrow^*_{\psi'} \gamma'$ and $goal' \subseteq UA'$. For each $r \in goal$, $(\bar{r}, r) \in SMER'$, so $r \notin \gamma'$. Thus, $goal \cap \gamma' = \emptyset$. This implies that the answer to I is `false`. Conversely, it is easy to show that if the answer to I' is `false`, then the answer to I is `true`. Thus, availability analysis can be reduced to reachability analysis in polynomial time, and the transformation preserves restrictions $\overline{EN}, \overline{CR}, \overline{D}, |goal| \leq k, |pre| \leq k$, and $|ppre| \leq k$. Thus, several of the complexity results and algorithms in Section 5 carry over to availability analysis. For example, availability analysis for the problem classes $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |goal| \leq k]$ and $[|pre| \leq 1, |goal| \leq k]$ are in P.

Containment Analysis A role-role containment problem instance has the form [LT06]: given a set A of administrative roles, an initial miniRBAC policy γ, and miniARBAC policy ψ, is every member of role r_1 also a member of role r_2 in every state reachable from γ by actions of administrators in A allowed by ψ? Let $I_c = (\gamma, A, r_1, r_2, \psi)$ denote this role-role containment analysis instance.

Theorem 32 Role-role containment analysis for the problem class without any restrictions is PSPACE-complete.

PROOF: Recall from Theorem 2 that RE for the problem class without any restrictions is PSPACE-hard. We give a polynomial-time reduction from reachability analysis to role-role containment analysis; this implies that role-role containment analysis is also PSPACE-hard. Given a (simplified) reachability problem instance $I = (\gamma, \{g_1, \ldots, g_n\}, \psi)$, we first transform it to a reachability problem instance with a singleton goal $\{g\}$, as described in Section 4. Let $I_1 = (\gamma_1, \{g\}, \psi_1)$ denote the resulting reachability problem instance. Next, we transform I_1 to a role-role containment analysis problem $I_c = (\gamma_2, g, r_2, \psi_2)$, where r_2 is a new role, γ_2 is obtained from γ_1 by adding r_2 to the set of roles (but not the user-role assignment), and γ_2 and ψ_2 are "unsimplified" (*cf.* the simplifications in Section 4.1) by adding dummy values for the implicit user, the implicit administrative role, and the relations related to permissions. Note that r_2 is unreachable, because r_2 is not in the initial user-role assignment in γ_2, and there are no *can_assign* rules for r_2. Based on this, it is easy to see that the answer to the original reachability problem instance I is true iff the answer to I_c is false.

We show that role-role containment analysis is in PSPACE, by showing that it is in co-NPSPACE, and recalling that PSPACE = co-NPSPACE (this is a corollary of Savitch's theorem). Role-role containment analysis is in co-NPSPACE, because a non-deterministic Turing Machine can show that the answer to a role-role containment problem instance $I_c = (\gamma, A, r_1, r_2, \psi)$ is false by guessing, one step at a time, a plan that leads to a miniRBAC state that contains r_1 but not r_2. At each step, the Turing Machine stores only the current state (*i.e.*, current miniRBAC policy), whose size is polynomial in the size of the problem instance. □

Theorem 33 Role-role containment analysis for the problem class $[\overline{R}]$ is co-NP-complete.

PROOF: Recall that a language is co-NP-complete if its complement is NP-complete, so we need to show that the complement of role-role containment analysis for $[\overline{R}]$ is NP-complete. It is NP-hard because reachability analysis for $[\overline{R}]$ is many-one reducible to it, using the same reduction as in the first half of the proof of Theorem 32, and reachability analysis for $[\overline{R}]$ is NP-hard, according to Theorem 17. It is in NP by the same reasoning as in the second half of the proof of Theorem 32, together with the observation that guessing and checking the plan can be done polynomial time, because in the absence of revocation, the length of the plan is bounded by the number of roles,

because the plan contains only *UserAssign* actions, and repeating a *UserAssign* action has no effect. □

A permission-role containment problem instance has the form: in every state reachable from the initial state, is every principal who has permission p also a member of at least one role in *Role*? An example is: is every user with permission $[assignGrade, GradeBook]$ a member of TA or Faculty? The permission-role containment analysis problem can be reduced to the negation of the following problem instance: does there exist a state, a user u and a role $r \notin Role$ such that u is a member of r and r has permission p? This problem can be transformed into a polynomial number of user-role reachability and role-permission reachability analysis problems by first computing a set of roles R_1 to which an administrator can grant p and then checking if there exist a role $r \in (R_1 - Role)$ and a user u such that r is reachable from $\{r_1 | (u, r_1) \in UA\}$. If so, the containment analysis result is false and true otherwise. The above reduction does not change the policy. Thus the permission-role containment problem is in P for policies where *can_assign* and *can_assignp* each satisfy the restrictions defining one of the problem classes (in Figure 2) for which user-role reachability is in P.

Role-permission and permission-permission containment analysis problems are not common in practice and hence are not considered in this paper.

6.2 Information Flow Analysis of miniARBAC

An information flow analysis query asks: given a miniRBAC policy γ, a miniARBAC policy ψ, a set A of administrative roles, and objects (i.e., resources) o_1 and o_2, can information flow from o_1 to o_2 by administrative actions of administrators in roles in A and normal actions of all users?

Our solution to this analysis problem builds on Osborn's definition of an *information flow graph* for an RBAC policy γ [Osb02]. Each node in the graph represents an object. There is an edge $o_1 \rightarrow o_2$ in the graph if information can flow directly from object o_1 to object o_2, i.e., if there exists a user u that has permission in γ to read from o_1 and write to o_2. For simplicity, we assume the only operations on objects are read and write; other permissions can be represented as read and write permissions for the purposes of information flow analysis. Note that [Osb02] considers only RBAC, not ARBAC.

Our algorithm constructs a graph called the *information flow transition graph*. Each state in the graph is a pair (γ, O) where γ is an RBAC policy and O is a set of objects; the state means that the system can reach a state in which the miniRBAC policy is γ and information has flowed from o_1 to the objects in O.

The graph contains a labeled edge (transition) $(\gamma_1, O_1) \stackrel{\alpha}{\hookrightarrow} (\gamma_2, O_2)$, where α is an administrative action permitted by ψ for an administrative role in A, if γ_2 is obtained from γ_1 by execution of α, and O_2 is the set of objects reachable from O_1 in the information flow graph for γ_2. Our algorithm starts from (γ, O), where O is the set of objects reachable from o_1 in the information flow graph for the initial miniRBAC policy γ, and computes the reachable states. The answer to the information flow query is true iff there exists a reachable state whose second component contains o_2. As an optimization, it is sufficient to consider only users with distinct sets of roles in the initial miniRBAC policy.

The following theorems establish the complexity of information-flow analysis for miniARBAC for some problem classes.

Theorem 34 Information flow analysis for miniARBAC without any restrictions is PSPACE-complete.

PROOF: Recall that user-role reachability analysis of ARBAC without any restrictions is PSPACE-complete (Theorem 2). Below, we show that information flow analysis for the problem class without any restriction is PSPACE-hard by giving a polynomial-time reduction from user-role reachability analysis to information-flow analysis. Given a user-role reachability problem instance, we first transform it to a user-role reachability problem instance with a singleton goal $\{g\}$, as described in Section 4. We then introduce two new objects o_1 and o_2, and modify the initial permission-role assignment to grant to role g permissions to read from o_1 and write to o_2. Note that no users are members of role g in the initial user-role assignment. It is easy to see that information can flow from o_1 to o_2 iff role g is reachable.

Information flow analysis of miniARBAC is in PSPACE by similar reasoning as in the proof that user-role reachability analysis is in PSPACE (Theorem 2). The main point is that a non-deterministic Turing Machine can guess a path in the information-flow transition graph, one step at a time, storing only the current state (node) after each step. The size of each node is polynomial in the size of the problem instance. □

Theorem 35 Information flow analysis for the problem class $[\overline{N}]$ can be solved in polynomial time.

PROOF: We describe a simple polynomial algorithm to solve the problem, based on the observation that, without negative preconditions, revocation is not useful for achieving any information flow goal. First, the algorithm computes the maximal set of roles that can be assigned to each user, by starting from the initial RBAC policy and repeatedly assigning roles until no more roles can be assigned. Next, the algorithm in [Osb02] is used to construct an information flow graph G based on these sets of roles. The information flow analysis problem is true if and only if there exists a path from the given source object o_1 to the given target object o_2 in G. □

Theorem 36 Information flow analysis problem for the problem classes $[\overline{R}]$, $[\overline{R}, \overline{EN}]$, and $[\overline{R}, \overline{EN}, |goal| = 1, |SMER(r)| \leq 1]$ is NP-complete.

PROOF: NP-hardness of information flow analysis for these problem classes is proved from NP-hardness of user-role reachability analysis for these problem classes (see Theorem 17), using the same reduction as in the proof of Theorem 34. That reduction preserves the restrictions $\overline{R}, \overline{EN}, |goal| = 1$, and $|SMER(r)| \leq 1$. The problem is in NP because, after guessing a sequence P of $UserAssign(r_i)$ actions, a Turing Machine can check in polynomial time whether P is allowed by the miniARBAC policy and whether P allows information to flow from a given source object to a given target object. It is sufficient to consider polynomial-length sequences P, because without revocation, it is sufficient to add each role at most once. □

7 Extensions

In this section, we consider user-role reachability analysis for two extensions of miniARBAC. The first extension allows role hierarchy, and the second extension allows a role to be both a regular role and an administrative role.

7.1 User-Role Reachability Analysis in Hierarchical RBAC

In this subsection, we first discuss reachability analysis for miniARBAC controlling changes to miniHRBAC policies with fixed role hierarchy, and then consider changes to the role hierarchy. *Fixed role hierarchy* means that the *can_modify* relation is empty, so the role hierarchy does not change.

Reachability analysis with fixed role hierarchy. It is easy to show, based on our results for non-hierarchical policies, that reachability analysis for miniARBAC with fixed role hierarchy is PSPACE-hard.

Theorem 37 For miniARBAC policies controlling changes to miniHRBAC policies with fixed role hierarchy (i.e., the *can_modify* relation is empty), *Reachability* (RE) is PSPACE-complete.

Proof: PSPACE-hardness of RE for this problem class is a corollary of Theorem 2, because miniRBAC is a restricted case of miniHRBAC with a fixed role hierarchy. Reachability for this problem class is in PSPACE by similar reasoning as in Theorem 2: a non-deterministic Turing Machine can guess a plan one step at a time, storing at each step only the current state. □

To extend other complexity results from the non-hierarchical case to the hierarchical case, we show how to transform reachability analysis problems for hierarchical policies into reachability analysis problems for non-hierarchical policies. The transformation makes the effects of inherited membership explicit; in the original problem, the effects of inherited membership are implicit in the semantics of preconditions.

Our transformation assumes that the SMER constraints do not conflict with the role hierarchy. Two kinds of conflicts are identified in [AH07]. A SMER constraint (r_1, r_2) conflicts with the role hierarchy if (1) r_1 and r_2 have a common senior role, or (2) r_1 is senior to r_2 or *vice versa*. For example, consider a policy in which *Manager* is senior to $Employee_1$ and $Employee_2$. The SMER constraint $(Employee_1, Employee_2)$ contains a conflict of type (1); intuitively, this reflects the fact that this SMER constraint forces *Manager* to be empty, because any member of *Manager* would be an implicit member of both $Employee_1$ and $Employee_2$. The SMER constraint $(Employee_1, Manager)$ contains a conflict of type (2); this reflects the fact that it also forces *Manager* to be empty, for similar reasons.

Let $I_h = (\gamma_h, goal_h, \psi_h)$ be a reachability problem instance for hierarchical RBAC with $\gamma_h = (R, UA, \succeq)$, $\psi_h = (can\_assign_h, can\_revoke_h, SMER_h)$, and $goal_h = \{g_1, g_2, \ldots, g_k\}$. Define a set of reachability problem instances for non-hierarchical RBAC as follows.

- Let $\gamma = (R, UA)$.
- The *can_assign* and *can_revoke* relations are generated in two steps from can_assign_h and can_revoke_h.
  1. For each $(c, r) \in can\_assign_h$, and for each $\neg t \in c$, replace $\neg t$ with $\bigwedge_{s \in Senior(t)} \neg s$. Transform the can_revoke_h relation in a similar manner. Let can_assign' and can_revoke' denote the transformed relations.
  2. For each $(c^+ \wedge c^-, r) \in can\_assign'$, where c^+ is a conjunction $r_1 \wedge \ldots \wedge r_k$ of positive roles, and c^- is a conjunction of negative roles, generate the Cartesian product $PosConjunct = Senior(r_1) \times \ldots \times Senior(r_k)$. For each $(r'_1, \ldots, r'_k) \in PosConjunct$ such that there is

no role that appears in both $(r'_1, \ldots, r'_k)$ and c^-, add the rule $(r'_1 \wedge \ldots \wedge r'_k \wedge c^-, r)$ to *can_assign*. Generate *can_revoke* from the *can_revoke'* in the same manner.

- Let $SMER = \{(r, s) : (r', s') \in SMER_h \wedge r \succeq r' \wedge s \succeq s'\}$.
- $Goals = Senior(g_1) \times Senior(g_2) \times \ldots \times Senior(g_n)$.

Note that transforming the goal $\{g_1, \ldots, g_n\}$ into $Goals$ ensures that if a role senior to g_i is reachable, then g_i is reachable. As a result, we do not need to transform the postconditions in *can_assign* and *can_revoke* rules. The answer to I_h is `true` if and only if there exists a $goal \in Goals$ such that the answer to $I = (\gamma, goal, \psi)$ is `true`. Moreover, it is easy to show that any plan for I_h is also a plan for I, and vice versa.

Starting from our results for analysis of non-hierarchical policies, we can derive results for analysis of a class of hierarchical policies, defined by some restrictions on the policies, by determining (1) the restrictions satisfied by the transformed policies, (2) the size of a transformed policy relative to the size of the original (hierarchical) policy, and (3) the number of transformed problem instances, *i.e.*, the number of transformed goals. We consider these issues in turn.

The restrictions $\overline{N}$, $\overline{EN}$, $\overline{R}$, $\overline{CR}$, $|ppre| \leq 1$, and $|goal| \leq k$ are preserved by the transformation; the proofs are straightforward. The transformation may invalidate other restrictions. Specifically, steps 1 and 2 in the transformation may invalidate the restrictions $|pre| \leq 1$ and $\overline{D}$, respectively, and the transformation from $SMER_h$ to $SMER$ may invalidate the $|SMER(r)| \leq 1$ restriction.

The size of the transformed policy might not be polynomial in the size of the original policy because, in the worst case, the Cartesian product $Senior(r_1) \times \ldots \times Senior(r_k)$ in step 2 may result in addition of $O(h^{|ppre|})$ rules, where h is a bound on the number of senior roles for each role, and $|ppre|$ is a bound on the number of positive preconditions in each *can_assign* rule. Therefore, in general, the transformation may increase the size of the policy by a factor exponential in $|ppre|$. This implies, for example, that results giving polynomial-time algorithms for a problem class do not carry over to analysis of hierarchical policies, unless $|ppre|$ is bounded. We do expect that in practice, the number of positive preconditions in each *can_assign* rule is bounded by a small constant.

The transformed goals are defined by a Cartesian product $Senior(g_1) \times Senior(g_2) \times \ldots \times Senior(g_n)$. In the worst case, the number of transformed goals is $O(h^{|goal|})$, where h is as in the previous paragraph. For problem classes with the restriction $|goal| \leq k$, the number of transformed goals is polynomial in the size of the original policy.

For example, recall that reachability analysis for the problem class $[\overline{EN}, \overline{CR}, |ppre| \leq 1, |goal| \leq k]$ for non-hierarchical policies can be solved in polynomial time. Based on the above observations, we conclude that reachability analysis for the problem class $[\overline{EN}, \overline{CR}, |ppre| \leq 1, |goal| \leq k]$ for hierarchical policies can also be solved in polynomial time.

As an optimization, we can compute dependencies between roles (based on preconditions) and transform only the part of the role hierarchy relevant to the goal.

Analysis for some classes of hierarchical policies can be solved more efficiently by a direct algorithm than by the above transformation. For example, reachability analysis for hierarchical policies that satisfy the $\overline{N}$ restriction can always be solved in polynomial time, using a fixed-point algorithm similar to the algorithm for reachability analysis for non-hierarchical policies satisfying this restriction.

Reachability allowing changes to role hierarchy. Next, we show that user-role reachability analysis is PSPACE-complete when changes to the role hierarchy are allowed. Recall from Section 3 that changes to the role hierarchy $\succeq$ are controlled by the RRA policy (note that the administrative role hierarchy $\succeq_a$ is fixed). When considering changes to the role hierarchy, we also adopt a modified definition of URA and PRA in which the third component of each tuple in *can_assign*, *can_revoke*, *can_assign_p*, and *can_revoke_p* is a set of roles expressed using range notation, like in the second component of *can_modify*. We refer to this modified version as *miniARBAC with role ranges*. Actually, this matches the original definition of ARBAC97 [SBM99]. Our use of a single role in the third component of those relations in other sections of this paper is a simplification that has essentially no effect on our results when the role hierarchy is fixed, but it would have an effect here. When new roles may be added, use of range notation is significant, because it allows existing tuples in *can_assign*, *can_revoke*, *etc.*, to apply to newly added roles that fall in the specified range.

Theorem 38 When changes to the role hierarchy are allowed, user-role reachability analysis for miniARBAC with role ranges is PSPACE-complete.

PROOF: PSPACE-hardness of this problem is a corollary of Theorem 2. The proof that the problem is in PSPACE relies on the following lemma.

Lemma 39 Let R be the set of roles in a miniRBAC policy γ, and let r be a role in R. If r is reachable from γ under a miniARBAC policy ψ, then r is reachable from γ by a plan that adds at most $|R|^2$ roles.

PROOF: A new role is related to roles in R only by the specification of its parent and child in the role hierarchy, and there are at most $|R|^2$ distinct choices for these among roles in R. Creating additional roles with the same parent and child in R has no additional effect on inheritance relationships between (or membership of) roles in R. Creating multiple roles between roles r_1 and r_2 in R (e.g., creating r and r' with $r_2 \succ r' \succ r \succ r_1$) is always unnecessary for reaching a role membership goal, because the same inheritance relationship between r_1 and r_2 can be produced by creating a single role between them. □Lemma 39

To see that the problem is in PSPACE, note that a non-deterministic Turing Machine can guess a plan one step at a time, storing at each step only the current state. The size of the state is polynomial in the number of roles. Based on Lemma 39, the non-deterministic Turing machine guesses only plans that add at most a quadratic number of roles, so the number of roles (and hence the size of the state) is polynomial in the size of the problem instance. □Theorem 38

7.2 Beyond the Separate Administration Restriction

miniARBAC, like ARBAC97, requires that administrative roles and regular roles are separate, i.e., $AR \cap R = \emptyset$. We call this requirement the *separate administration restriction*. In this subsection, we consider user-role reachability analysis for a variant of miniARBAC without this restriction. In this subsection, we do not consider hierarchical RBAC (in other words, the miniARBAC policies control changes to miniRBAC policies not miniHRBAC policies), and we do not consider changes to the role hierarchy (i.e., we assume *can_modify* is empty).

Without the separate administration restriction, reachability analysis must consider plans that may contain administrative actions that change the role membership of any user. Consider the following ARBAC policy: $\mathit{can\_assign} = \{(r_{a1}, \mathit{true}, r_{a2}), (r_{a2}, \mathit{true}, r)\}$. Let $UA = \emptyset$, $\mathit{goal} = \{r\}$,

and u_t be the target user. Observe from the policy that, a user with administrative role r_{a1} cannot directly assign role r to u_t, but u_a can assign him/herself to r_{a2} and then assign role r to u_t. Thus, we cannot assume "single user" and "implicit administrative role" (*cf.* Section 4.1) in this context.

Theorem 40 *Reachability* (RE) for miniARBAC without the separate administration restriction is PSPACE-complete.

PROOF: PSPACE-hardness of reachability for this problem class is a corollary of Theorem 2. Reachability for this problem class is in PSPACE, because a non-deterministic Turing Machine can guess a plan one step at a time, storing at each step only the current state. The size of the state is polynomial in the numbers of users and roles, which do not grow, so the size of the state is polynomial in the size of the problem instance. □

8 Related Work

We classify related work on security policy analysis into three categories, which focus on different and complementary analysis problems.

The first category is analysis (including enforcement) of a fixed security policy. We mention some representative papers in this category. Jajodia, Samarati, and Subrahmanian [JSS97] propose a policy language that can express positive and negative authorizations and derived authorizations (similar to delegation), and they give polynomial-time algorithms to check consistency and completeness of a given policy. Cholvy and Cuppens [CC97] use SOL-deduction to check consistency of a security policy that expresses positive and negative permissions and obligations. Bandara, Lupu, and Russo [BLR03] use abductive logic programming to detect conflicts in a policy expressed in a language based on Event Calculus that can express positive and negative authorizations, obligations, and refrain conditions. Jaeger *et al.* [JEZ03, JSZ03] give algorithms to check integrity and completeness of a Security-Enhanced Linux (SELinux) policy. Guttman *et al.* [GHRS05] describe a technique to analyze information flow in a SELinux policy.

The second category is analysis of a single change to a fixed policy or, similarly, analysis of the differences between two fixed policies. Jha and Reps [JR04] present analysis algorithms, based on push-down model checking, to check properties of a given SPKI/SDSI policy and to analyze the effects of a given change to a given policy. Fisler *et al.* [FKMT05] consider policy analysis for a subset of XACML. They give decision-diagram-based algorithms to check properties of a given policy and to compute the semantic difference of two given policies and check properties of the difference.

Work in the first two categories differs significantly from our work (and other work in the third category) by not considering the effect of sequences of changes to the policy.

The third category is analysis that considers the effect of sequences of changes to a policy; the allowed changes are determined by parts of the policy that we call "administrative policy". Harrison, Ruzzo, and Ullman [HRU76] present an access control model based on access matrices, which can express administrative policy, and show that the safety analysis problem is undecidable for that model. Following this, a number of access control systems were designed in which safety analysis is more tractable, *e.g.*, [LS77, San88, San92]. While each of these papers proposes a specific model designed with tractable analysis in mind, we start with the ARBAC97 model [SBM99] and explore the difficulty of policy analysis in a range of models obtained by combinations of simple restrictions on the policy language. Also, we consider features not considered in those papers, such

as negative preconditions, and we consider availability as well as safety (*i.e.*, reachability). Guelev, Ryan, and Schobbens [GRS04] present a low-level access control model and an algorithm to check properties of the policies; they note that the worst-case complexity of their algorithm is high and non-optimal, and they leave identification of problem classes for which it has lower complexity as future work.

Schaad and Moffett [SM02] express RBAC and ARBAC97 in Alloy, a relational modeling language, and use the Alloy analyzer [JSS00] to check separation of duty properties. They do not consider preconditions for any operations; this greatly simplifies the analysis problem. They do not present any analysis algorithms or complexity results. The Alloy analyzer translates bounded-size problem instances into SAT problems, and solves them with a SAT solver.

Li and Tripunitara [LT06] introduce two restricted versions of ARBAC97, called AATU and AAR, and give algorithms and complexity results for various analysis problems—primarily safety, availability, and containment—for those two models. The results are based on Li, Mitchell, and Winsborough's results for analysis of trust management policies [LMW05]. Our work goes significantly beyond theirs by considering negative preconditions and SMER (static mutually exclusive roles) constraints. They do not consider these features. Since we consider these features, we are driven to consider other restrictions, such as bounds on the size of preconditions, that they do not consider.

Jha *et al.* also studies reachability analysis of URA97 [JLT+08]. One difference is that we allow preconditions in the *can_revoke* relation (Section 3 gives an example of a policy that requires conditional role revocation), and they do not; as a result, their proof that reachability analysis for URA97 is PSPACE-hard is stronger than our PSPACE-hardness result, while our proof that reachability analysis for URA97 with conditional role revocation is in PSPACE is stronger than their "in PSPACE" result. Also, while they consider a few problem classes that we do not (for example, they show that reachability analysis for ARBAC with no explicit negative preconditions is PSPACE-complete), we consider numerous problem classes not considered in [JLT+08]. Specifically, we consider restrictions on $|SMER(r)|$, $|pre|$, $|ppre|$, and $|goal|$, as well as the Polynomial-size Plan (EPP) and Bounded Reachability (BRP) problems, while they do not. This implies that they do not prove the results in our Theorems 5, 6, 11, 12, 18, 19, 22, 27, 28, and 29. In addition, we consider reachability analysis without the separate administration restriction, availability analysis, containment analysis, and information flow analysis, while they do not.

Stoller *et al.* [SYRG07] present two algorithms for reachability analysis of a variant of URA97, and use parameterized complexity theory to show that each algorithm is efficient for policies that may be large overall but are small with respect to certain metrics. In contrast, in this paper, we consider a much wider range of problem classes defined by various restrictions rather than metrics. Also, we consider availability analysis and information flow analysis, while [SYRG07] does not.

Sistla and Zhou [SZ08], like [LMW05], consider trust management policies changing in accordance with *role restrictions* that indicate, for each role, whether arbitrary rules defining that role may be added, and whether they may be removed. The administrative policies we consider are finer-grained than such role restrictions.

Munawer and Sandhu [MS99] shows that Augmented Typed Access Matrix Model (ATAM) can be simulated by appropriate configuration of RBAC96 [SCFY96] components. The undecidability of reachability analysis of ATAM suggests that reachability analysis for RBAC may be undecidable when a complicated administrative model (similar in expressiveness to ATAM) is used.

Crampton shows undecidability of the safety problem—specifically, user-role reachability analysis—

for RBAC96/ARBAC97 [Cra02, Chapter 5]. However, Crampton's definition of reachability is different than ours. Our definition is based closely on ARBAC: the transitions are exactly those permitted by a given ARBAC policy. In contrast, Crampton's definition of reachability is based on [HRU76]: the transitions are exactly those obtained by executing a given set of commands, where a *command* consists of a pre-condition and a sequence of administrative operations. The commands used in Crampton's undecidability proof are not expressible as ARBAC97 policies for several reasons: some of the preconditions of commands are not expressible in the form allowed in URA and PRA policies in ARBAC97, some commands use administrative operations that change the ARBAC97 policy (specifically, the *can_assign* relation), and the commands define sequences of several administrative operations that must be performed atomically. Therefore, Crampton's undecidability result does not apply to reachability analysis as defined in this paper.

This paper extends our workshop paper [SYSR06] in several ways. It provides detailed proofs of the results in [SYSR06]. It adds results for reachability analysis when changes to role hierarchy are allowed and when the separate administration restriction is removed (Section 7). It also adds results for several other analysis problems, namely, permission-role and user-permission reachability for goals containing multiple permissions, role-role containment, permission-role containment, and information flow analysis (Section 6).

9 Conclusion

We considered the problem of analyzing the consequences of sequences of changes to RBAC policies that are allowed by miniARBAC policies. We found that the general analysis problem is intractable, and remains so even when a number of fairly strong syntactic restrictions are imposed on the ARBAC policies. For example, safety (reachability) analysis remains NP-hard even when revocation of roles is not allowed. It also remains NP-hard even when each role assignment has at most one precondition. We identified a few combinations of syntactic restrictions under which safety analysis can be done in polynomial time. More experience is needed to determine how often these restrictions are satisfied in practice. We expect that the restrictions $\overline{CR}$ (all roles can be unconditionally revoked) and $\overline{EN}$ (negation is used only for specifying mutual exclusion of roles, *i.e.*, separation of duty) are satisfied reasonably often in practice. Other restrictions, such as the absence of disjunction and restrictions on the number of preconditions, may be harder to satisfy in practice. We also expect that in many cases, when one of these restrictions is violated, the policy mostly satisfies the restriction, for example, only a few role assignment rules have more than one precondition.

This work is a step towards a deeper understanding of policy analysis for ARBAC. An important direction for future work is to study the effect of more global properties of the policy (as opposed to syntactic restrictions), for instance, to determine whether the analysis problem becomes tractable when dependencies between roles are acyclic. Another interesting direction for future work is to extend our results to trust management policies [BFL96, LM03].

References

[AH07] Gail-Joon Ahn and Hongxin Hu. Towards realizing a formal rbac model in real systems. In *Proceedings of the 12th ACM Symposium on Access Control Models And Technologies (SACMAT)*, pages 215–224, June 2007.

[Ame04] American National Standards Institute (ANSI), International Committee for Information Technology Standards (INCITS). Role-based access control. ANSI INCITS Standard 359-2004, February 2004.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proc. 1996 IEEE Symposium on Security and Privacy*, pages 164–173, May 1996.

[BK91] Christer Bäckström and Inger Klein. Parallel non-binary planning in polynomial time. In *Proc. IJCAI '91*, pages 268–273, 1991.

[BLR03] Arosha K. Bandara, Emil C. Lupu, and Alessandra Russo. Using event calculus to formalise policy specification and analysis. In *Proc. 4th IEEE Workshop on Policies for Distributed Systems and Networks*, 2003.

[BN95] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–656, 1995.

[Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[CC97] Laurence Cholvy and Frédéric Cuppens. Analysing consistency of security policies. In *Proc. IEEE Symposium on Security and Privacy*, 1997.

[Cra02] J. Crampton. Authorizations and antichains. Technical report, Ph.D. thesis, Birbeck College, University of London, 2002.

[CW87] David D. Clark and David R. Wilson. A comparison of commercial and military security policies. In *Proc. 1987 IEEE Symposium on Security and Privacy*, pages 184–194, 1987.

[FKMT05] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proc. 22nd International Conference on Software Engineering (ICSE)*, pages 196–205, 2005.

[GHRS05] Joshua D. Guttman, Amy L. Herzog, John D. Ramsdell, and Clement W. Skorupka. Verifying information flow goals in Security-Enhanced Linux. *Journal of Computer Security*, 13(1):115–134, 2005.

[GRS04] Dimitar P. Guelev, Mark Ryan, and Pierre-Yves Schobbens. Model-checking access control policies. In *Proc. 7th Information Security Conference (ISC)*, pages 219–230, 2004.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[JEZ03] Trent Jaeger, Antony Edwards, and Xiaolan Zhang. Policy management using access control spaces. In *ACM Transactions on Information Systems Security*, August 2003.

[JLT+08] Somesh Jha, Ninghui Li, Mahesh Tripunitara, Qihua Wang, and William Winsborough. Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 5(2), Apr–Jun 2008.

[JR04] Somesh Jha and Tom Reps. Model-checking SPKI-SDSI. *Journal of Computer Security*, 12:317–353, 2004.

[JSS97] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proc. 1997 IEEE Symposium on Security and Privacy*, pages 31–42, 1997.

[JSS00] Daniel Jackson, Ian Schechter, and Ilya Shlyakhter. Alcoa: the alloy constraint analyzer. In *Proc. 22nd International Conference on Software Engineering (ICSE)*, pages 730–733, 2000.

[JSZ03] Trent Jaeger, Reiner Sailer, and Xiaolan Zhang. Analyzing integrity protection in the SELinux example policy. In *Proc. USENIX Security Symposium*, August 2003.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum, 1972.

[LM03] Ninghui Li and John C. Mitchell. RT: A role-based trust-management framework. In *Proc. Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–212. IEEE Computer Society Press, 2003.

[LMW05] Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM*, 2005. To appear.

[LS77] R. J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *J. ACM*, 24(3):455–464, 1977.

[LT06] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 9(4):391–420, 2006.

[LTB07] Ninghui Li, Mahesh V. Tripunitara, and Ziad Bizri. On mutually-exclusive roles and separation of duty. *ACM Transactions on Information and Systems Security (TISSEC)*, 10(2):42–51, May 2007.

[MS99] Q. Munawer and R. S. Sandhu. Simulation of the augmented typed access matrix model (ATAM) using roles. In *International Conference on Informaton and Security*, 1999.

[Osb02] Sylvia L. Osborn. Information flow analysis of an rbac system. In *Proc. 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 163 – 168, 2002.

[San88] Ravi Sandhu. The schematic protection model: its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, 1988.

[San92] Ravi Sandhu. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–136, 1992.

[SBM99] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.

[SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SM02] Andreas Schaad and Jonathan D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *Proc. 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 13–22, 2002.

[SMJ01] Andreas Schaad, Jonathan Moffett, and Jeremy Jacob. The role-based access control system of a European bank: A case study and discussion. In *Proc. 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 3–9, 2001.

[SYRG07] Scott Stoller, Ping Yang, C. R. Ramakrishnan, and Mikhail Gofman. Efficient policy analysis for administrative role based access control. In *ACM Conference on Computer and Communication Security (CCS)*, Alexandria, Virginia, Oct 2007. ACM Press.

[SYSR06] Amit Sasturkar, Ping Yang, Scott D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. In *19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 124–138. IEEE Computer Society Press, 2006.

[SZ08] A. Prasad Sistla and Min Zhou. Analysis of dynamic policies. *Information & Computation*, 206(2–4):185–212, 2008.