Policy Analysis for Administrative Role Based Access Control*

Amit Sasturkar Ping Yang Scott D. Stoller C.R. Ramakrishnan
Department of Computer Science, Stony Brook University, Stony Brook, NY, 11794, USA
E-mail: {amits,pyang,stoller,cram}@cs.sunysb.edu

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) is a model for expressing 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, in particular to determine reachability properties (e.g., whether a user can eventually be assigned to a role by a group of administrators) and availability properties (e.g., whether a user cannot be removed from a role by a group of administrators) implied by a policy. We first establish the connection between security policy analysis and planning in Artificial Intelligence. Based partly on this connection, 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.

1. Introduction

Background. Role-Based Access Control (RBAC) [27] 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 role-permission 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.

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 Role-Based Access Control (ARBAC) [26] 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. For instance, a *can_assign* rule can be used to specify that a department manager may appoint as a TA only users who are already students. 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

*This work was supported in part by NSF under Grants CCR-0205376 and CCR-0311512, and by ONR under grant N00014-04-1-0722.

by other administrators. The ARBAC policy should be designed to prevent such unexpected interactions. In large organizations with many roles (*e.g.*, [29] 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.*, [12, 23, 24, 25, 30, 17, 22, 21]. In a role-based policy framework, a natural analysis problem is to check potential role membership. The *reachability* (or *safety* [12]) 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* [21] 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* [21]: 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.

In this paper, we consider analysis of ARBAC policies. We focus on reachability and availability analysis, which are simpler than containment analysis but still difficult. For general ARBAC policies, even reachability and availability analyses are intractable (PSPACE-complete).

Contributions. We consider general ARBAC policies that (i) control administrative operations that change user-role as well as permission-role relationships; and (ii) allow all administrative operations to have positive preconditions and negative preconditions (e.g., not a student in the same course).

Our main results are obtained by considering the role reachability problem for ARBAC in terms of the *planning problem* in Artificial Intelligence (AI). Given a set of actions, a starting state, and a goal, the AI planning problem is to find a sequence of actions that leads from the starting state to the goal. For role reachability analysis, the RBAC policy is the state of the system, the ARBAC policy determines the allowed actions that can change the state, and the goal is to add the given user to the given role. To the best of our knowledge, this paper is the first to study the connection between security analysis and the well-studied area of planning in AI. A few of our results are corollaries of existing results in the literature on planning, but most of our results are new.

We show that reachability analysis (RE) for general ARBAC policies is PSPACE-complete. This 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 [19] (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 [2, 3].

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 explore the complexity of reachability analysis, the above variants of it (BRE and EPP), and availability analysis under combinations of these restrictions on policies. In many cases, the analysis problem still has high computational complexity. 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 ARBAC policies is PSPACE-complete;
- determine the computational complexity of reachability analysis, bounded reachability analysis, and availability analysis, and determine existence of polynomial-size plans, for several categories of ARBAC policies; and
- give algorithms for cases where the analysis problem is solvable in polynomial time.

Sections 2 and 3 formally define the policy frameworks and the analysis problems. Our algorithms and complex-

ity results for reachability and availability analysis appear in Section 4. Due to space limitations, we present only selected proof sketches in this paper. Detailed proofs are available in [28].

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 [1]. 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 [1].

Definition 1 *A $miniRBAC$ policy γ is a tuple $\langle U, R, P, UA, PA\rangle$ 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 $\gamma = \langle U, P, R, UA, PA\rangle$, 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 [1] extends the $miniRBAC$ model with *role hierarchies* that are a natural means for structuring roles to reflect an organization's lines of authority and responsibility.

Definition 2 *A $miniHRBAC$ policy γ_h is a tuple $\langle U, R, P, UA, PA, \succeq\rangle$ where*

- *U, R, P, UA and PA are as in $miniRBAC$.*
- *$\succeq \subseteq R \times R$ is a partial order on the set R of roles.*

$r_1 \succeq r_2$ 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 $\gamma_h = \langle U, P, R, UA, PA, \succeq\rangle$, 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

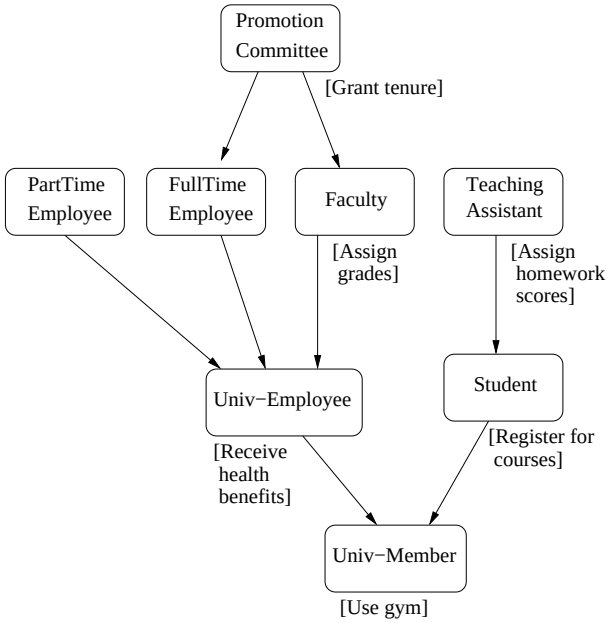

Figure 1. Example of $miniRBAC$ and $miniHRBAC$ policy.

`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 [26]. 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 $miniARBAC$. $miniARBAC$ specifies the URA and PRA policies, but does not specify a RRA policy; it does not allow any changes to the role hierarchy. Unlike ARBAC97, our model does not formally distinguish administrative roles from normal roles; this is a minor simplification that does not materially affect any of our results.

URA policy. The URA policy controls changes to the user-role assignment UA. Its specification uses *preconditions* (called prerequisite conditions in [26]) 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* $\subseteq R \times C \times R$ relation, where C is the set of all preconditions on R. 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 $\gamma = \langle U, P, R, UA, PA\rangle$ 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 $\gamma' = \langle U, P, R, UA \cup \{(u, r)\}, PA\rangle$. 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.

Permission to revoke users from roles is specified by the *can_revoke* $\subseteq R \times C \times R$ relation, where C is as above. [26] 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 $\gamma = \langle U, P, R, UA, PA\rangle$ 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 $\gamma' = \langle U, P, R, UA \setminus \{(u, r)\}, PA\rangle$.

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.

Static Mutually Exclusive Roles (SMER) constraints. $miniARBAC$ also includes a set of SMER constraints [19] which are used to enforce separation of duty [8]. A 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 vice versa. We choose to explicitly represent SMER constraints (and not specify them in the URA model using negation) because this allows us to develop specialized algorithms for analyzing policies that use negation only to enforce SMER constraints; this is a common case.

$miniARBAC$ **policy.** A $miniARBAC$ policy is represented as $\psi = \langle$*can_assign*, *can_revoke*, *can_assign_p*, *can_revoke_p*, $SMER\rangle$, where the five relations are as defined above. 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 $UserAssign$, $UserRevoke$, $PermAssign$, and $PermRevoke$ specified above, 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 $(r_a, \texttt{Student}, \texttt{TA}) \in$ *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 $(r_a,$ `Faculty` $\wedge$ `FullTime-Employee`, `Promotion-Committee`$) \in$ *can_assign*, where r_a is an appropriate administrative role.
- **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 $SMER = \{\{\texttt{Faculty}, \texttt{Student}\}\}$.
- **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 : $(r_a, \texttt{Student}, \texttt{TA}) \in$ *can_assign*, and $(r_a, \neg\texttt{TA}, \texttt{Student}) \in$ *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 [26], 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. Analysis of ARBAC policies

As mentioned in Section 1, policy analysis is useful for policy understanding and maintenance, and can also help in policy enforcement. 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)?
- **User-Role Availability Analysis [21]:** Given a role r and a member u of r, can u be removed from r ?
- **Permission-Role Availability Analysis [21]:** Given a role r and a permission p granted to r, can p be revoked from r?

4.1. User-Role Reachability Analysis without Role Hierarchy

A query Q of this kind 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 transform γ to another $miniRBAC$ policy γ' under the restrictions imposed by ψ such that u is a member of all roles in $goal$ in γ' ? We can simplify the problem as follows.

1. **Ignoring permissions:** The answer to Q is affected only by the user-role assignment relation and the *can_assign*, *can_revoke* and $SMER$ components of ψ, so we can ignore the other components of γ and ψ when answering Q. This also implies that only the $UserAssign$ and $UserRevoke$ actions are relevant.
2. **Implicit administrative role:** We can remove from ψ all administrative roles not in A and their corresponding *can_assign* and *can_revoke* rules. Then Q asks about reachability under all 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 $R \times C \times R$) where R is the set of roles and C is the set of all preconditions on R.
3. **Single user:** The preconditions for $UserAssign(a, u, r)$ and $UserRevoke(a, u, r)$ depend only on the current role memberships of user u. 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 $miniRBAC$ policy γ is a pair $\langle R, UA\rangle$ where $UA \subseteq R$, an action is $UserAssign(r)$ or $UserRevoke(r)$, and a $miniARBAC$ policy ψ is a triple $\langle$*can_assign*, *can_revoke*, $SMER\rangle$ where *can_assign*, *can_revoke* $\subseteq C \times R$. A reachability query for the simplified policy can be represented by a set $goal$ of roles (since u and A are now implicit). A goal set $goal$ is satisfied in

a RBAC policy state $\gamma = \langle R, UA \rangle$, denoted $\gamma \vdash goal$, iff $goal \subseteq UA$.

Definition 3 *A 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. 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 $\gamma' \vdash goal$.*

We consider the following problems related to reachability.

- *Reachability* **(RE):** Given a problem instance I, does there exist a plan for I ?
- *Bounded Reachability* **(BRE):** Given a problem instance I and an integer k, does there exist a plan for I of length at most k ? Existence of bounded plans is an interesting problem to consider in cases where existence of general plans is difficult to determine.
- *Existence of Polynomial-size Plan* **(EPP):** Given a set S of problem instances, is there 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 $f(|I|)$? The size $|I|$ of a problem instance I is the sum of the sizes of all the sets in it.

The *Reachability* problem is PSPACE-complete in general. To understand the problem better and identify efficiently solvable cases of practical interest, we impose various structural restrictions on the $miniARBAC$ policy and the query, and for each restricted class of problems, we analyze the complexity of RE and BRE and determine whether EPP holds.

We first define some auxiliary functions. Given a $miniRBAC$ policy $\gamma = \langle R, UA \rangle$, and a $miniARBAC$ policy $\psi = \langle$*can_assign*, *can_revoke*, $SMER\rangle$, define for each role $r \in R$

- *Num-SMER*$(r) = |\{r' : \{r, r'\} \in SMER\}|$. *Num-SMER*$(r)$ counts the number of SMER constraints that a role r is involved in.
- $Disjuncts(r) = |\{c : (c, r) \in$ *can_assign*$\}|$. $Disjuncts(r)$ counts the number of different rules in ψ that allow an administrator to assign a user to role r (we regard the preconditions in those rules as disjuncts in the policy's overall condition for assigning a user to role r). Similarly, $Disjuncts(not(r)) = |\{c : (c, r) \in$ *can_revoke*$|$.
- *Size*(c) for a precondition c is the number of literals in c. For example, $Size(r_1 \wedge \neg r_2) = 2$.
- *Size-Pos*(c) for a precondition c is the number of positive role literals in c. For example, *Size-Pos*$(r_1 \wedge r_2 \wedge \neg r_3) = 2$.

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 all roles r, $Disjuncts(r) \leq 1$ and $Disjuncts(not(r)) \leq 1$; in other words, there is at most one rule in ψ for assigning/revoking every role in 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. When considering powerful administrative roles, this restriction is reasonable because preconditions on revocation are relatively rare; recall that ARBAC97 does not support conditional revocation.
- **Size restrictions:** ψ satisfies $|pre| \leq k$ if $\forall (c, r) \in$ *can_revoke* : $Size(c) \leq k$ and $\forall (c, r) \in$ *can_assign* : $Size(c) +$ *Num-SMER*$(r) \leq k$ (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*: *Size-Pos*$(c) \leq k$. ψ satisfies $|SMER(r)| \leq k$ if $\forall r \in R$: *Num-SMER*$(r) \leq k$. ψ satisfies $|goal| \leq k$ if the size of the goal set is at most k. 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. When EPP is `false` for a problem instance in a class $\mathcal{C}$, we say that reachability analysis for $\mathcal{C}$ is *intractable*, since the worst case running-time of any algorithm that generates plans for $\mathcal{C}$ is more than polynomial in the size of the problem instance.

Figure 2 summarizes our results. The problem classes 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 $\mathcal{C}_2$. The bibliographic reference to [22] means that the result was proved there. A reference to [6] or [3] means that we proved complexity results for that problem class by reduction from complexity results for planning given in that reference. Some observations follow.

1. If *Existence of Polynomial-size Plan* (EPP) for a problem class $\mathcal{C}$ is `true`, then *Reachability* (RE) for $\mathcal{C}$ is in NP, because a non-deterministic Turing machine can guess the plan, and verify it in polynomial time.
2. 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 $\mathcal{C}$.
3. 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.
4. The restriction $\overline{CR}$ (only unconditional revocation) makes the *Existence of Polynomial-size Plan* (EPP) problem easier; 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`.
5. 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.
6. 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. 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.

4.2. Complexity Results for User-Role Reachability Analysis without Role Hierarchy

In this section we give proof sketches for selected representative results from Figure 2. Complete proofs of all the complexity results appear in [28]. Theorem 1 shows that solving the *Reachability* problem in the general case is PSPACE-complete. The proof of Theorem 2 provides a polynomial time algorithm for solving *Reachability* (RE) for a problem class that is still general enough to be interesting in practice. Theorems 5 and 6 show that even when three or four restrictions are applied simultaneously, reachability may remain a hard problem, not solvable in polynomial time.

Theorem 1 *Reachability (RE) for the problem class without any restrictions is PSPACE-complete.*

PROOF SKETCH: [3] shows that Plan-Existence for a SAS^+ planning problem 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—$UserAssign(r)$ and $UserRevoke(r)$—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. *Reachability* for unrestricted ARBAC policies is in PSPACE because a Turing Machine can guess and execute the plan, storing at each step only the current state whose size is polynomial in the size of the problem instance. Thus *Reachability* for unrestricted ARBAC policies is PSPACE-complete. □

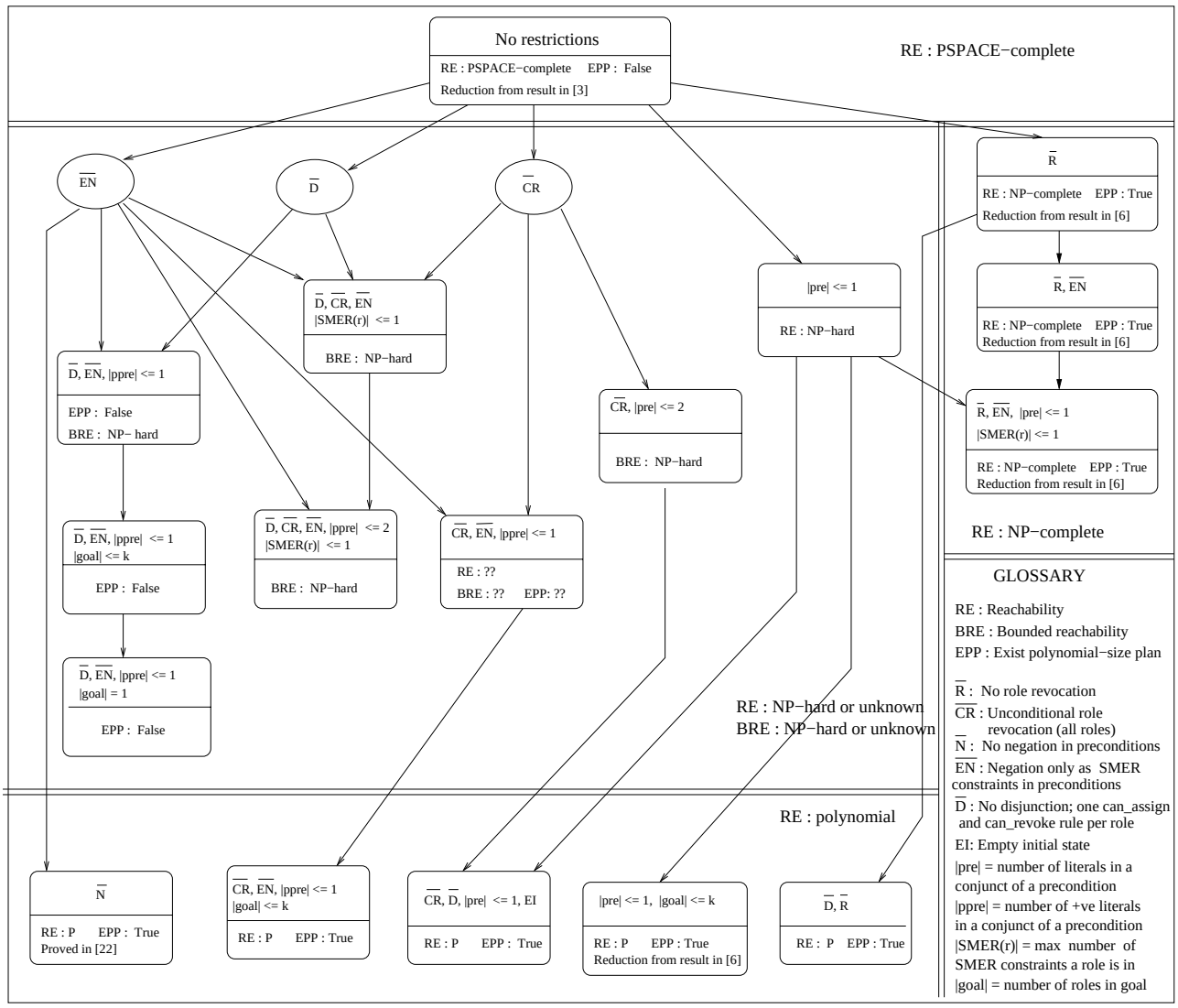

Figure 2. Complexity of Reachability Analysis

Theorem 2 *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 SKETCH: 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. One can show first that a cycle cannot contain both a `pos` edge and a `neg` edge and then that *Reachability* (RE) is `false` if and only if either (`C1`) G_ψ contains a `pos` cycle Y such that $Y \cap goal \neq \emptyset$, or (`C2`) G_ψ contains a `neg` cycle Y such that $Y \subseteq goal$, or (`C3`) $goal \not\subseteq G_\psi$. 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, either $UserAssign(r)$ occurs before $UserAssign(s)$ (if there is a path of `pos` edges from r to s) or $UserRevoke(s)$ occurs between $UserAssign(s)$ and $UserAssign(r)$. In the latter case, 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.

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. □

The problem class to which Theorem 2 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; the proofs are straightforward.

Lemma 3 *Suppose ψ satisfies $\overline{CR}$. RE for $(\langle R, UA\rangle, goal, \psi)$ is `true` if RE for $(\langle R, \emptyset\rangle, goal, \psi)$ is `true`.*

Lemma 4 *RE for $(\langle R, UA\rangle, goal, \psi)$ is `false` if RE for $(\langle R, \emptyset\rangle, UA, \psi)$ is `true` and RE for $(\langle R, \emptyset\rangle, goal, \psi)$ is `false`.*

Theorem 5 *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 pre-requisites) is NP-hard.*

PROOF SKETCH: The proof is by reduction from the `CLIQUE` problem which is known to be NP-complete [18]. Given a graph $G = (V, E)$ and an integer k, the `CLIQUE` problem asks whether G has a clique of size k, *i.e.*, a completely connected subgraph with k vertices. We construct a problem instance $I = (\gamma, goal, \psi)$ in the problem class $[\overline{D}, \overline{EN}, |ppre| \leq 1]$ such that G has a clique of size k if and only if I has a plan of size at most $15n - 2k$, where $n = |V|$.

The proof of Theorem 8 in [3] establishes NP-hardness of Bounded-Plan-Existence for a planning problem, which is equivalent to our *Bounded Reachability* problem for the problem class $\overline{D}$, by reduction from the `CLIQUE` problem. Our reduction and proof is similar to theirs in structure, but since our aim is to show NP-hardness of *Bounded Reachability* for a more restricted problem class $[\overline{D}, \overline{EN}, |ppre| \leq 1]$, our construction and proof is significantly more involved. □

Theorem 6 *Existence of Polynomial-size Plan (EPP) for the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |SMER(r)| \leq 1]$ (no disjunction, only unconditional role revocation, no explicit negation, at most one SMER constraint per role) is `false`.*

PROOF SKETCH: Consider the problem instance $I_n = (\gamma, goal, \psi)$ where the set of roles $R = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n\}$, $\gamma = \langle R, \emptyset\rangle$, $goal = \{u_n\}$, and $\psi = \langle$*can_assign*, *can_revoke*, $SMER\rangle$ where

- $SMER = \{\{u_i, v_i\} : 1 \leq i \leq n\}$
- $\forall\, 1 \leq i \leq n : (\texttt{true}, v_i) \in$ *can_assign*
- $\forall\, 1 \leq i \leq n : (\texttt{true}, v_i) \in$ *can_revoke*
- $\forall\, 1 \leq i \leq n : (\texttt{true}, u_i) \in$ *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.

It is easy to check that I_n is in the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |SMER(r)| \leq 1]$.

We claim there exists a plan for I_n, and a minimum plan for I_n has size exponential in n. Note that reachability of a role u_i depends only on roles u_j where $j < i$. Intuitively, in order to reach u_{2k} for some integer k, we must reach a state γ in which the goal set $\{u_1, \ldots, u_{2k-1}\}$ is satisfied. u_{2k-1} can only be reached from a state γ' in which the goal set $\{v_1, v_2, \ldots, v_{2k-3}, u_{2k-2}\}$ is satisfied. Since for all i, $\{v_i, u_i\} \in SMER$, it follows that $u_1, u_2, \ldots, u_{2k-3}$ are all `false` in γ'. Therefore, to go from γ' to γ, the roles $v_1, v_2, \ldots, v_{2k-3}$ must first be revoked, and then the goal set $g = \{u_1, u_2, \ldots, u_{2k-3}\}$ must be proved. But, to prove goal u_{2k-2} (while reaching state γ') starting from the initial empty state γ'', the same goal set g must be proved. Therefore, the length of the plan $\gamma' \rightarrow^* \gamma$ is greater than the length of $\gamma'' \rightarrow^* \gamma'$, implying that the length of $\gamma'' \rightarrow^* \gamma$ is at least twice the length of $\gamma'' \rightarrow^* \gamma'$. Thus, the length of a minimum plan to reach u_{2k} is at least twice the length of a minimum plan to reach u_{2k-2}. It follows that the length of

a minimum plan to reach u_n is exponential in n. Since $|I_n|$ is $O(n^2)$, the length of a minimum plan to reach u_n is not polynomial in $|I_n|$. □

4.3. User-Role Reachability Analysis in Hierarchical RBAC

Recall that $miniARBAC$ does not consider a RRA policy, *i.e.*, $miniARBAC$ does not allow changes to the role hierarchy. This allows us to transform analysis problems for hierarchical policies into 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.

Let $I_h = (\gamma_h, goal_h, \psi_h)$ be a reachability problem instance for hierarchical RBAC with $\gamma_h = \langle R, UA, \succeq \rangle$, $\psi_h = \langle \textit{can\_assign}_h, \textit{can\_revoke}_h, SMER_h \rangle$, and $goal_h = \{r_1, r_2, \ldots, r_k\}$. Define a set of reachability problem instances for non-hierarchical RBAC as follows.

- Let $\gamma = \langle R, UA \rangle$.
- The *can_assign* and *can_revoke* relations are generated in two steps from $\textit{can\_assign}_h$ and $\textit{can\_revoke}_h$.
  1. For each $(c, r) \in \textit{can\_assign}_h$, and for each $\neg t \in c$, replace $\neg t$ with $\bigwedge_{s \in Senior(t)} \neg s$. Transform the $\textit{can\_revoke}_h$ 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 $(r'_1, \ldots, r'_k) \in PosConjunct$ 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 $SMER = \{(r, s) : (r', s') \in SMER_h \wedge r \succeq r' \wedge s \succeq s'\}$.
- $Goals = Senior(g_1) \times Senior(g_2) \times \ldots \times Senior(g_n)$.

Then, 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 in Section 4.1 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 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. In particular, 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. It might be possible to find an algorithm whose running time is exponential only in the number of negative preconditions in the policy; this is a topic for future work.

4.4. Other Analysis Problems

Permission-Role Reachability Analysis Consider queries of the form "Can administrators in administrative roles in A assign a permission p to all roles in $goal$?".

Since $miniRBAC$ and $miniARBAC$ specifications for the user-role and permission-role assignment relations are symmetrical, permission-role reachability analysis can be performed in exactly the same manner as user-role reachability with $SMER = \emptyset$. Thus, the results of Section 4.2 apply directly.

User-Permission Reachability Analysis Consider queries of the form "Can administrators in administrative roles in A give user u permission p?". Such a query can be answered by checking whether there exists a role r such that (1) user u is already 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. Furthermore, a plan for the original problem can be obtained by simply concatenating the plans for the two sub-problems (*i.e.*, a plan for adding user u to r, and a plan for granting permission p to r). These observations imply that the results in Section 4.2 can easily be used to obtain algorithms and complexity results for the *Reachability*, *Bounded Reachability* and *Existence of Polynomial-Size Plan* problems for user-permission problem classes.

Availability Analysis User-Role Availability analysis checks whether a given member of a given role always remains in the role. As for user-role reachability analysis, we simplify the problem by ignoring permissions and the permission-role assignment, and assuming a single implicit user and a single implicit administrative role. Formally, a user-role availability analysis problem instance has the form $I = (\gamma, goal, \psi)$ where $\gamma = \langle R, UA\rangle$ is a simplified $miniRBAC$ policy, $\psi = \langle$*can_assign*, *can_revoke*, $SMER\rangle$ 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 $\gamma' = \langle R', UA\rangle$ where $R' = R \cup goal'$.
   - $\psi' = \langle$*can_assign'*, *can_revoke'*, $SMER'\rangle$ where (1) $\forall \bar{r} \in goal' : (\texttt{true}, \bar{r}) \in$ *can_assign'*, (2) $\forall \bar{r} \in goal' : (\texttt{true}, \bar{r}) \in$ *can_revoke'*, 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 $\gamma' = \langle R, UA'\rangle$ 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, and we can apply the complexity results and algorithms in Section 4.2.

5 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 policy. We mention some representative papers in this category. Jajodia, Samarati, and Subrahmanian [16] 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 [7] use SOL-deduction to check consistency of a security policy that expresses positive and negative permissions and obligations. Bandara, Lupu, and Russo [4] 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.* [14, 15] give algorithms to check integrity and completeness of a Security-Enhanced Linux (SELinux) policy. Guttman *et al.* [11] 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 [17] 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.* [9] 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 [12] 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.*, [23, 24, 25]. While each of these papers proposes a specific model designed with tractable analysis in mind, we start with the ARBAC97 model [26] 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 [10] 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.

Li and Tripunitara [22] 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 [21]. Our work goes significantly beyond theirs by considering negative preconditions and SMER (static mutually exclusive roles) constraints. They do not consider these features. Indeed, they write: "Many other more sophisticated cases of security analysis in RBAC remain open. For example, it is not clear how to deal with negative preconditions in role assignment, and how to deal with constraints such as mutually exclusive roles" [22]. 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.

Schaad and Moffett [30] express RBAC and ARBAC97 in Alloy, a relational modeling language, and use the Alloy analyzer [13] 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.

6. Conclusion

We considered the problem of analyzing the consequences of sequences of changes to RBAC policies that are allowed by ARBAC 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 develop analysis algorithms that perform well for policies that mostly satisfy combinations of the syntactic restrictions. The complexity of such algorithms would be polynomial in policy size parameters expected to be large in practice and exponential in parameters expected to be relatively small, *e.g.*, the number of roles that are involved in mutual exclusion constraints and have more than one positive precondition that constrains their assignment.

Another 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 apply to containment analysis [22] and trust management policies [5, 20].

References

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

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

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

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

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. 1996 IEEE Symposium on Security and Privacy*, pages 164–173, May 1996.

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

[7] L. Cholvy and F. Cuppens. Analysing consistency of security policies. In *Proc. IEEE Symposium on Security and Privacy*, 1997.

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

[9] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proc. 22nd International Conference on Software Engineering (ICSE)*, pages 196–205, 2005.

[10] D. P. Guelev, M. Ryan, and P.-Y. Schobbens. Model-checking access control policies. In *Proc. 7th Information Security Conference (ISC)*, pages 219–230, 2004.

[11] J. D. Guttman, A. L. Herzog, and J. D. Ramsdell. Information flow in operating systems: Eager formal methods. In *Proc. 2003 Workshop on Issues in the Theory of Security (WITS)*, 2003.

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

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

[14] T. Jaeger, A. Edwards, and X. Zhang. Policy management using access control spaces. In *ACM Transactions on Information Systems Security*, Aug. 2003.

[15] T. Jaeger, R. Sailer, and X. Zhang. Analyzing integrity protection in the SELinux example policy. In *Proc. USENIX Security Symposium*, Aug. 2003.

[16] 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.

[17] S. Jha and T. Reps. Model-checking SPKI-SDSI. *Journal of Computer Security*, 12:317–353, 2004.

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

[19] N. Li, Z. Bizri, and M. V. Tripunitara. On mutually-exclusive roles and separation of duty. In *In Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 42–51, Oct. 2004.

[20] N. Li and J. 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.

[21] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM*, 2005. To appear.

[22] N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *Proc. 9th ACM Symposium on Access Control Models and Techniques (SACMAT)*, June 2004.

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

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

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

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

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

[28] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role-based access control. Technical report, Stony Brook University, 2006. Available from `http://www.cs.sunysb.edu/~stoller/arbac.html`.

[29] A. Schaad, J. Moffett, and J. 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.

[30] A. Schaad and J. 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.