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 (e.g., whether a user can eventually be assigned to a role by a group of administrators) and availability (e.g., whether a user cannot be removed from a role by a group of administrators) properties 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.**

I. 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 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) [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 an user, and under what conditions, and *can_revoke* rules that specify the roles from which an administrator may remove an 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 an user to the TA role is governed by both a positive pre-condition (1), and a negative pre-condition (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], [LT04], [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]: 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 pre-requisite conditions and negative pre-requisite conditions (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): 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 pre-requisite conditions.
2. $\overline{EN}$: negative pre-requisite conditions for role assignment are used only in the form of static mutually-exclusive role (SMER) constraints [LBT04], 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 pre-requisite conditions for role assignment.

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 (PP) 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 PP), 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 II and III formally define the policy frameworks and the analysis problems. Our algorithms and complexity results for reachability analysis appear in Section IV. In Section V, we extend those results to consider role hierarchy, and we consider other analysis problems, including availability analysis. Due to space limitations and the technical nature of the proofs, we present only selected proof sketches in the main paper. For the convenience of the reviewer, complete proofs have been attached as an appendix.

II. 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. RBAC facilitates specification and management of security policies in large systems. In this paper, we study policy analysis only for models of RBAC based on [FSG+01]. 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 [FSG+01].

Definition 1: A *miniRBAC* policy $\gamma = \langle U, R, P, UA, PA \rangle$ where

- U, R and P are sets of users, roles and permissions respectively. A permission represents approval 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 [FSG+01] 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.

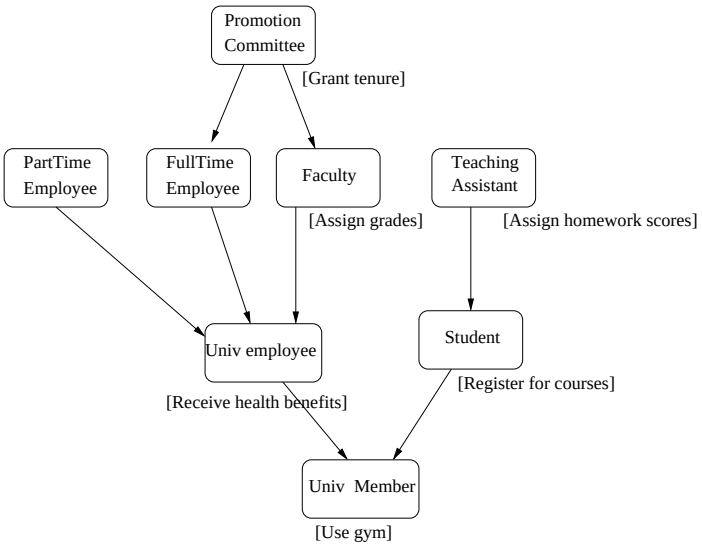

Fig. 1. Example of *miniRBAC* and *miniHRBAC* policy

Definition 2: A *miniHRBAC* policy $\gamma_h = \langle U, R, P, UA, PA, RH \rangle$ where

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

$r_1 \succeq_{RH} r_2$ 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 $\gamma_h = \langle U, P, R, UA, PA, RH \rangle$, 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 $Senior(r) = \{r' \in R : r' \succeq_{RH} r\}$.

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.

III. 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]. In ARBAC97, administrator roles are separate from normal roles, and changes to the ARBAC policy (*e.g.*, adding users 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 only make changes that are consistent with the constraints.

There are three main parts in a ARBAC97 policy : the user-role administration (URA) policy, the permission-role administration (PRA) policy, and the role-role administration (RRA) policy that control changes to user-role assignment UA, the permission-role assignment PA, and the role hierarchy respectively. In this paper, we consider slightly modified and simplified 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.

URA Policy. The URA policy controls changes to the user-role assignment UA. Its specification uses *prerequisite conditions* 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 pre-requisite condition $\wedge_i l_i$, 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 pre-requisite conditions on R. A $UserAssign(r_a, u, r)$ action specifies that an administrator who is a member of the administrative role r_a adds a user u to a 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 pre-requisite conditions 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 u is removed 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 the set of all pre-requisite conditions on R. [SBM99] mentions the option of including pre-requisite conditions 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 a user u from the membership of a 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 [LBT04] which are used to enforce separation of duty [CW87]. A SMER constraint is an unordered pair of roles $s = \{r_1, r_2\}$ and is satisfied in the state γ, denoted by $\gamma \vdash s$, 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 pre-requisite condition 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 all the five relations are as defined above. A $miniARBAC$ policy specifies a transition relation between $miniRBAC$ policies. We denote a transition by $\gamma \xrightarrow{act}_\psi \gamma'$ where act is one of the administrative actions $UserAssign$, $UserRevoke$, $PermAssign$, and $PermRevoke$ with semantics as specified above, and γ and γ' satisfy the SMER constraints in ψ.

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

- **Positive pre-requisites :** A user can be made member of the `Teaching-Assistant` 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 $(r_a, \texttt{Student}, \texttt{Teaching-Assistant}) \in can\_assign$.
- **Conjunction in pre-requisite conditions:** 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 $(r_a, \texttt{Faculty} \wedge \texttt{FullTime-Employee}, \texttt{Promotion-committee}) \in can\_assign$, 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 pre-requisites :** Negative pre-requisites 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 `Teaching-Assistant` 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 `Teaching-Assistant`. This policy can be enforced with the following rules : $(r_a, \texttt{Student}, \texttt{Teaching-Assistant}) \in can\_assign$, and $(r_a, \neg\texttt{Teaching-Assistant}, \texttt{Student}) \in can\_revoke$. The second rule forces an administrator to revoke the user's `Teaching-Assistant` role before revoking his `Student` role.
- **Conditional role revocation:** Recall that $miniARBAC$, unlike ARBAC97 [SBM99], allows pre-requisite conditions in role revocation. The policy with negative pre-requisites described above is also an example of a policy that requires conditional role revocation.

IV. Analysis of RBAC policies

As mentioned in Section I, 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 [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 ?

In the next section, we consider the first kind of query without role hierarchy. In Section V we consider the other kinds of queries.

A. User-Role Reachability Analysis without Role Hierarchy

A query Q of this kind has the form: Given a user u, 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, and 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 pre-requisite conditions on R.
3. **Single user:** Note that the pre-requisite conditions 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 $I^{ur} = (\gamma, goal, \psi)$ where γ is a *miniRBAC* policy, ψ is a *miniARBAC* policy and $goal \subseteq R$ is a query. 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^{ur} if $\gamma \xrightarrow{act_1}_{\psi} \ldots \xrightarrow{act_n}_{\psi} \gamma'$ and $\gamma' \vdash goal$.

We consider two variants of reachability analysis.

- *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.
- *Polynomial-size plan* **(PP) :** 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 PP 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* counts the number of SMER constraints that a role r is involved in.
- $Disjuncts(r) = |\{c : (c, r) \in can\_assign\}|$; *i.e.*, $Disjuncts(r)$ counts the number of different rules in ψ that allow an administrator to assign a user to the role r. Similarly, $Disjuncts(not(r)) = |\{c : (c, r) \in can\_revoke|$.
- *Size*(c) for a pre-requisite condition c is the number of role literals in c; *e.g.*, $Size(r_1 \wedge \neg r_2) = 2$.
- *Size-Pos*(c) for a pre-requisite condition c is the number of positive role literals in c; *e.g.*, *Size-Pos*$(r_1 \wedge r_2 \wedge \neg r_3) = 2$.

We consider four categories of restrictions on ψ.

- **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 either 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 $Disjuncts(not(r)) \leq 1$; *i.e.*, there is at most one rule in ψ for assigning/revoking every role in R.
- **Restricting Revocation:**
  - **No revocation ($\overline{R}$) :** ψ satisfies the 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 pre-requisite conditions 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 $\forall (c, r) \in can\_assign : Size(c) + Num\text{-}SMER(r) \leq k$ (if $\{r_1, r_2\}$ is a SMER constraint, then $\neg r_1$ is counted as part of every pre-requisite condition for r_2 in can_assign, and vice versa). ψ satisfies $|ppre| \leq k$ if $\forall (c, r) \in can\_assign \cup can\_revoke : Size\text{-}Pos(c) \leq k$. ψ satisfies $|SMER(r)| \leq k$ if $\forall r \in R : Num\text{-}SMER(r) \leq k$. 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 pre-requisite conditions contains only positive literals. For each class we consider the RE and BRE problems and check whether PP is `true` for every problem instance in the class. When PP is `false` for a problem instance in a class $\mathcal{C}$, reachability analysis is said to be *intractable* for $\mathcal{C}$ since the worst case running-time of any algorithm that generates plans for $\mathcal{C}$ is at least exponential 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 [LT04] means that the result was proved there. A reference to [Byl94] or [BN95] means that we proved complexity results for that problem class by reduction from complexity results for planning given in that reference. Some observations are

1. If *Polynomial-size plan* (PP) for a problem class $\mathcal{C}$, is `true`, then *Reachability* (RE) for $\mathcal{C}$ is in the complexity class `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 re-write 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 polynomial-time solvable. Not allowing disjunction in pre-requisite conditions 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 *Polynomial-size plan* problem easier; PP for $[\overline{D}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$ and hence for $[\overline{EN}, |ppre| \leq 1, |G| \leq k]$ is `false`, implying that a polynomial time algorithm for generating a plan for this problem class does not exist. PP for $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$ is `true`. In Appendix, we give a polynomial-time plan generation algorithm (Algorithm 3) for this problem class.
5. The restriction $\overline{R}$ (no revocation) ensures that the answer to the *Polynomial-size plan* problem is `true`. When role revocation is not allowed, the implicit 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 PP 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 $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$, but the worse-case complexity of RE for the class $[\overline{CR}, |ppre| \leq 1, |G| \leq k]$ is unknown.

B. Complexity Results for User-Role Reachability Analysis

In this section we give proof sketches for three representative results from Figure 2. Detailed proofs of all the complexity results appear in the Appendix. The first theorem shows that solving the *Reachability* problem in the general case is `PSPACE-complete`. The second theorem and proof provide a polynomial time algorithm for solving *Reachability* (RE) for a problem class that is still general enough to be interesting in practice. The fourth and fifth theorems show that even when three or four restrictions are applied simultaneously, reachability may remain a hard problem, not solvable in polynomial time.

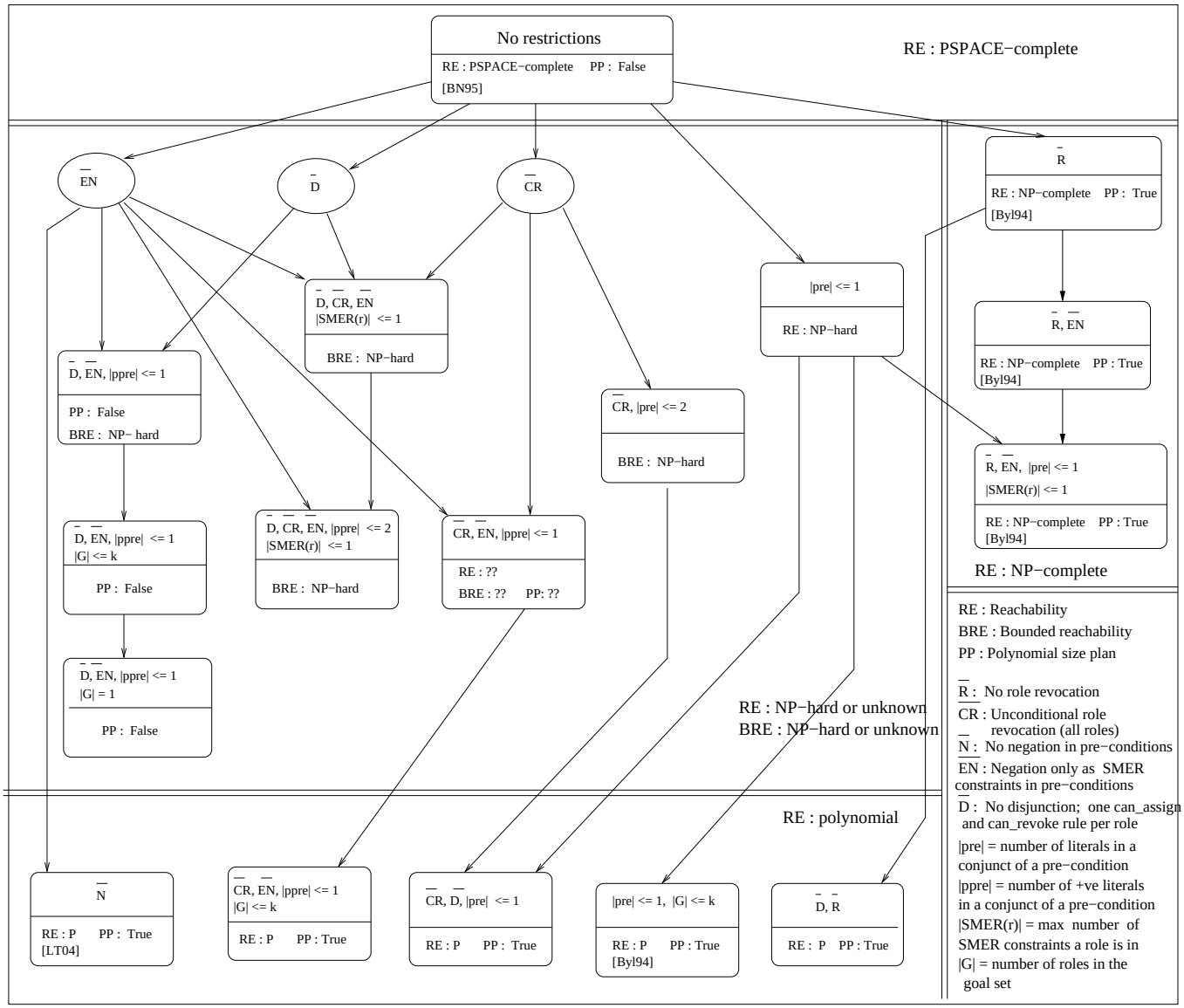

Fig. 2. Complexity of Reachability Analysis

Theorem 1: Reachability (RE) for the problem class without any restrictions is `PSPACE-complete` (Theorem 24 in Appendix).

PROOF SKETCH: [BN95] 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 in policy analysis - $UserAssign(r)$ and $UserRevoke(r)$ - are binary actions that also have a single effect, since they either add the implicit user to r or revoke the user from r. We can encode Plan-Existence for a SAS$^+$ planning problem that satisfies the U and B restrictions as a *Reachability* problem for a problem instance with an unrestricted $miniARBAC$ policy. This establishes that solving *Reachability* for general $miniARBAC$ 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* of unrestricted ARBAC policies is PSPACE-complete. ■

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 pre-requisite precondition, empty initial state) is solvable in polynomial time (Theorem 12 in Appendix).

PROOF SKETCH: Let $I = (\gamma, goal, \psi)$ be a problem instance in the problem class $[\overline{CR}, \overline{D}, |pre| \leq 1]$, where $\gamma = \langle R, UA \rangle$, $\psi = \langle can\_assign, can\_revoke, SMER \rangle$, and $goal$ is a set of roles. Then, for every role $r \in R$, (1) there is at most one state change rule $(r_a, c, r) \in can\_assign$, (2) $|c| \leq 1$, and (3) $(r_a, \texttt{true}, r) \in can\_revoke$. Also, since negative pre-requisite conditions are allowed, we can assume

without loss of generality that $SMER = \emptyset$.

Construct the graph $G_\psi = (V_\psi, E_\psi)$ as follows. The set of vertices is the set of roles $V_\psi = R$. There are two kinds of edges in E_ψ, positive and negative. For each $(r_a, r', r) \in can\_assign$, $e = (r', r) \in E_\psi$, and $label(e) =$ `pos`. For each $(r_a, \neg r', r) \in can\_assign$, $e = (r, r') \in E_\psi$ and $label(e) =$ `neg`. Note that `neg` edges have reverse direction as the `pos` edges. Intuitively, the 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 $UserAssign(r')$. A cycle is called a `pos` cycle if it is composed of only `pos` edges; `neg` cycles are defined similarly.

Lemma 3: Let $\mathcal{C}$ be a cycle in G_ψ. Then $\mathcal{C}$ is either a `pos` cycle or a `neg` cycle. (Lemma 13 in Appendix)

PROOF: Suppose $\mathcal{C}$ contains a `pos` edge $e = (r, r')$ and a `neg` edge $e = (s, s')$. Then, there is a path $P_1 = \langle e_1, e_2, \ldots, e_m \rangle$ in $\mathcal{C}$ such that $e_1 = (r', s_1)$, $\forall 2 \leq i \leq m-1 : e_i = (s_i, s_{i+1})$ and $e_m = (s_m, s)$. From the definition of `pos` and `neg` edges we have $(r_a, r, r') \in can\_assign$ and $(r_a, \neg s', s) \in can\_assign$. If $label(e_m) =$ `pos` then $(r_a, s_1, s) \in can\_assign$ contradicting that ψ satisfies the $\overline{D}$ restriction. Thus, $label(e_m) =$ `neg` and $(r_a, \neg s, s_1) \in can\_assign$. Continuing the argument, we can show that $e_1, \ldots, e_m$ are all `neg` labeled edges. Thus, $(r_a, \neg s_1, r') \in can\_assign$ and ψ violates the $\overline{D}$ restriction. Thus, $\mathcal{C}$ does not contain both `pos` and `neg` edges, and is either a `pos` cycle or a `neg` cycle. ■

Lemma 4: Reachability (RE) for $I = (\gamma, goal, \psi)$, where $\gamma = \langle R, \emptyset \rangle$, is `false` if and only if either (`C1`) G_ψ contains a `pos` cycle C such that $C \cap goal \neq \emptyset$, or (`C2`) G_ψ contains a `neg` cycle C such that $C \subseteq goal$. (Lemma 14 in Appendix)

PROOF SKETCH: Suppose `C1` is `true`. Let $r \in C \cap goal$. Since we start from the empty state γ, the action $UserAssign(r)$ can never be enabled; thus, r (and hence *goal*) is not reachable. Suppose `C2` is `true`; let $C = (r_1, r_2, \ldots, r_k, r_1)$ be a `neg` cycle such that $r_1, \ldots, r_k \in goal$. Let P be a plan for I, and assume without loss of generality that $UserAssign(r_k)$ is the last action in P. Then, since the pre-requisite condition for $UserAssign(r_k)$ is $\neg r_1$, it follows that $UserAssign(r_k)$ is not the last action in P which is a contradiction. Thus, *goal* is not reachable. If both `C1` and `C2` are `false`, then either (1) G is acyclic, in which case the topological-sort ordering of G gives the order in which roles must be assigned to reach *goal*, or (2) all cycles in G are `neg` cycles C such that there exists $s \in C$ and $s \notin goal$. We break such cycles by deleting every `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 (1) $UserAssign(r)$ occurs before $UserAssign(s)$ (if there is a path of `pos` edges from r to s) or (2) $UserRevoke(s)$ occurs between $UserAssign(s)$ and $UserAssign(r)$. In the latter case, we need to ensure that every $UserAssign(s')$ that has a pre-requisite condition s occurs before $UserRevoke(s)$ and hence before $UserAssign(r)$ in the plan. We add edge (s', r) to G to ensure this. 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 takes polynomial time, and $|G| = |I|$. Validity of `C1` can be checked by restricting G 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 if any cycle contains a role not in *goal*. Validity of condition `C2` can be checked by restricting G to only vertices in *goal* and `neg` edges, and checking whether the restricted graph contains a cycle; a simple Depth-First Search can accomplish this. Hence both `C1` and `C2` can be checked in polynomial time. Transforming G to an acyclic graph G' takes polynomial time, and $|G'|$ is $O(|G|^2)$ since for each `neg` edge in G, at most $|G|$ new edges may be added. Topologically sorting G' 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.

Lemma 5: RE for $I = (\langle R, UA \rangle, goal, \psi)$ is true if RE for $I_1 = (\langle R, \emptyset \rangle, goal, \psi)$ 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 $\langle R, \emptyset \rangle$. We then check if $\langle R, goal \rangle$ is reachable from $\langle R, \emptyset \rangle$. Thus, if a state $\langle R, UA_1 \rangle$ is reachable from the empty state, then it is reachable from any state. ■

Lemma 6: RE for $I = (\langle R, UA \rangle, goal, \psi)$ is `false` if $I_1 = (\langle R, \emptyset \rangle, UA, \psi)$ is `true` and $I_2 = (\langle R, \emptyset \rangle, goal, \psi)$ is `false`.

PROOF: Since I_1 is `true` there is a path P_1 from the empty state to $\langle R, UA \rangle$. Since I_2 is `false`, there is no path from the empty state to $\langle R, goal \rangle$. Thus, if there was a path from $\langle R, UA \rangle$ to $\langle R, goal \rangle$, I_2 would be `true` which is a contradiction. Thus, there is no path from $\langle R, UA \rangle$ to $\langle R, goal \rangle$ and I is `false`. ■

Theorem 7: 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`** *(Theorem 23 in Appendix).*

PROOF SKETCH: The proof is by reduction from the `CLIQUE` problem which is known to be `NP-complete` [Kar72]. 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 of size k. 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 $|V| = n$.

The proof of Theorem 8 in [BN95] 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 simi-

lar 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 8: Polynomial-size plan (PP) for the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |SMER(r)| \leq 1]$ (no disjunction, only unconditional role revocation, at most one SMER constraint per role, no explicit negation) is `false`. (Theorem 20 in Appendix)

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

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

The *can_revoke* relation specifies that for every role $r \in R$, $UserRevoke(r)$ has a `true` pre-requisite condition. Thus, I satisfies the $\overline{CR}$ restriction (every role can be unconditionally revoked). For every role $r \in R$, there is a unique state-change rule (r_a, c, r) in both *can_assign* and *can_revoke*. Thus I satisfies the $\overline{D}$ restriction. ψ does not use explicit negation; only SMER constraints. Thus I satisfies $\overline{EN}$ restriction. Each role u_i appears only with v_i in $SMER$. Thus I satisfies the $|SMER(r)| \leq 1$ restriction.

We claim that the minimum plan for I has size exponential in $|I|$. 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 the plan to reach u_{2k} is at least twice the length of the plan to reach u_{2k-2}; it follows that the length of the plan to reach u_n is exponential in n. Since $|I|$ is $O(n)$, the length of the plan to reach u_n is exponential in $|I|$. ■

V. OTHER ANALYSIS PROBLEMS

A. User-Role Reachability Analysis in Hierarchical RBAC

Our approach is 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 pre-requisite conditions.

Let $I_h = (\gamma_h, goal_h, \psi_h)$ be a reachability problem instance for hierarchical RBAC with $\gamma_h = \langle R, UA, RH \rangle$, $\psi_h = \langle can\_assign_h, can\_revoke_h, SMER_h \rangle$, 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 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 $(r'_1, \ldots, r'_k) \in PosConjunct$ 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 $SMER = \{(r, s) : (r', s') \in SMER_h \wedge r \succeq_{RH} r' \wedge s \succeq_{RH} s'\}$.
- $Goals = Senior(g_1) \times Senior(g_2) \times \ldots \times Senior(g_n)$.

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 IV 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 $|G| \leq k$ 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 pre-requisite conditions 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 pre-requisite conditions 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 $O(h^{|G|})$, where h is as in the previous paragraph. For problem classes with the restriction $|G| \leq k$, the number of transformed goals is polynomial in the size of the original policy.

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

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 in the number of negative pre-requisite conditions in the policy; this is a topic for future work.

B. Permission-Role Reachability Analysis

Consider queries of the form "Can administrators in administrative roles in A assign a permission p to all roles in a set *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 IV apply directly.

C. User-Permission Reachability Analysis

Consider queries of the form "Can administrators in administrative roles in A give a user u a 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 IV can easily be used to obtain algorithms and complexity results for the *Reachability*, *Bounded Reachability* and *Polynomial-size plan* problems for user-permission problem classes.

D. Availability Analysis

Recall from Section IV that User-Role Availability analysis checks whether a given member of a given role always remains in the role. As we did for user-role reachability analysis, we simplify *miniRBAC* and *miniARBAC* policies by ignoring permissions and the permission-role assignment, and assuming a single implicit user u and a single implicit administrative role r_a. 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.*, $\gamma \rightarrow_\psi * \gamma'$), user u is member of at least one role in *goal* in the 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 $I' = (\gamma', goal', \psi')$ 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' has opposite answers. Suppose I' has the answer `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 I' has the answer `false`, then I the answer `true`. Thus, availability analysis can be reduced to reachability analysis, and we can apply the complexity results and algorithms in Section IV.

VI. 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 [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.* [GHR03] 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 pre-requisite conditions, 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.

Li and Tripunitara [LT04] 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 pre-requisite conditions 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" [LT04]. Since we consider these features, we are driven to consider other restrictions, such as bounds on the size of pre-requisite conditions, that they do not consider.

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 pre-requisite conditions 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.

VII. 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 pre-requisite condition. 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 pre-requisite conditions, 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, when only a few role assignment rules have more than one pre-requisite condition.

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 pre-requisite condition that governs 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.

We also plan to extend our results to apply to containment analysis [LT04] and trust management policies [BFL96], [LM03].

References

[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 *IJCAI*, 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.

[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. IEEE Computer Society Press, 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.

[FSG+01] David F. Ferraiolo, Ravi Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[GHR03] Joshua D. Guttman, Amy L. Herzog, and John D. Ramsdell. Information flow in operating systems: Eager formal methods. In *Proc. 2003 Workshop on Issues in the Theory of Security (WITS)*, 2003.

[GRS04] Dimitar P. Guelev, Mark Ryan, and Pierre-Yves Schobbens. Model-checking access control policies. In *Proc. 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 219–230. Springer-Verlag, 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.

[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.

[LBT04] Ninghui Li, Ziad Bizri, and Mahesh V. Tripunitara. On mutually-exclusive roles and separation of duty. In *In Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 42–51, October 2004.

[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.

[LT04] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. In *Proc. 9th ACM Symposium on Access Control Models and Techniques (SACMAT)*, June 2004.

[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. ACM Press, 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)*. ACM Press, 2001.

Appendices

In this section, we shall present detailed proofs of the complexity results for User-Role reachability analysis without role hierarchy stated in Section IV.

A. Polynomial-time Reachability analysis

Lemma 9: Let $I = (\gamma, \psi, goal)$ be a reachability analysis problem instance where $\gamma = \langle R, UA\rangle$ and $\psi = \langle can\_assign, can\_revoke, SMER\rangle$. If $can\_revoke = \emptyset$ then if I has a plan, then I has a plan of length at most $|R|$.

Proof: Let $P = \langle a_1, a_2, \ldots, a_n\rangle$ be a plan for I. Since $can\_revoke = \emptyset$, for each $1 \leq i \leq n$, action $a_i = UserAssign(r_i)$ for some $r_i \in R$. Construct P' from P by eliminating all duplicate actions; $\forall 1 \leq i \leq n$, if $\nexists\, j \leq i : a_j = a_i$, then $a_i \in P'$, else $a_i \notin P'$. Then, $|P'| \leq |R|$. It is easy to see that P' is also a plan for I. Suppose $P = P_1.\langle UserAssign(r)\rangle.P_2.\langle UserAssign(r)\rangle.P_3$, Let $\gamma \xrightarrow{P_1.\langle UserAssign(r)\rangle} \gamma_1$, $\gamma_1 \xrightarrow{P_2} \gamma_2$, and $\gamma_2 \xrightarrow{\langle UserAssign(r)\rangle.P_3} \gamma_3$ where $\gamma_1 = \langle R, UA_1\rangle$, $\gamma_2 = \langle R, UA_2\rangle$ and $\gamma_3 = \langle R, UA_3\rangle$. Since P_2 does not contain any $UserRevoke$ action and $r \in UA_1$, we have $r \in UA_2$. Thus, $\gamma_2 \xrightarrow{P_3} \gamma_3$ and it follows that $P_1.\langle UserAssign(r)\rangle.P_2.P_3$ is a plan for I. Continuing in this manner, we can remove multiple occurrences of the same $UserAssign$ action from P (transforming P to P') while retaining the property that it is a plan for I. Thus, P' is a plan for I. ∎

Theorem 10: RE for the problem class $[\overline{N}]$ can be solved in polynomial time. In addition, PP for $[\overline{N}]$ is `true` [LT04].

Proof: Let $I = (\gamma, \psi, goal)$ be a reachability analysis problem instance in the problem class $[\overline{N}]$ where $\gamma = \langle R, UA\rangle$ and $\psi = \langle can\_assign, can\_revoke, SMER\rangle$. Then, $SMER = \emptyset$. Since the pre-requisite conditions in can_assign do not contain $\neg$, revoking a role (from the implicit user) does not satisfy any pre-requisite condition; we can assume $can\_revoke = \emptyset$ for reachability analysis. From Lemma 9 it follows that if I has a plan then I has a plan of length at most $|R|$. Thus, PP for the problem class $[\overline{N}]$ is `true`.

RE for $[\overline{N}]$ can be reduced to simple safety analysis for the Assignment and Trusted Users (AATU) class of security analysis problems [LT04]. Construct an AATU problem instance $I' = (\gamma, \psi', q, \exists)$ where $\psi' = \langle can\_assign, \emptyset\rangle$ (set of trusted users is $\emptyset$), and $q = goal \sqsupseteq \{u\}$ (the implicit user u is a member of every role in goal). It is easy to see that a plan for I is also a plan for I' and vice versa. Since q is a semi-static query, it follows from [LT04] that I' can be solved in time polynomial in $|I'|$. Since $|I| = |I'|$, RE for I can be solved in time polynomial in $|I|$. ∎

The proofs of Theorems 11, 16, and 18 are based on the complexity results for propositional STRIPS planning described in [Byl94]. Next, we describe the propositional STRIPS planning model of [Byl94].

Definition 4: An instance of *propositional STRIPS planning* is specified by a tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$ 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 pre-conditions o^+ and the negative pre-conditions o^- of the operator.
  - $Post$ is a satisfiable conjunction of positive and negative conditions, respectively called the positive post-conditions o_+ and the negative post-conditions o_- of the operator.
- $\mathcal{I} \in \mathcal{P}$ 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 $\mathcal{G}_-$.

The $PLANSAT$ problem is the decision problem for determining whether there exists a plan for a given a propositional STRIPS planning problem $\phi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$. $PLANSAT^{\alpha}_{\beta}$ is the decision problem for plan existence under the restriction that the pre-conditions of ϕ satisfy α and the post-conditions of ϕ satisfy β. For example, $PLANSAT^1_+$ indicates that all the post-conditions of ϕ are positive, and all pre-conditions have size at most 1.

Theorem 11: For the problem class $[|pre| \leq 1, |G| \leq k]$ RE can be solved in polynomial time and PP is `true`.

Proof: From Theorem 3.8 in [Byl94] we know that the $PLANSAT^1$ problem limited to constant number of goals is solvable in polynomial time. Given a problem instance I in the problem class $[|pre| \leq 1, |G| \leq k]$, we show that I can be translated to a $PLANSAT^1$ problem ϕ limited to k goals such that $|\phi| = O(|I|)$, thus proving that RE for I can be solved in polynomial time. The proof of Theorem 3.8 in [Byl94] also constructs a plan for I as it checks for plan existence. Thus, if a plan for I exists, it can be constructed in time polynomial in $|I|$ implying that the length of the plan is polynomial in $|I|$. Thus, PP for the problem class $[|pre| \leq 1, |G| \leq k]$ is `true`.

Let $I = (\gamma, goal, \psi)$ be in the problem class $[|pre| \leq 1, |G| \leq k]$ where $\gamma = \langle R, UA\rangle$ and $\psi = \langle can\_assign, can\_revoke, SMER\rangle$. Without loss of generality, we can assume $SMER = \emptyset$, since if $SMER \neq \emptyset$, then for each $(r_i, r_j) \in SMER$, add $\neg r_i$ to pre-requisite conditions for r_j in can_assign and can_revoke, and vice versa. Then, $|goal| \leq k$, and for each $(r_a, c, r) \in can\_assign \cup can\_revoke$, $|c| \leq 1$. Construct an instance $\phi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$ of the $PLANSAT^1$ problem as follows.

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

For every operator $o \in \mathcal{O}$, $pre(o) \leq 1$ since for each $(r_a, c, r) \in can\_assign \cup can\_revoke$, $|c| \leq 1$. 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. ■

Theorem 12: Reachability (RE) for the problem class $[\overline{CR}, \overline{D}, |pre| \leq 1, EI]$ (only unconditional role revocation, no disjunction, at most one pre-requisite precondition, empty initial state) is solvable in polynomial time.

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

Construct the graph $G_\psi = (V_\psi, E_\psi)$ as follows. The set of vertices is the set of roles $V_\psi = R$. There are two kinds of edges in E_ψ, positive and negative. For each $(r_a, r', r) \in can\_assign$, $e = (r', r) \in E_\psi$, and $label(e) = \texttt{pos}$. For each $(r_a, \neg r', r) \in can\_assign$, $e = (r, r') \in E_\psi$ and $label(e) = \texttt{neg}$. Note that `neg` edges have reverse direction as the `pos` edges. Intuitively, the 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 $UserAssign(r')$. A cycle is called a `pos` cycle if it is composed of only `pos` edges; `neg` cycles are defined similarly.

Lemma 13: Let $\mathcal{C}$ be a cycle in G_ψ. Then $\mathcal{C}$ is either a `pos` cycle or a `neg` cycle.

PROOF: Suppose $\mathcal{C}$ contains a `pos` edge $e = (r, r')$ and a `neg` edge $e = (s, s')$. Then, there is a path $P_1 = \langle e_1, e_2, \ldots, e_m \rangle$ in $\mathcal{C}$ such that $e_1 = (r', s_1)$, $\forall 2 \leq i \leq m-1 : e_i = (s_i, s_{i+1})$ and $e_m = (s_m, s)$. From the definition of `pos` and `neg` edges we have $(r_a, r, r') \in can\_assign$ and $(r_a, \neg s', s) \in can\_assign$. If $label(e_m) = \texttt{pos}$ then $(r_a, s_1, s) \in can\_assign$ contradicting that ψ satisfies the $\overline{D}$ restriction. Thus, $label(e_m) = \texttt{neg}$ and $(r_a, \neg s, s_1) \in can\_assign$. Continuing the argument, we can show that $e_1, \ldots, e_m$ are all `neg` labeled edges. Thus, $(r_a, \neg s_1, r') \in can\_assign$ and ψ violates the $\overline{D}$ restriction. Thus, $\mathcal{C}$ does not contain both `pos` and `neg` edges, and is either a `pos` cycle or a `neg` cycle. ■

The next lemma makes use of the hypothesis EI, by assuming γ has the form $\langle R, \emptyset \rangle$.

Lemma 14: RE for $I = (\gamma, goal, \psi)$ where $\gamma = \langle R, \emptyset \rangle$ is `false` if and only if either (`C1`) G_ψ contains a `pos` cycle $\mathcal{C}$ such that $\mathcal{C} \cap goal \neq \emptyset$, or (`C2`) G_ψ contains a `neg` cycle $\mathcal{C}$ such that $\mathcal{C} \subseteq goal$.

PROOF:

Case 1: We show that condition `C1` implies RE for I is `false`. Suppose `C1` is true; *i.e.*, G_ψ contains a `pos` cycle $\mathcal{C}$ and $r \in \mathcal{C} \cap goal$. 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 $\mathcal{C}$, including r itself. Thus, deriving role r entails deriving itself first, and hence no plan for deriving r exists. Thus RE for I is `false`.

Case 2: We show that condition `C2` implies RE for I is `false`. Suppose `C2` is true; *i.e.*, G_ψ contains a `neg` cycle and $\mathcal{C} \subseteq goal$ (every role in $\mathcal{C}$ is also in the goal set). Let $\mathcal{C} = (r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_k \rightarrow r_1)$. Then, $(r_a, \neg r_1, r_k) \in can\_assign$, and for $1 \leq i \leq k-1$, $(r_a, \neg r_{i+1}, r_i) \in can\_assign$. 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 pre-requisite condition 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: We show that if RE for I is `false`, then `C1` or `C2` is `true`. We prove the contrapositive, *i.e.*, we assume conditions `C1` and `C2` are false, and we show that RE for I is `true`, by giving an algorithm that constructs a plan for I. That algorithm consists of lines 4–27 of Algorithm 1 (that fragment of the overall algorithm corresponds to this case in the proof). The main idea of the algorithm is to construct a plan for I by topologically sorting G_ψ and assigning roles in the topologically sorted order. For this to work, we first need to break cycles in G_ψ, if any. From Lemma 3 and the hypothesis that conditions `C1` and `C2` are `false`, it follows that if $\mathcal{C}$ is a cycle in G_ψ, then $\mathcal{C}$ is a `neg` cycle, and there exists a role $r \in \mathcal{C}$ such that $r \notin goal$. To break the cycles in G_ψ, construct a graph G'_ψ from G by (1) deleting all cycles $\mathcal{C}'$ where $\mathcal{C}' \cap goal = \emptyset$ and (2) for each edge $e = (r, s)$ where $label(e) = \texttt{neg}$, $r \in goal$ and $s \notin goal$, delete e and (3) if there is no longer any path from r to s then add edges $(s_1, r), (s_2, r), \ldots, (s_m, r)$ where $(s, s_i) \in E_\psi$ (*i.e.*, add edges from every child of s to r). It is clear that G'_ψ is acyclic. We delete from G'_ψ all the components that do not contain any $goal$ role by defining $G''_\psi = comp(G'_\psi, goal)$. Intuitively, only the roles in G''_ψ are relevant for deriving the $goal$ roles. The graph $comp(G, S) = (S', E')$ given a graph G and a set of vertices S is defined as: set of vertices $S' = S \cup \{v \in V : \exists s \in S, \exists$ a path $v \rightarrow *s \in E$ or a path $s \rightarrow *v \in E\}$, and set of edges $E' = \{(s, t) \in E : s, t \in S'$ (restriction of edges in E to vertices in S'). The algorithm constructs a plan for I by topologically sorting G''_ψ and assigning roles in the order in which they are visited. The generated plan satisfies the properties that for each role $r \in goal$, (1) $UserAssign(r)$ is executed at most once, and $UserRevoke(r)$ is never executed, and for each role $s \notin goal$, $UserAssign(s)$ and $UserRevoke(s)$ are executed at most once, and (2) if a role $r \in goal$ has a pre-requisite condition $\neg s$ where $s \notin goal$ (note that such edges are not present in G'_ψ), then for every child s' of s ($(s, s') \in E'_\psi$, $UserAssign(s')$ precedes $UserRevoke(s)$ which precedes $UserAssign(r)$. ■

To prove the theorem, it remains to show that the running time of Algorithm 1 is polynomial in $|I|$. Constructing graph G_ψ takes polynomial time, and $|G_\psi| = |I|$. Checking the validity of conditions `C1` and `C2` reduces to finding the existence of a cycle in a graph; hence this check can be performed in polynomial time. Constructing G'_ψ and G''_ψ takes polynomial time, and $|G''_\psi| = O(|G_\psi|^2)$. Topologically sorting a graph takes polynomial time. The loop body is executed at most $|G''_\psi|$ times, and each iteration

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

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

1: **if** `C1` or `C2` are `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: $(R, UA) = \gamma$
9: $UA_{new} = UA$
10: **for all** $r \in V''_\psi$ in topologically sorted order **do**
11: $(r_a, c, r) \in can\_assign$
12: **if** $c =$ `false` **then**
13: return `false`
14: **else if** $c = \neg s \wedge s \in UA_{new}$ **then**
15: $P = P.\langle UserRevoke(s)\rangle$
16: $UA_{new} = UA_{new} \setminus \{s\}$
17: **else if** $c = s \wedge s \notin UA_{new}$ **then**
18: return `false`
19: **end if**
20: $P = P.\langle UserAssign(r)\rangle$
21: $UA_{new} = UA_{new} \cup \{r\}$
22: **end for**
23: **if** $goal \subseteq UA_{new}$ **then**
24: return P
25: **else**
26: return `false`
27: **end if**

takes at most linear time in I, so execution of the entire loop takes polynomial time. Thus, the algorithm executes in polynomial time. Each loop iteration adds at most two steps to the plan, so the size of the plan is polynomial in I. Thus, RE for the problem class $[\overline{CR}, \overline{D}, |pre| \leq 1]$ can be solved in polynomial time, and PP for the class is `true`. ■

Thus, RE for the problem class $[\overline{CR}, \overline{D}, |pre| \leq 1]$ can be solved in polynomial time, and PP for the class is `true`. ■

Theorem 15: For the problem class $[\overline{D}, \overline{R}]$, RE is solvable in polynomial time and PP is `true`.

PROOF: Let $I = (\gamma, goal, \psi)$ be a problem instance that satisfies the $[\overline{D}, \overline{R}]$ restriction where $\gamma = \langle R, UA\rangle$, $\psi = \langle can\_assign, \emptyset, \emptyset\rangle$. Simplify *can_assign* by replacing every occurrence of $\neg r$ in any pre-requisite condition with `false` for each assigned role r in the initial state γ. Formally, $can\_assign' = \{(r_a, c, r) \in can\_assign : c$ does not contain $\neg r'$ for some $r' \in UA\}$, and $\psi' = \langle can\_assign', \emptyset, emptyset\rangle$.

Following the proof of Theorem 12, construct the graph $G_{\psi'}$ and $G'_{\psi'} = comp(G_{\psi'}, goal)$. Topologically sort $G'_{\psi'}$ and assign roles in the sorted order. The algorithm is similar to and a simplified version of Algorithm 1 and is given below.

Algorithm 2 Plan generation for problem class $[\overline{D}, \overline{R}]$

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

1: Construct the graph $G'_{\psi'}$.
2: **if** $G'_{\psi'}$ has a cycle **then**
3: return `false`
4: **end if**
5: Topologically sort $G'_{\psi'}$.
6: Plan $P = \langle\rangle$
7: $(R, UA) = \gamma$
8: $UA_{new} = UA$
9: **for all** $r \in V'_{\psi'}$ in topologically sorted order **do**
10: $(r_a, c, r) \in can\_assign$
11: **if** $UA_{new} \nvdash c$ **then**
12: return `false`
13: **else**
14: $P = P.\langle UserAssign(r)\rangle$
15: $UA_{new} = UA_{new} \cup \{r\}$
16: **end if**
17: **end for**
18: **if** $goal \subseteq UA_{new}$ **then**
19: return P
20: **else**
21: return `false`
22: **end if**

Constructing the graph $G'_{\psi'}$ and checking if it has a cycle takes polynomial time in $|I|$. Topologically sorting $G'_{\psi'}$ takes polynomial time. The for-loop iterates at most $|R$ times (where R is the set of roles), and hence executes in polynomial time. Thus, RE for the problem class $[\overline{D}, \overline{R}]$ can be solved in polynomial time. The size of the plan is at most $|R| = O(|I|)$. Thus, PP for the problem class $[\overline{D}, \overline{R}]$ is `true`. ■

Theorem 16: RE for the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$ (where k is a constant) is solvable in polynomial time. In addition, PP for the class is `true`.

PROOF: The proof is similar to the proof in Theorem 3.8 in [Byl94]. Let $I = (\gamma, goal, \psi)$ be a problem instance that satisfies the $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$ restriction where $\gamma = \langle R, UA\rangle$, $\psi = \langle can\_assign, can\_revoke, SMER\rangle$. Then, for each $r \in R$ (1) $(r_a, \texttt{true}, r) \in can\_revoke$ (*i.e.*, every role can be unconditionally revoked, (2) if $(r_a, c, r) \in can\_assign$ then $|c| \leq 1$, and (3) $|goal| \leq k$.

Algorithm 3 checks plan-existence for the above problem class. If a plan exists, the algorithm returns a directed graph $G_\psi = (V_\psi, E_\psi)$ and a set $Init$ of vertices ($Init \subseteq V_\psi$) called *initial* states. Every vertex in the graph is a set of at most k roles; $S \subseteq R$ where $|S| \leq k$. Thus, $|V_\psi| \leq \binom{|R|}{k} 2^k$.

The algorithm starts with the state $\nu = goal$. At each step it picks an unexplored node and explores it. During the process of exploration, new states are generated

Algorithm 3 *Reachability* for the problem $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$

Input: Problem instance $I = (\gamma, goal, \psi)$
Output: Returns a directed graph and a non-empty set of initial states if a plan exists for I, else returns `false`.

```
1:  Vertex set V_ψ = {goal}.
2:  Set of initial states Init = ∅
3:  while There exists an unexplored node in V_ψ do
4:    Pick an unexplored node ν from V_ψ.
5:    Mark ν as explored
6:    (R, UA) = γ
7:    UA_new = UA
8:    if ν ⊆ UA_new then
9:      Init = Init ∪ ν
10:   else
11:     for all r ∈ ν do
12:       for all (r_a, s, r) ∈ can_assign do
13:         if s == true then
14:           ν_new = ν \ {r}
15:         else
16:           ν_new = (ν ∪ {s}) \ {r}
17:         end if
18:         Let S = {r' ∈ R : (r, r') ∈ SMER}
19:         if ν_new ∩ S ≠ ∅ then
20:           Init = ∅
21:           return false
22:         end if
23:         if ν_new ∉ V_ψ then
24:           Add ν_new to V_ψ and mark it unexplored
25:         end if
26:         Add edge e = (ν_new, ν) to E_ψ and set
            label(e) = UserAssign(r)
27:       end for
28:     end for
29:   end if
30: end while
31: return (V_ψ, E_ψ) and Init
```

that are then recursively explored. In this way all states that can reach the goal state are explored. Only those states $\nu \subseteq \gamma$ (γ is the initial state) are not explored since there is a trivial path from γ to such states: $\gamma \xrightarrow{P} \nu$ where $P = \{UserRevoke(r) : r \in gamma \wedge r \notin \nu\}$ (note: when the order of steps in a plan doesn't matter, we sometimes represent it as a set instead of a sequence). All such states are marked as initial states, and are used to start plan generation in the second part of the algorithm. If the set of initial states is $\emptyset$ then there does not exist a plan for I.

If plan-existence for I is `true`, then a plan can be generated from G_ψ and $Init$ as follows. Let $P = \langle e_1, e_2, \ldots, e_n \rangle$ be a path in G_ψ from an initial state $\nu_{init} \in Init$ to the final state $\nu_{final} = goal$. We know that such a path exists, since otherwise $Init = \emptyset$ in which case plan-existence for I would be `false`. 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 can easily be seen that π is a plan for I.

$|G_\psi| = O(|I|^{2k})$ since $|V_\psi| = O(2^k).\binom{|R|}{k}$. Thus, G_ψ can be constructed in polynomial time. Thus, Algorithm 3 for *Reachability* runs in time polynomial in $|I|$. Thus RE for the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$ can be solved in polynomial time. A plan for I is at most the size of the longest path in G_ψ which is $|V_\psi| = O(|I|^k)$. Thus, PP for the problem class $[\overline{CR}, \overline{EN}, |ppre| \leq 1, |G| \leq k]$ is `true`. ■

B. NP-complete Reachability analysis

Theorem 17: For the problem class $[\overline{R}]$, RE is in `NP` and PP is `true`.

PROOF: Let $I = (\gamma, goal, \psi)$ be a reachability analysis problem in the problem class $[\overline{R}]$ where $\gamma = \langle R, UA \rangle$ and $\psi = \langle can\_assign, can\_revoke, SMER \rangle$. Since ψ satisfies the $\overline{R}$ restriction, $can\_revoke = \emptyset$. From Lemma 9, 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 PP 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 $\gamma' = \langle R, UA' \rangle$, $\gamma \xrightarrow{P} \gamma'$, 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 18: RE for the problem class $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER| \leq 1]$ is `NP-hard` [Byl94].

PROOF: The proof follows the proof of `NP-hard`ness of the $PLANSAT_+$ problem (Theorem 3.5 [Byl94]) and is reproduced here for easy readability. We reduce 3-SAT to RE for the above problem class. Consider a 3-SAT formula $\phi = C_1 \wedge C_2 \wedge \ldots C_n$ where each $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$ and each $l_{ij} \in V$ (set of literals). 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. $\gamma = \langle R, \emptyset \rangle$. $goal = \{c_1, c_2, \ldots, c_n\}$.
3. $\psi = \langle can\_assign, can\_revoke, SMER \rangle$ is defined as follows. r_a is the single administrator.
   (a) For $1 \leq i \leq n : (r_a, \texttt{true}, t_i) \in can\_assign$.
   (b) For $1 \leq i \leq n : (r_a, \texttt{true}, f_i) \in can\_assign$.
   (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_i then $(r_a, t_i, c_i) \in can\_assign$, else if l_{ij} is the literal $\neg x_i$ then $(r_a, f_i, c_i) \in can\_assign$.
   (d) $can\_revoke = \emptyset$.
   (e) $SMER = \{(t_i, f_i) : x_i \in V\}$.

$can\_revoke = \emptyset$ - I satisfies the $\overline{R}$ restriction. can_assign does not contain $\neg$ and t_i appears only with f_i in $SMER$ (and vice versa) - I satisfies the $\overline{EN}$ and $|SMER| \leq 1$ restrictions. can_assign also satisfies the $|pre| \leq 1$ restriction. Thus, I is in the problem class $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER| \leq 1]$.

Claim 1: If RE for I is `true` then $\phi = C_1 \wedge C_2 \ldots C_n$ is satisfiable.

PROOF: From Theorem 17 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 $\gamma \xrightarrow{a_1} \gamma_1 \xrightarrow{a_2} \gamma_2 \ldots \xrightarrow{a_m} \gamma_m$, where $\forall 1 \leq i \leq m : \gamma_i = \langle R, UA_i \rangle$. Since P is a plan, $\forall\ 1 \leq i \leq n : c_i \in UA_m$. Thus, for each c_i, $\exists\ j \leq m : a_j = UserAssign(c_i)$. Since $UserAssign(c_i)$ is enabled in γ_{j-1}, there exists a state change rule $(r_a, X, c_i) \in can\_assign$ such that $X \in UA_{j-1}$. Thus, $\exists\ k < j-1 : a_k = UserAssign(X)$. From the construction of I it follows that if $X = t_i$ then $x_i \in C_i$ and $\pi(x_i) =$ `true`, and if $X = f_i$ then $\neg x_i \in C_i$ and $\pi(x_i) =$ `false`. Thus, $\pi(C_i) =$ `true`. Since this is true for every $1 \leq i \leq n$, $\pi(\phi) =$ `true`. Thus, ϕ is satisfiable. ■

Claim 2: 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, $(r_a, t_k, c_i) \in can\_assign$. Similarly, if $l_{ij} = \neg x_k$ then $\pi(x_k) =$ `false` and $UserAssign(f_k) \in P'$, and $(r_a, f_k, c_i) \in can\_assign$. Thus all actions $UserAssign(c_i) \in P''$ are enabled in γ'. Thus $\gamma' \xrightarrow{P''} \gamma''$ where $\gamma'' = \langle R, UA'' \rangle$ and $\forall 1 \leq i \leq n : c_i \in UA''$. Thus $P = P'.P''$ is a plan for I. ■

From Claims 1 and 2 it follows that RE for the problem class $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER| \leq 1]$ is `NP-hard`. ■

Theorem 19: For the problem classes $[\overline{R}]$, $[\overline{R}, \overline{EN}]$, and $[\overline{R}, \overline{EN}, |pre| \leq 1, |SMER| \leq 1]$, RE is `NP-complete` and PP is `true`.

PROOF: Follows immediately from Theorems 17 and 18. ■

C. NP-hard Reachability analysis

Theorem 20: PP for the complexity class $[\overline{D}, \overline{CR}, \overline{EN}, |SMER| \leq 1]$ is `false`.

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

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

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

Define the following sequences.

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

Claim 3: α_n is the minimum size plan for I

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

1. if $i = 2k$ then $\gamma_{2k-1} \xrightarrow{UserAssign(u_{2k})} \gamma_{2k}$, thus $\gamma_0 \xrightarrow{U_{2k}} \gamma_{2k}$
2. if $i = 2k+1$ then $\gamma_{2k} \xrightarrow{U'_{2k-1}.V_{2k-1}, \langle UserAssign(u_{2k+1}) \rangle.V'_{2k-1}} \gamma'$ where $\gamma' = \langle R, \{u_{2k}, u_{2k+1}\} \rangle$. Thus, $\gamma' \xrightarrow{U_{2k-1}} \gamma_{2k+1}$, and $\gamma_0 \xrightarrow{U_{2k+1}} \gamma_{2k+1}$.

Also note that $u_n \in \gamma_n$. Thus, U_n is a plan for I. α_n is a prefix of U_n, thus α_n is a feasible sequence of actions. Since the last action in α_n is $UserAssign(u_n)$, α_n is a plan for I.

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 pre-condition 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
$\nu_{2j} = \langle R, \{u_1, u_2, \ldots, u_{2j-1}, u_{2j}\}\rangle$. Similarly, if $i = 2j+1$ (*i.e.*, i is odd), then since the pre-condition 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 $\nu_{2j+1} = \langle R, \{v_1, v_2, \ldots, v_{2j-1}, u_{2j}, u_{2j+1}\}\rangle$. Thus, if $\nu_0 = \langle R, \emptyset\rangle$, 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}. Thus, $P_1 = \alpha_{2j}$, and it is clear that $P_2 = U'_{2j-1}.V_{2j-1}.\langle UserAssign(u_{2j+1})\rangle$. Thus, $P = \alpha_{2j}.U'_{2j-1}.V_{2j-1}.\langle UserAssign(u_{2j+1})\rangle = \alpha_{2j+1}$. Thus, 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}. Thus, $P_1 = \alpha_{2j-1}$. 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. But when 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}. But since $u_{2j-2} \in \nu_{2j-1}$, P_2 includes α_{2j-2} except the last action $UserAssign(u_{2j-2})$. Therefore $P_2 = V'_{2j-3}.\langle \alpha_{2j-2} \setminus \{UserAssign(u_{2j-2})\}, UserAssign(u_{2j})\rangle$. Thus, $P = \alpha_{2j-1}.V'_{2j-3}.U_{2j-3}.\langle UserAssign(u_{2j})\rangle = U_{2j-1}.\langle UserAssign(u_{2j})\rangle = \alpha_{2j}$. 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. ■

Claim 4: $|\alpha_n| = \Omega(2^{p(n)})$ where $p(n)$ is a polynomial in n.
PROOF: We first show that $|U_n| = \Omega(2^n)$. Note that $|U_{2i+1}| = |U_{2i}| + |U_{2i-1}| + ci$ where c is a constant. Also, $|U_{2i}| = |U_{2i-1}| + 1$. Thus, it follows that $|U_{2i+1}| = 2|U_{2i-1}| + ci$. Thus, $|U_{2i+1}| = 2^i|U_1| + c\Sigma_{j=0}^{i} 2^{i-j}.j = \Omega(2^i)$. Thus, $|U_n| = \Omega(2^{n/2})$. Since $|\alpha_n| > |U_n|$ we have $|\alpha_n| = \Omega(2^{n/2})$. ■

Note that $|I|$ is polynomial in n. Thus, from Claims 3 and 4, it follows that the minimum size plan for I is exponential in $|I|$. Thus, PP for the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |SMER| \leq 1]$ is `false`. ■

Theorem 21: PP for the complexity class $[\overline{D}, \overline{EN}, |ppre| \leq 1, |G| \leq 1]$ is `false`.
PROOF: Consider the problem instance $I = (\gamma, goal, \psi)$ where:

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

For each role r_i, the *can_assign* and *can_revoke* relations have pre-requisite condition 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 $\forall\ 2 \leq i \leq n : R_i = R_{i-1}.S_{i-2}.\langle UserAssign(r_i)\rangle$

It is easy to see that R_n is a plan for I. We can show that R_n is the minimum plan for I similar to proof of Theorem 20. In addition, $|R_k| = |R_{k-1}| + |S_{k-2}| + 1 = |R_{k-1}| + 1 + \Sigma_{i=1}^{k-2}|R'_i| = k - 1 + \Sigma_{i=1}^{k-1}|R_i|$. Since $|R_1| = 1$, it follows that $|R_n| = \Omega(2^n)$. Thus, the minimum plan for I has size exponential in $|I|$. Thus, PP for the problem class $[\overline{D}, \overline{EN}, |ppre| \leq 1]$ is `false` ■

Theorem 22: Bounded Reachability (BRE) for the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |ppre| \leq 2, |SMER| \leq 1]$ is `NP-hard`.
PROOF: The proof is by reduction from the `CLIQUE` problem that is known to be `NP-complete` [Kar72]. Given a graph $G = (V, E)$ and an integer k, the `CLIQUE` problem asks whether G has a clique of size k. We construct a problem instance $I = (\gamma, goal, \psi)$ in the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |ppre| \leq 2, |SMER| \leq 1]$ such that G has a clique of size k if and only if I has a plan of size at most $n^2 + 13n - 2k$ where $|V| = n$. The construction and the proof is based on the proof of Theorem 8 in [BN95].

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

1. $\gamma = \langle R, UA\rangle$. 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 $a_i, b_i, c_i, \bar{c}_i, d_i, \bar{d}_i, e_i, g_{i,1}, g_{i,2}, \ldots, g_{i,n}$, and $h_{i,1}, h_{i,2}, \ldots, h_{i,n}$. Thus $R = \{v_i, a_i, b_i, c_i, \bar{c}_i, d_i, \bar{d}_i, e_i, g_{i,j}, h_{i,j} : 1 \leq i \leq n, 1 \leq j \leq n\}$, and $UA = \{c_i, d_i : 1 \leq i \leq n\}$.
2. $\psi = \langle can\_assign, can\_revoke, SMER\rangle$ where
   - *can_assign* is defined as
     (a) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, v_i) \in can\_assign$
     (b) $\forall 1 \leq i \leq n : (r_a, \bar{c}_i \wedge d_i, a_i) \in can\_assign$
     (c) $\forall 1 \leq i \leq n : (r_a, c_i \wedge \bar{d}_i, b_i) \in can\_assign$
     (d) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, \bar{c}_i) \in can\_assign$
     (e) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, \bar{d}_i) \in can\_assign$
     (f) $\forall 1 \leq i \leq n, : (r_a, \texttt{true}, g_{i,1}) \in can\_assign$, and $\forall 1 \leq i \leq n, \forall 1 \leq j \leq n-1 : (r_a, g_{i,j}, g_{i,j+1}) \in can\_assign$.

(g) $\forall 1 \leq i \leq n, : (r_a, N_{i,1}, h_{i,1}) \in can\_assign$, and $\forall 1 \leq i \leq n, \forall 1 \leq j \leq n-1 : (r_a, h_{i,j} \wedge N_{i,j+1}, h_{i,j+1}) \in can\_assign$, where $N_i = \{v \in V : (v, v_i) \notin E\}$ and $N_{i,j}$ denotes the j^{th} element of the set N_i.
(h) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, e_i) \in can\_assign$

- *can_revoke* is defined as
(a) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, a_i) \in can\_revoke$
(b) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, b_i) \in can\_revoke$
(c) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, c_i) \in can\_revoke$
(d) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, \bar{c}_i) \in can\_revoke$
(e) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, d_i) \in can\_revoke$
(f) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, \bar{d}_i) \in can\_revoke$
- $SMER$ is defined as
(a) $\forall 1 \leq i \leq n : (c_i, \bar{c}_i) \in SMER$
(b) $\forall 1 \leq i \leq n : (d_i, \bar{d}_i) \in SMER$
(c) $\forall 1 \leq i \leq n : (v_i, \bar{e}_i) \in SMER$

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

Note that all the roles have pre-requisite conditions of size at most 2 - I satisfies the $|ppre| \leq 2$ restriction. Each role $r \in R$ has at most one state change rule in *can_assign* and *can_revoke* - I satisfies the $\overline{D}$ restriction, each role is allowed to be unconditionally revoked - I satisfies the $\overline{CR}$ restriction, and negation is specified only as SMER constraints, and each role appears at most once in a SMER constraint - c_i with $\bar{c}_i$, d_i with $\bar{d}_i$ and v_i with e_i - I satisfies the $\overline{EN}$ and $|SMER| \leq 1$ restrictions. Thus, I is in the problem class $[\overline{D}, \overline{CR}, \overline{EN}, |ppre| \leq 2, |SMER| \leq 1]$.

Claim 5: Suppose $G = (V, E)$ has a clique of size k. Then I has a plan of size $n^2 + 13n - 2k$.

PROOF: Let C be a clique in G of size k. Consider the sequence of actions $P = P_1.P_2.P_3.P_4.P_5.P_6.P_7.P_8.P_9$ where UA and UR abbreviate $UserAssign$ and $UserRevoke$ respectively.

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

From the definition of *can_assign* for the roles $h_{i,j}$, it follows that for every $v_i \in C$, the sequence of actions H_i can be executed in a state where every $v_i \in V - C$ is `true`. Thus, P_4 is a feasible subsequence of P. Also, the sequence of actions G_i can be executed from any state. Thus, P_5 is also a feasible subsequence of P. Now, it is easy to see that P is indeed a plan for I and $|P| = n^2 + 13n - 2k$. ■

Claim 6: Suppose there exists a plan for I of length $n^2 + 13n - 2k$ or less. Then G has a clique of size k.

PROOF: Let P be a plan for I and let $|P| = p \leq n^2 + 13n - 2k$ be the length of P.

Since for each $1 \leq i \leq n$, $\neg v_i \in \gamma$ and $e_i \in goal$, a $UA(v_i)$ must be followed by a $UR(v_i)$. Let P contain

$$UA(v_i), \ldots, UR(v_i), \ldots$$

for all $v_i \in W$ where $W \subseteq V$.

For each $1 \leq i \leq n$, P must contain $UA(e_i)$ for all $1 \leq i \leq n$ and either

$$UR(c_i), ..., UA(\bar{c}_i), ..., UA(a_i), ..., UR(\bar{c}_i), ..., G_i, UA(c_i), ..., UR(d_i), ..., UA(\bar{d}_i), ..., UA(b_i), ..., UR(c_i), ..., UA(\bar{c}_i)$$

or

$$UR(d_i), ..., UA(\bar{d}_i), ..., UA(b_i), ..., UR(\bar{d}_i), ..., H_i, UA(d_i), ..., UR(c_i), ..., UA(\bar{c}_i), ..., UA(a_i), ..., UR(d_i), ..., UA(\bar{d}_i)$$

Suppose $v_t \in V - W$. Then, since $UserAssign(v_t) \notin P$, and the pre-requisite condition for $UserAssign(c_t)$ is v_t, it follows that P contains

$$UR(d_t), ..., UA(\bar{d}_t), ..., UA(b_t), ..., UR(\bar{d}_t), ..., H_t, UA(d_t), ..., UR(c_t), ..., UA(\bar{c}_t), ..., UA(a_t), ..., UR(d_t), ...UA(\bar{d}_t)$$

Consider a vertex $v_s \in V - W$. Then, 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* it follows that v_s is in the pre-requisite condition of $UserAssign(d_t)$. Thus, since $UserAssign(d_t) \in P$, it follows that $UserAssign(v_s) \in P$ conttradicting that $v_s \in V - W$. Thus, there is an edge $(v_s, v_t) \in E$. Since this is true for any vertex in $V - W$, it follows that $V - W$ is a clique.

The size of the plan containing the above sequences is $p' = |V| + 2|W| + (10 + |V|)|V - W| + (10 + |V|)|W| = n^2 + 11n + 2|W| = n^2 + 13n - 2|V - W|$. Since $p' \leq p = n^2 + 13n - 2k$, it follows that $|V - W| \geq k$. Thus, G contains a clique of size k. ■

From Claims 5 and 6 it follows that BRE for the class $[\overline{D}, \overline{CR}, \overline{EN}, |ppre| \leq 2, |SMER| \leq 1]$ is `NP-hard`. ■

Theorem 23: Bounded Reachability (BRE) for the problem class $[\overline{D}, \overline{EN}, |ppre| \leq 1]$ is `NP-hard`.

PROOF: The proof is similar to that of Theorem 22. Given a graph $G = (V, E)$ 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. $\gamma = \langle R, UA \rangle$. 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. $\psi = \langle can\_assign, can\_revoke, SMER \rangle$ where
- *can_assign* is defined as
(a) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, v_i) \in can\_assign$
(b) $\forall 1 \leq i \leq n : (r_a, d_i, a_i) \in can\_assign$

(c) $\forall 1 \leq i \leq n : (r_a, c_i, b_i) \in can\_assign$
(d) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, c_i) \in can\_assign$
(e) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, d_i) \in can\_assign$
(f) $\forall 1 \leq i \leq n : (r_a, a_i, e_i) \in can\_assign$
(g) $\forall 1 \leq i \leq n : (r_a, b_i, f_i) \in can\_assign$
(h) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, g_i) \in can\_assign$
(i) $\forall 1 \leq i \leq n : (r_a, \texttt{true}, h_i) \in can\_assign$

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

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

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

Claim 7: If G has a clique of size k, then I has a plan of size $13n - 2k$.

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. $P_1 = \ldots UR(c_i), UA(a_i), UA(e_i) \ldots$ where $v_i \in V - C$. $|P_1| = 3n - 3k$.
2. $P_2 = \ldots UR(d_i), UA(b_i), UA(f_i) \ldots$ where $v_i \in C$. $|P_2| = 3k$.
3. $P_3 = \ldots UR(c_i) \ldots$ where $v_i \in C$. $|P_3| = k$.
4. $P_4 = \ldots UR(d_i) \ldots$ where $v_i \in V - C$. $|P_4| = n - k$.
5. $P_5 = \ldots UA(v_i) \ldots$ for all $1 \leq i \leq n$. $|P_5| = n$.
6. $P_6 = \ldots UR(a_i) \ldots$ where $v_i \in V - C$. $|P_6| = n - k$.
7. $P_7 = \ldots UR(b_i) \ldots$ where $v_i \in C$. $|P_7| = k$.
8. $P_8 = \ldots UR(v_i) \ldots$ where $v_i \in V - C$. $|P_8| = n - k$.
9. $P_9 = \ldots UA(c_i), UA(b_i), UA(f_i), UR(c_i) \ldots$ where $v_i \in V - C$. $|P_9| = 4n - 4k$.
10. $P_{10} = \ldots UA(d_i), UA(a_i), UA(e_i), UR(d_i) \ldots$ where $v_i \in C$. $|P_10| = 4k$.
11. $P_{11} = \ldots UA(v_i) \ldots$ where $v_i \in V - C$. $|P_{11}| = n - k$.
12. $P_{12} = \ldots UR(a_i) \ldots$ where $v_i \in C$. $|P_{12}| = k$.
13. $P_{13} = \ldots UR(b_i) \ldots$ where $v_i \in V - C$. $|P_{13}| = n - k$.
14. $P_{14} = \ldots UA(g_i) UA(h_i) \ldots$ where $1 \leq i \leq n$. $|P_{14}| = 2n$.

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

Claim 8: 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 Claim 6. ■

From Claims 7 and 8 it follows that G has a clique of size k if and only if I has a plan of size at most $13n - 2k$. Thus, BRE for $[\overline{D}, \overline{EN}, |ppre| \leq 1]$ is `NP-hard`. ■

D. PSPACE-complete Reachability Analysis

Theorem 24 is based on complexity results for SAS$^+$ planning described in [BN95]. Next, we describe the SAS$^+$ planning model of [BN95].

Definition 5: An instance of the SAS$^+$ planning problem is given by the tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ 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 $\langle pre, post, prv \rangle$ where $pre, post, prv \in S^+$ denote the **pre**, **post**, and **prevail** conditions. $\mathcal{O}$ is subject to the following restrictions. For every operator $\langle pre, post, prv \rangle$,
  – **(R1)** $\forall v \in V : pre[v] \neq u \rightarrow pre[v] \neq post[v] \neq u$.
  – **(R2)** $\forall v \in V : post[v] = u$ 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 $o = \langle pre, post, prv \rangle$ is an operator, we write $pre(o)$ to denote pre, *etc.*.

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

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

Theorem 24: Reachability for the problem class without any restrictions is `PSPACE-complete`.

Claim 9: Reachability for unrestricted ARBAC policies is in PSPACE.

PROOF: A non-deterministic Turing Machine can solve the problem in polynomial space as follows. Starting from the initial state, it can guess the next action, execute the action, and transform the initial state to a new state. It then discards the old state, and stores only the newly reached state. It continues this process until the required goal state is reached. Since only a single state is stored at any point in time, and the size of the state is polynomial in size of the problem instance, the Turing Machine takes polynomial space. ■

Claim 10: Reachability for unrestricted ARBAC policies is PSPACE-hard.

PROOF: [BN95] shows that Plan-Existence for a SAS^+ planning problem under the U and B restrictions is `PSPACE-complete`. We reduce this problem to to the *Reachability* problem for unrestricted ARBAC policy.

Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS^+ problem instance that satisfies the U and B restrictions. Construct a *Reachability* problem $I = (\gamma, goal, \psi)$ where $\psi = \langle can\_assign, can\_revoke, SMER \rangle$ is the unrestricted ARBAC policy, $SMER = \emptyset$, $\gamma = \langle R, UA \rangle$ is the initial RBAC policy, and *goal* is the set of roles to be proved as follows.

1. Define $R = \mathcal{V}$. Since for all $v \in \mathcal{V}$, $|D_v| = 2$, every $v \in \mathcal{V}$ is a boolean variable. Thus, the set of roles R is well defined.
2. Define $UA = \{v \in \mathcal{V} : s_0[v] = \texttt{true}$.
3. Define $goal = \{v \in \mathcal{V} : s_*[v] = \texttt{true}$.
4. For each operator $o \in \mathcal{O}$, let $v \in \mathcal{V}$ be the unique variable for which $post(o)[v] \neq u$.
   - For each $w \in \mathcal{V}$, if $prv(o)[w] = \texttt{true}$, then $w \in c$, and if $prv(o)[w] = \texttt{false}$ then $\neg w \in c$.
   - if $pre(o)[v] = \texttt{false}$ (equivalently, $post(o)[v] = \texttt{true}$), then include (c, v) in can_assign.
   - if $pre(o)[v] = \texttt{false}$ (equivalently, $post(o)[v] = \texttt{true}$), then include (c, v) in can_revoke.

Note that $|I| = O(|\Pi|)$. It is clear that γ is a well-defined RBAC policy, and that ψ is a well-defined ARBAC policy. It is easy to see that any plan for Π is also a plan for I, and any plan for I is also a plan for Π. Thus, we have reduced Plan-Existence for SAS^+ planning under U and B restrictions to the *Reachability* problem for unrestricted ARBAC policy. Thus, the *Reachability* problem for unrestricted ARBAC policy is PSPACE-hard. ■

From Claims 9 and 10 it follows that the *Reachability* problem for unrestricted ARBAC policies is PSPACE-complete. ■