Efficient Policy Analysis for Administrative Role Based Access Control[*]

Scott D. Stoller[†] Ping Yang[‡] C.R. Ramakrishnan[†] Mikhail I. Gofman[‡]

ABSTRACT

Administrative RBAC (ARBAC) policies specify how Role-Based Access Control (RBAC) policies may be changed by each administrator. It is often difficult to fully understand the effect of an ARBAC policy by simple inspection, because sequences of changes by different administrators may interact in unexpected ways. ARBAC policy analysis algorithms can help by answering questions, such as user-role reachability, which asks whether a given user can be assigned to given roles by given administrators. This problem is intractable in general. This paper identifies classes of policies of practical interest, develops analysis algorithms for them, and analyzes their parameterized complexity, showing that the algorithms may have high complexity with respect to some parameter k characterizing the hardness of the input (such that k is often small in practice) but have polynomial complexity in terms of the overall input size when the value of k is fixed.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security; D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*

General Terms: Security

1. INTRODUCTION

Role Based Access Control (RBAC) is a widely used framework for access control [26]. RBAC simplifies access control by decomposing the association of permissions with users into two relations: the user-role assignment specifies the roles of each user, and the role-permission assignment specifies the permissions associated with each role. Inheritance relationships may also be specified, in the form of a role hierarchy.

[*]This work was supported in part by NSF under grant CCR-0205376 and by ONR under grants N00014-04-1-0722 and N00014-07-1-0928.

[†]Computer Science Dept., Stony Brook University, Stony Brook, NY, 11794-4400. Email: stoller@cs.stonybrook.edu, cram@cs.stonybrook.edu.

[‡]Dept. of Computer Science, Binghamton University, Binghamton, NY, 13902. Email: pyang@cs.binghamton.edu, mgofman1@binghamton.edu

Administrative RBAC (ARBAC) policies specify how RBAC policies may be changed by each administrator. ARBAC policies are important for security in organizations with multiple administrators. Several ARBAC models have been proposed [25, 23, 5, 18]. In this paper, we focus on the classic ARBAC97 model [25], although many of our results can be adapted to other ARBAC models, as discussed in Section 2.2. ARBAC97 decomposes the administration problem into three simpler sub-problems, for controlling changes to the user-role assignment, role-permission assignment, and role hierarchy. Administrative operations on the user-role assignment include adding users to roles, and removing users from roles. Permission to perform these administrative operations is specified by the can_assign and can_revoke relations, respectively. A can_assign rule has the form "administrators in role r_a can add users satisfying precondition c to role r". The precondition may contain positive preconditions (roles the user must already be in) and negative preconditions (roles the user must not be in). For example, assignment of the Dean role may require that the user is in the Faculty role and not in the DepartmentChair role. In short, an ARBAC policy defines a transition relation that describes allowed (possible) changes to the RBAC policy.

Correct understanding of a system's current RBAC policy and the implications of its ARBAC policy are critical to assure system security. Although ARBAC policies are specified using relatively simple rule languages, it is often difficult to understand the effect of an ARBAC policy by simple inspection, largely because (without help) people often fail to see the full effects of multiple-step (transitive) relations and their interactions. The transitive relations here are the RBAC policy reachability relation (*i.e.*, the transitive closure of the ARBAC policy's transition relation) and the inheritance relation (*i.e.*, the role hierarchy). Policy analysis helps system designers and administrators understand RBAC and ARBAC policies by answering questions (queries) about them. For example, user-role reachability analysis answers questions of the form: given an initial RBAC policy state, an ARBAC policy, a set of administrators, a target user, and a set of roles (called the "goal"), is it possible for those administrators to modify the RBAC state so that the target user is a member of those roles? Unfortunately, many policy analysis problems, including user-role reachability, are intractable even under various restrictions on the ARBAC policy [20, 27]. Those papers give polynomial algorithms for some special cases, but the restrictions significantly limit the practical applicability of the algorithms.

Can practical algorithms be devised to analyze useful classes of ARBAC policies, despite the intractability results? We tackle this question by identifying classes of policies of practical interest, developing analysis algorithms for them, and analyzing their *parameterized complexity* [6]. The key idea of parameterized complexity is to identify an aspect of the input that make the problem computationally difficult, introduce a parameter k to measure that aspect of the input, and develop a solution algorithm that may have high complexity in terms of k, but has polynomial complexity in terms of the overall input size when the value of k is fixed. If such an algorithm exists, the problem is said to be *fixed parameter tractable* (FPT) with respect to k. Even if the problem is NP-hard or worse, the FPT algorithm is efficient for problem instances with relatively small values of k.

Main Contributions. We consider a variant of ARBAC97 called miniARBAC.

1. We give a general algorithm for user-role reachability analysis of general miniARBAC policies (see Section 3), and show that it is fixed-parameter tractable with respect to the number of mixed roles (roles that appear both negatively and positively in the policy). The algorithm uses forward search optimized using a *reduction* that exploits left-commutativity of transitions. We describe how to *slice* a policy with respect to a goal, to help avoid exploration of irrelevant states.

2. We give an efficient backward-search algorithm for user-role reachability analysis for miniARBAC policies with at most one positive precondition per rule. The algorithm is fixed-parameter tractable with respect to the number of irrevocable roles and has a similar tractability property with respect to the size of the goal (see Section 4). We also present a partial-order reduction to optimize the backward search.

3. The above algorithms rely on an aspect of ARBAC97 that we call *separate administration*, which requires that administrative roles and regular roles are disjoint. Prior work on ARBAC policy analysis [29, 20, 27] generally makes this assumption, but it is unrealistic in many cases. We lift this restriction and explore two approaches to policy analysis in this more general setting, by identifying conditions under which the general problem can be reduced to policy analysis with separate administration, and by extending our forward analysis algorithm to handle the general problem (see Section 5).

4. We describe two case studies: ARBAC policies for a university and a health-care institution (see Section 7). We observe several structural properties of them, and relate them to the assumptions and complexity parameters of our algorithms.

5. We measure the performance of our analysis algorithms on families of synthetic (randomly generated) policies, in order to validate our parameterized complexity results, determine whether the worst-case complexity manifests itself, and compare the performance of the forward and backward analysis algorithms when both apply (see Section 8).

6. We also consider other analysis problems, including role containment (is every member of role r_1 also a member of a role r_2 in all reachable policy states?) [20] and weakest preconditions (what are minimal sets of initial roles for a user, in order for that user to get added to roles in the goal?) (see Section 6).

2. PRELIMINARIES

2.1 Role Based Access Control (RBAC)

The central notion of RBAC is that users are assigned to appropriate roles, and roles are assigned appropriate permissions. In this paper, we study policy analysis for models of RBAC based on [1]. Following Sasturkar *et al.* [27], we adopt a simplified model, called miniRBAC, that does not support sessions, because the policy analysis queries we consider are independent of sessions.

miniRBAC. A miniRBAC policy γ is a tuple $\langle U, R, P, UA, PA\rangle$ where

- U, R and P are finite sets of users, roles, and permissions, respectively. A permission represents authorization to invoke a particular operation on a particular resource.

- $UA \subseteq U \times R$ is the user-role assignment relation. $\langle u, r\rangle \in UA$ means that user u is a member of role r.

- $PA \subseteq P \times R$ is the permission-role assignment relation. $\langle p, r\rangle \in PA$ means that members of role r are granted the permission p.

The miniHRBAC model based on Hierarchical RBAC [1] extends the miniRBAC model with *role hierarchies*, which are a natural means for structuring roles to reflect an organization's lines of authority and responsibility.

miniHRBAC. A miniHRBAC policy γ_h is a tuple $\langle U, R, P, UA, PA, \succeq\rangle$ where U, R, P, UA and PA are as in miniRBAC, and $\succeq \subseteq R \times R$ is a partial order on the set R of roles.

$r_1 \succeq r_2$ means r_1 is senior to r_2, *i.e.*, every member of r_1 is also a member of r_2, and every permission assigned to r_2 is also available to members of r_1. Thus, r_2 inherits all the users of r_1 and r_1 inherits all the permissions of r_2. A user u is an *explicit member* of a role r if $\langle u, r\rangle \in UA$. A user u is an *implicit member* of a role r if $\langle u, r'\rangle \in UA$ for some r' such that $r' \succeq r$ and $r' \neq r$.

2.2 Administrative Role Based Access Control (ARBAC)

ARBAC97 is a classic model for decentralized administration of RBAC policies [25]. ARBAC97 has three components: (1) user-role administration (URA), (2) permission-role administration (PRA), and (3) role-role administration (RRA) for administration of the role hierarchy. We consider a modified version of ARBAC97 called miniARBAC similar to [27]. miniARBAC includes only the URA component; the permission assignment and role hierarchy are considered fixed. Extending our analysis techniques to handle changes to them is discussed at the end of this section. Since we do not consider changes to the role hierarchy, we adopt a simple ARBAC model without authority ranges [25], scopes [17, 5], or administrative domains [18]; policies using those features can be expressed without them when the role hierarchy is fixed.

The URA policy controls changes to the user-role assignment UA. Permission to assign users to roles is specified by the relation $can\_assign \subseteq R \times C \times R$, where C is the set of all preconditions (called prerequisite conditions in [25]) over R. The precondition is a conjunction of *literals*, where each literal is either r or $\neg r$ for some role r in R. Given a miniRBAC policy γ and a user u, u satisfies a precondition $\wedge_i l_i$, denoted $u \models_\gamma \wedge_i l_i$, iff for all i, either l_i is a role r and u is a member of r in γ, or l_i is a negated role $\neg r$ and u is not a member of r in γ. A $UserAssign(r_a, u, r)$ action specifies that an administrator who is a member of the administrative role r_a adds user u to role r. This action is enabled in state $\gamma = \langle U, P, R, UA, PA\rangle$ iff there exists a precondition c such that $\langle r_a, c, r\rangle \in can\_assign$ and $u \models_\gamma c$. Execution of this action transforms γ to the state $\gamma' = \langle U, P, R, UA \cup \{\langle u, r\rangle\}, PA\rangle$.

Permission to revoke users from roles is specified by the relation $can\_revoke \subseteq R \times R$. This is analogous to can_assign, except that it does not allow preconditions, because there is little evidence that preconditions for revocation are useful [25]. The action $\mathit{UserRevoke}(r_a, u, r)$ is defined similarly to $\mathit{UserAssign}$.

miniARBAC policy. A miniARBAC policy is represented as $\psi = \langle \mathit{can\_assign}, \mathit{can\_revoke} \rangle$. We often refer to tuples in these two relations as *rules*. The role r to which users or permissions are being assigned or removed (*i.e.*, the third component of the tuple) is called the *target* of the rule. A miniARBAC policy specifies a transition relation between miniRBAC (or miniHRBAC) policies, which we often refer to as *states*. A transition is denoted by $\gamma \xrightarrow{a}_{\psi} \gamma'$, where a is one of the four administrative actions defined above. When we do not care about the action a, we omit it from the transition, and when the miniARBAC policy ψ is clear from context, we omit it from the transition.

SMER Constraints. A Static Mutually Exclusive Roles (SMER) constraint is an unordered pair of roles $\langle r_1, r_2 \rangle$ whose membership is required to be disjoint. SMER constraints help enforce separation of duty. We treat SMER constraints as an abbreviation for negative preconditions that enforce the required disjointness. Specifically, a SMER constraint $\langle r_1, r_2 \rangle$ is an abbreviation for conjoining $\neg r_1$ to the precondition of each $\mathit{can\_assign}$ tuple with target r_2, and *vice versa*.

Separate Administration. A role r is an *administrative role* if it has an administrative permission, *i.e.*, there is a $\mathit{can\_assign}$ tuple with r in the first component. A role r is a *regular role* if it has a regular permission, *i.e.*, there is a *PA* tuple with r in the first component. Our framework, like SARBAC [5], UARBAC [18], and Oracle, allows a role to be both a regular role and an administrative role. This flexibility allows many policies to be expressed more naturally. For example, in a university, a department chair has both regular permissions (*e.g.*, authorize expenses from the department's accounts) and administrative permissions (*e.g.*, appoint faculty to committees).

Earlier work on ARBAC, such as ARBAC97 and ARBAC02 [25, 23], requires that (1) regular roles and administrative roles are separate (*i.e.*, no role is in both categories), and (2) in every $\mathit{can\_assign}$ tuple, the first component is an administrative role, the condition mentions only regular roles, and the target is a regular role. We call this the *separate administration* restriction. Work on ARBAC policy analysis [29, 27, 20] generally adopts this restriction, with the exception of analysis for the AATU model in [20].[1] This helps simplify a difficult problem, enabling steps towards more general solutions. The AATU model in [20] does not adopt this restriction, but adopts two other big restrictions instead (no revocation; no negative preconditions or SMER constraints).

The analysis algorithms in Sections 3 and 4 take advantage of the separate administration restriction. Section 5 tackles policy analysis without this restriction.

Other ARBAC Frameworks. Our focus on ARBAC97-like user-role administration model is not a fundamental limitation of our work. The proposals for user-role administration, permission-role administration, and role-hierarchy administration in [25, 23, 17, 5, 18] differ from each other in important ways, but the aspects most relevant to our algorithms—primarily the form of preconditions of administrative operations—are generally not more complicated than in the URA policies we analyze, so the ideas underlying our algorithms can be adapted to analyze them. Handling RHA's conditions for scope preservation [5] would require some extensions. UARBAC^P [18] is a schematic framework that allows arbitrary constraints over arbitrary types to be used as (roughly speaking) preconditions. Our algorithms can be adapted for instantiations of UARBAC^P that use typical constraints such as checking whether a user or object is a member of a given role, scope, or domain.

[1]The restriction adopted in the AAR model in [20] is similar although not identical.

2.3 User-Role Reachability

The user-role reachability problem is: Given an initial miniRBAC policy $\gamma_0 = \langle R, \mathit{UA}_0 \rangle$, a miniARBAC policy ψ, a set U_0 of users, a user u_t in U_0 (called the "target user"), and a set goal of roles, can the users in U_0 together transform γ_0 (under the restrictions imposed by ψ) to another miniRBAC policy γ in which u_t is a member of all roles in goal? A sequence of administrative actions (*i.e.*, $\mathit{UserAssign}$ and $\mathit{UserRevoke}$ actions) that performs such a transformation is called a *plan* for (or *solution* to) the problem instance. We will often refer to user-role reachability simply as "reachability". A reachability problem instance can be represented as a tuple $\langle \gamma_0, \psi, U_0, u_t, \mathit{goal} \rangle$. We sometimes refer to miniRBAC policies as "states".

Reachability Under Separate Administration. The separate administration restriction allows the specification of a problem instance to be simplified. When adopting this restriction, we also assume, without loss of generality, that the target user u_t is initially a member of regular roles only (if the target user were initially a member of administrative roles, we could give those roles to another user in U_0 instead, without affecting the answer to the reachability problem). Note that the separate administration restriction implies that the target user cannot be added later to an administrative role.

We call the other users in U_0 "administrators" (we could require that they are members of administrative roles only, but allowing them to be members of regular roles as well has no impact on the analysis). With this restriction, it is sufficient to consider only administrative actions by the administrators that change the target user's role memberships. Let A be the set of administrative roles of users in U_0 in the initial state γ_0. We can merge those roles into a single administrative role with the union of the administrative permissions of roles in A, and eliminate all other administrative roles. We can then make this single administrative role implicit, *i.e.*, we can eliminate the first component of the $\mathit{can\_assign}$ and $\mathit{can\_revoke}$ relations. Similarly, because role memberships of different users are independent, we can eliminate all users other than the target user. We can then make this user implicit, *i.e.*, we can eliminate the first component of the user assignment UA. We can also eliminate the first two parameters of administrative actions.

With these simplifications, a reachability problem instance can be represented as a tuple $\langle \gamma_0, \psi, \mathit{goal} \rangle$ where $\gamma_0 = \langle R, \mathit{UA}_0 \rangle$ is a simplified miniRBAC policy, $\psi = \langle \mathit{can\_assign}, \mathit{can\_revoke} \rangle$ is a simplified miniARBAC policy, and $\mathit{goal} \subseteq R$. Since the set of roles is fixed, we sometimes elide it, representing a state as UA, instead of $\langle R, \mathit{UA} \rangle$.

2.4 Parameterized Complexity

Parameterized complexity [6] is an approach to deal with computationally difficult problems. The idea is to identify an aspect of the input that makes the problem computationally difficult, introduce a parameter to measure that aspect of the input, and develop a solution algorithm that may have high complexity in terms of that parameter, but has polynomial complexity in terms of the overall input size when the value of that parameter is fixed. This is called fixed-parameter tractability. Formally, a problem is *fixed-parameter tractable (FPT)* with respect to parameter k if there exists an algorithm that solves it in $O(f(k) \times n^c)$ time, where f is an arbitrary function (depending only on its argument k), n is the input size, and c is a constant.

We say that a problem is *fixed-parameter polynomial* with respect to parameter k if there is an algorithm that solves it in $O(n^{ck})$ time, where n is the input size, and c is a constant. Note that, for a fixed value of k, the time complexity is polynomial in n. We say that a problem is *fixed-parameter k_1-tractable and k_2-polynomial* if there exists an algorithm that solves it in $O(f(k_1) \times n^{ck_2})$ time for some function f and constant c.

3. FIXED-PARAMETER TRACTABILITY OF REACHABILITY UNDER SEPARATE ADMINISTRATION

This section presents parameterized complexity results for user-role reachability under the separate administration restriction. Our exposition is for policies without role hierarchy. Analysis of policies with role hierarchy can be reduced to analysis of policies without role hierarchy, by transforming the policy and goal to make the effects of inheritance explicit, as described in [27].

A role is *negative* (in a problem instance) if it appears negated in some precondition in the policy; other roles are *non-negative*. A role is *positive* if it appears positively (*i.e.*, not negated) in some precondition in the policy or appears in the goal; other roles are called *non-positive roles*. A role that is both negative and positive is called *mixed*. This section shows that reachability analysis is fixed-parameter tractable with respect to the number of mixed roles. We prove this constructively, by giving a fixed-parameter tractable algorithm based on a reduction theorem that shows that it is safe to execute certain sequences of transitions atomically, *i.e.*, as a single larger (composite) transition. In some ways, our reduction is a special case of Lipton's reduction [21], but in another way, our reduction differs from Lipton's reduction and its numerous successors. Those reductions justify treating given sequences of transitions (which appear in the control flow graph of the program) as atomic, *i.e.*, as composite transitions. An ARBAC policy has no control flow, so our reduction itself defines composite transitions, and justifies using them instead of the original transitions.[2] In addition, we prove fixed-parameter tractability for our algorithm. We are not aware of any similar complexity results in the literature for reductions or partial-order reductions, whose performance is usually evaluated in a purely empirical way.

The heart of our method is the definition of a reduced transition relation $\rightsquigarrow$ that takes larger steps than the original transition relation $\rightarrow$; specifically, a single step of $\rightsquigarrow$ may correspond to multiple steps of $\rightarrow$. The reachability algorithm itself is a straightforward exploration of the states reachable from the initial state via the reduced transition relation. Note that increasing the transition size by a factor of k can reduce the number of explored states by a much greater factor, because it can eliminate many intermediate states produced by execution of different subsets of the original transitions that are aggregated into the reduced transitions.

An *invisible transition* is a transition that adds a non-negative role or revokes a non-positive role; other transitions are called *visible transitions*. The reduced transition relation differs from the original transition relation in two ways. (1) Transitions that revoke non-negative roles or add non-positive roles are prohibited; they are useless because they do not add roles in the goal and do not enable any transitions. (2) Invisible transitions get combined with a preceding visible transition to form a single composite transition. Invisible transitions can safely be executed immediately after the preceding visible transition, because they never disable any transitions.

[2]Our algorithm is not a special case of traditional partial-order reductions [9, 4], because our algorithm exploits the fact that certain transitions are left-movers, while traditional partial-order algorithms exploit only full commutativity (independence) of transitions.

More formally, for a state γ, let $\mathrm{closure}(\gamma)$ denote the largest (with respect to $\supseteq$) state γ' reachable from γ by executing invisible transitions. A straightforward proof, based on the definition of invisible transition, shows that this closure is well-defined, *i.e.*, there is a unique largest such state. The reduced transition relation is defined by: $\gamma_1 \overset{a}{\rightsquigarrow} \gamma_2$ iff there exists a state γ such that $\gamma_1 \xrightarrow{a} \gamma$ and $\gamma_2 = \mathrm{closure}(\gamma)$ and $\gamma_1 \neq \gamma_2$ and a is $UserAssign(r)$ or $UserRevoke(r)$ for some negative role r (we don't need to allow non-negative roles here, because they are added implicitly by closure and never revoked).

The following theorem shows that user-role reachability can be solved by exploring the reduced transition relation. For a problem instance I, let PR_I, NR_I, and $\overline{NR_I}$ denote the sets of positive, negative, and non-negative roles, respectively, in I. For a relation $\rightarrow$, let $\rightarrow^*$ denote its reflexive-transitive closure. A goal is *reachable* from a state γ_0 iff $(\exists \gamma : \gamma_0 \rightarrow^* \gamma \wedge goal \subseteq \gamma)$.

THEOREM 1. *For all miniRBAC states γ_0 and all goals goal, goal is reachable from γ_0 iff $(\exists \gamma : \mathrm{closure}(\gamma_0) \rightsquigarrow^* \gamma \wedge goal \subseteq \gamma)$.*

The proof is straightforward. This theorem implies that reachability analysis can be solved by computing the states reachable from $\mathrm{closure}(I)$ via $\rightsquigarrow$, and checking whether the goal is a subset of any of the resulting states. The graph constructed by this computation is called the *reduced state graph*.

THEOREM 2. *The reduced state graph can be constructed in time $O(f(|NR_I \cap PR_I|)|I|^c)$, for some function f and some constant c. Thus, user-role reachability is fixed-parameter tractable with respect to the number of mixed roles.*

PROOF: To reduce clutter, we omit the subscript I on NR and PR. We introduce some terminology. An *NR-state* is a subset of NR. The NR-state graph is the projection of the reduced state graph onto NR-states; thus, an edge (s_1, s_2) in the state graph induces an edge $(s_1 \cap NR, s_2 \cap NR)$ in the NR-state graph. Let G_{red} and G_{NR} denote the reduced state graph and the NR-state graph, respectively.

We show that the number of states in G_{red} is $O(f(|NR \cap PR|))$ for some function f. Every state in G_{red} is reachable by a simple path in G_{red}. Every simple path in G_{red} corresponds, by projection, to a distinct path in G_{NR}, because every $\rightsquigarrow$ transition changes the set of NR roles in the state. Furthermore, these paths in G_{NR} contain at most one occurrence of each cycle in G_{NR}, because going around a cycle in G_{NR} a second time would not add any more $\overline{NR}$ roles to the state, hence the corresponding fragment of the path in G_{red} (note that a path in G_{NR} uniquely determines a corresponding path in G_{red}, because $\rightsquigarrow$ adds *all* allowed positive roles at each step) would be a cycle, contradicting the assumption that the path in G_{red} is simple. Therefore, the number of states in G_{red} is bounded by the number of paths in G_{NR} that go around each cycle at most once. This number is clearly bounded by some function of the number of nodes in G_{NR}. The number of nodes in G_{NR} is clearly bounded by some function of $|NR|$. To see that it is bounded by some function of $|NR \cap PR|$, note that the set of non-positive roles in the state is the same in every state in G_{red} except the initial state, because the reduced transition relation never adds non-positive roles to the state, and revocation has no preconditions, and hence invisible transitions that revoke non-positive roles occur only in composite transitions leaving the initial state. Thus, the set of non-positive roles is the same in every state in G_{NR} except the initial state, so the number of nodes in G_{NR} is bounded by some function of the number of positive roles in NR.

The time complexity of standard state-graph construction algorithms is polynomial in the size of the input and linear in the size of the output (*i.e.*, the reduced state graph). Therefore, the worst-case time complexity of constructing the reduced state graph is $O(f(|NR \cap PR|)|I|^c)$, for some function f and some constant c.

□

Slicing. Before applying the above algorithm, we apply a slicing transformation that back-chains along the rules to identify roles and rules relevant to the given goal, and then eliminates the irrelevant ones. The special twist here, compared to traditional cone-of-influence reduction [4], is to take into account whether a role appears positively or negatively. Let $ppre(t)$ be the set of roles used as positive preconditions in a can_assign rule t. Let $posPre = \{\langle p, r\rangle \mid \exists t \in can\_assign : p \in ppre(t) \wedge target(t) = r\}$. Define Rel_+ ("relevant positive roles") by $Rel_+ = \{p \mid \langle p, r\rangle \in posPre^* \wedge r \in goal\}$. Note that Rel_+ contains every role r such that adding r to the state might be useful in reaching the goal. Roles that appear negatively in the preconditions of can_assign rules whose target is in Rel_+ are also relevant, so we define Rel_- to contain those roles.

The sliced problem instance is obtained by deleting roles not in $Rel_+ \cup Rel_-$, deleting can_assign rules whose target is not in Rel_+, and deleting can_revoke rules whose target is not in Rel_-. Note that slicing can turn a negative role into a non-negative role, increasing the benefit of the reduction.

Example. Consider the ARBAC policy

$$\begin{aligned} can\_assign &= \{\langle r_1, r_2\rangle, \langle r_2, r_3\rangle, \langle r_3 \wedge \neg r_4, r_5\rangle, \\ &\quad \langle r_5, r_6\rangle, \langle \neg r_2, r_7\rangle, \langle r_7, r_8\rangle\} \qquad (1) \\ can\_revoke &= \{r_1, r_2, r_3, r_5, r_6, r_7\} \qquad (2) \end{aligned}$$

Consider the reachability problem for this policy with initial state $UA_0 = \{r_1, r_4, r_7\}$ and $goal = \{r_6\}$. The goal is not reachable from the initial state. Figure 3 describes the sets of reachable states and transitions generated using four variants of forward search, obtained by independently turning reduction and slicing on and off. For this policy, $NR = \{r_2, r_4\}$.

Total Revocation and State Merging. Total Revocation holds for a problem instance if every role that can be assigned by one of the administrators can also be revoked by one of the administrators, *i.e.*, $(\forall \langle c, r\rangle \in can\_assign : r \in can\_revoke)$.[3] In most ARBAC policies, an administrator who can assign users to a role can also revoke users from that role, and vice versa. Thus, in practice, most problem instances satisfy total revocation. For those problem instances, state merging can be used to optimize the above algorithm. Specifically, two reachable states s_1 and s_2 can be merged during the search if $(s_1 \cap NR) = (s_2 \cap NR)$ and $(s_1 \cap NR) \supseteq (UA_0 \cap NR)$, because a plan that reaches $s_1 \cup s_2$ from γ_0 can be constructed by concatenating (1) a plan that reaches s_1 from γ_0, (2) revocations of roles in $((s_1 \cap NR_I) \setminus UA_0)$, and (3) a plan that reaches s_2 from γ_0. A corollary of this result is: for problem instances I that satisfy total revocation and $UA_0 \cap NR_I$, two reachable states s_1 and s_2 can be merged if $s_1 \cap NR = s_2 \cap NR$. This implies that, for such problem instances, the state graph has at most $2^{|NR|}$ nodes, *i.e.*, the function f in Theorem 2 is $f(x) = 2^x$ (in general, f could be larger than exponential).

[3]This simple definition is suitable when the separate administration restriction or hierarchical role assignment (*cf.* Section 5) holds.

4. FIXED-PARAMETER TRACTABILITY OF REACHABILITY WITH ONE POSITIVE PRECONDITION UNDER SEPARATE ADMINISTRATION

The "one positive precondition" restriction, denoted $|ppre| \leq 1$, means that the precondition of each can_assign rule contains at most one positive literal. This section considers policy analysis under the $|ppre| \leq 1$ and separate administration restrictions.

Sasturkar *et al.* showed that reachability for policies satisfying $|ppre| \leq 1$, separate administration, $\overline{CR}$ (all roles can be unconditionally revoked), and $\overline{EN}$ (no explicit negation, *i.e.*, negation is used only in the form of SMER constraints) is fixed-parameter polynomial with respect to the goal size [27].

We generalize that result by eliminating the restrictions on revocation and negation. This leads to the result that reachability for policies satisfying $|ppre| \leq 1$ is fixed-parameter $|Irrev|$-tractable and $|goal|$-polynomial, where $Irrev$ is the set of irrevocable roles. If we allow those parameters to vary, the worst-case running time is exponential in the goal size, and doubly exponential in the number of irrevocable roles. We believe that the algorithm is practical nevertheless, primarily because both parameters are very small (two or less) in all natural examples we have considered so far. Also, in experiments with synthetic examples with more irrevocable roles (see Section 8), the measured running time increases only modestly with $|Irrev|$; we expect that the worst-case doubly-exponential behavior occurs only in contrived examples.

Let $I = (\gamma_0, \psi, goal)$ be a problem instance satisfying $|ppre| \leq 1$, where $\gamma_0 = \langle R, UA_0\rangle$ and $\psi = \langle can\_assign, can\_revoke\rangle$. Because $|ppre| \leq 1$, each element of can_assign can be written in the form $\langle p \wedge \neg N, r\rangle$, where p is a positive literal (*i.e.*, a role) or $true$, and N is a (possibly empty) set of roles; $\neg\{n_1, n_2, \ldots\}$ abbreviates $\neg n_1 \wedge \neg n_2 \wedge \cdots$. Let $Irrev$ be the set of irrevocable roles, *i.e.*, $Irrev = R \setminus can\_revoke$.

The algorithm has two stages.

The first stage uses backwards search from the goal to construct a directed graph $G = (V, E)$. The nodes correspond to states (*i.e.*, sets of roles). The graph contains an edge from a state UA_1 to UA_2 if there is a can_assign rule $(p \wedge \neg N, r)$ such that, starting from UA_1, revoking roles in UA_1 that appear in N (we say that those roles "conflict" with the rule), and then adding r using this rule leads to UA_2. Given UA_2, if such a predecessor state UA_1 exists, then we say that the rule is backwards enabled in UA_2. The predecessor function and backwards enabled function are defined as follows.

$$\begin{aligned} &\text{pred}(\langle p \wedge \neg N, r\rangle, UA) \qquad (3) \\ &\quad = (p = true)\ ?\ UA \setminus \{r\} :\ (UA \setminus \{r\}) \cup \{p\} \\ &\text{backEnab}(\langle p \wedge \neg N, r\rangle, UA) \qquad (4) \\ &\quad = r \in UA \wedge \text{pred}(\langle p \wedge \neg N, r\rangle, UA) \cap N = \emptyset \end{aligned}$$

The graph is defined to be the least fixed-point of the following rules. The graph is computed by a straightforward workset algorithm.

$$\begin{aligned} &goal \in V \\ &(\forall\ UA_2 \in V, t \in can\_assign : \text{backEnab}(t, UA_2) \\ &\quad \Rightarrow e \in E\ \wedge\ label(e) = t \text{ where } e = \langle \text{pred}(t, UA_2), UA_2\rangle) \\ &(\forall\ \langle UA_1, UA_2\rangle \in E : UA_1 \in V) \qquad (5) \end{aligned}$$

The second stage of the algorithm uses the graph $\langle V, E\rangle$ to determine plan existence and, if the goal is reachable, produce a plan. The plan corresponds to a path in the graph from an initial node to the goal. However, each state encountered during the plan is

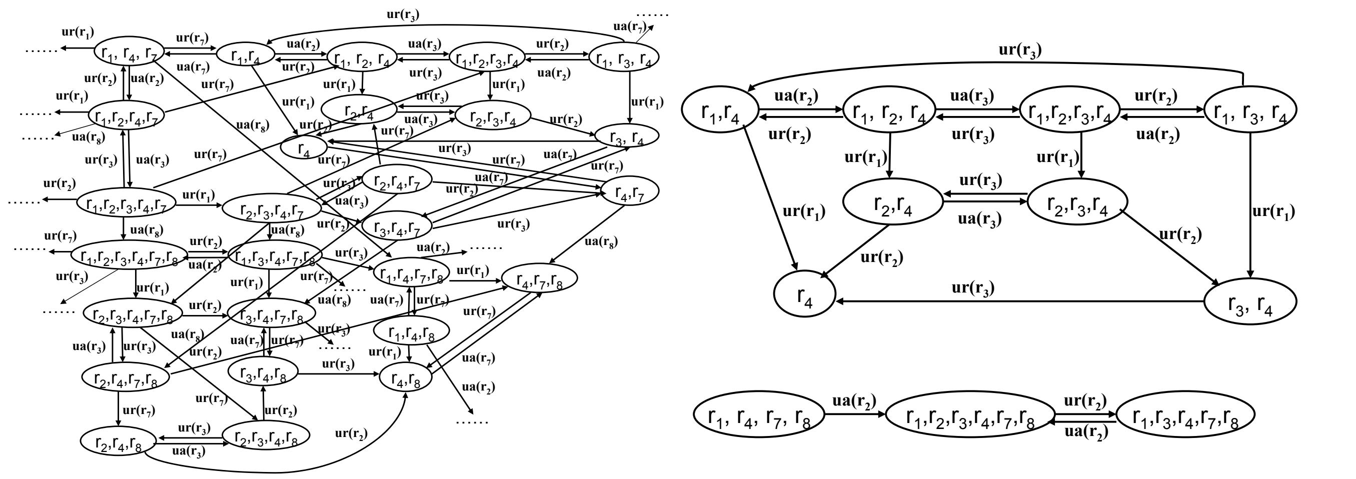

Figure 1: Left: Part of state space generated without reduction and without slicing (32 states, 96 transitions). Top right: State space generated without reduction and with slicing (8 states, 16 transitions). Bottom right: State space generated with reduction and without slicing (3 states, 3 transitions). ua **and** ur **abbreviate** $UserAssign$ **and** $UserRevoke$**, respectively. The state space generated with reduction and slicing (not shown) contains 1 state, namely,** $\{r_1, r_2, r_3, r_4\}$**, and 0 transitions.**

not simply (the set of roles in) the corresponding node in the path; rather, each state encountered in the plan may be a superset of the corresponding node in the path. This is because roles that were needed to satisfy preconditions of earlier transitions in the plan might still be in the state. This possibility is unavoidable, because some of those roles might be irrevocable. In addition, our algorithm leaves revocable roles in the state unless and until they need to be revoked to enable the next transition.

Let $P = \langle e_1, e_2, \ldots, e_n\rangle$ be a path in $\langle V, E\rangle$, represented as a sequence of edges. The candidate plan corresponding to P, denoted $plan(P)$, is $A_1.A_2.\cdots.A_n$ where

- $\langle p_i \wedge \neg N_i, r_i\rangle = label(e_i)$.
- UA'_i, the intermediate state in the plan immediately before execution of A_i, is defined by (1) UA'_1 is the source node of e_1, and (2) for $i \geq 1$, $UA'_{i+1} = UA'_i \setminus revoke_i \cup \{r_i\}$. Note that these intermediate states in the plan may be supersets of the corresponding nodes in the path, as discussed above.
- $revoke_i = N_i \cap UA'_i$ (*i.e.*, the set $revoke_i$ of roles that need to be revoked by A_i contains roles that are present in the current state and conflict with the next *can_assign* transition).
- $A_i = \{UserRevoke(r) : r \in revoke_i\}.UserAssign(r_i)$.

Note that A_i consists of the indicated $UserRevoke$ actions in arbitrary order, followed by the indicated $UserAssign$ action.

We call $plan(P)$ a "candidate plan" because it might attempt to revoke an irrevocable role. A path P is *feasible* if $plan(P)$ does not contain $UserRevoke(r)$ for any $r \in Irrev$.

A node UA in the graph is an *initial node* if it is a subset of the initial state UA_0. This definition allows initial revocation of revocable roles in $UA_0 \setminus UA$. This is necessary because every edge in the graph corresponds to a sequence of operations that adds at least one role to the state. Irrevocable roles in $UA_0 \setminus UA$ are placed in $airs(UA)$, defined below.

LEMMA 3. *There is a plan for I iff there is a feasible path P from an initial node to the goal node.*

PROOF: This follows directly from the construction of the graph and the above definitions. □

To determine whether a feasible path exists, we compute, for each node UA in the graph, the set $airs(UA)$ of sets of additional irrevocable roles that can be in states corresponding to that node; "additional" here means "not in UA". More precisely, $S \in airs(UA)$ iff (1) UA is an initial node and $S = (UA_0 \cap Irrev) \setminus UA$, or (2) $S \subseteq Irrev$ and $S \cap UA = \emptyset$ and there exists a feasible path P from an initial node UA_1 to UA such that execution of $plan(P)$ from UA_1 leads to the state $UA \cup S$.

LEMMA 4. *There is a plan for I iff* $airs(goal)$ *is non-empty.*

PROOF: This is a corollary of the previous lemma, and the observation that (by definition of $airs$) $airs(goal)$ is non-empty iff there exists a feasible path from an initial node to $goal$. Note that $airs(goal)$ might contain only the empty set. That counts! It implies there exists a feasible path P from an initial node to $goal$ such that $plan(P)$ adds no additional irrevocable roles. □

Considering every path individually would be very expensive, so we introduce an alternate characterization of $airs$ that leads to a more efficient algorithm. Specifically, $airs$ is the least (with respect to the pointwise extension of the subset ordering) solution of the following set inclusion constraints, where the set comprehension $\{f(x_1, x_2, \ldots) : x_1 \in S_1, x_2 \in S_2, \ldots \mid p(x_1, x_2, \ldots)\}$ denotes the set obtained by iterating over each combination of values $x_1, x_2, \ldots$ in $S_1 \times S_2 \times \cdots$ and, if $p(x_1, x_2, \ldots)$ holds, adding $f(x_1, x_2, \ldots)$ to the result set.

- For each initial node UA, $airs(UA) \supseteq (UA_0 \cap Irrev) \setminus UA$.
- For each edge $UA_1 \overset{\langle p\wedge\neg N, r\rangle}{\longrightarrow} UA_2$, if UA_1 is reachable with additional irrevocable roles S in the state, and if this edge is a feasible transition from that state (*i.e.*, if $((UA_1 \cap Irrev) \cup S) \cap N = \emptyset$), then UA_2 is reachable with additional irrevocable roles $S \cup ((UA_1 \setminus UA_2) \cap Irrev)$ (in other words, irrevocable roles present in UA_1 must still be present in the next state and, if they do not appear in UA_2, are "additional" in that state, by definition); formally,

  $airs(UA_2) \supseteq \{S \cup ((UA_1 \setminus UA_2) \cap Irrev) :$
  $S \in airs(UA_1) \mid ((UA_1 \cap Irrev) \cup S) \cap N = \emptyset\}$

  The role r added along this edge does not appear in the set constraint, because the construction of the graph ensures that r appears in UA_2 and hence is never "additional" in states corresponding to UA_2. As an optimization, $((UA_1 \cap Irrev) \cup S) \cap N$ can be simplified to $S \cap N$, because the construction of the graph implies $UA_1 \cap N = \emptyset$. Similarly, the construction implies that $(UA_1 \setminus UA_2)$ equals $\{p\}$ if $p \neq true$ and equals $\emptyset$ otherwise.

These constraints can be rewritten as a monotonic recursive definition.

$$\begin{aligned} airs(UA_2) = {} & (UA_2 \in initialNodes\ ?\ \{\emptyset\} : \emptyset) \cup \\ & \{S \cup ((UA_1 \setminus UA_2) \cap Irrev)\ : \\ & \quad UA_1 \xrightarrow{\langle N,r\rangle} UA_2 \in inedges(UA_2), \\ & \quad S \in airs(UA_1) \mid ((UA_1 \cap Irrev) \cup S) \cap N = \emptyset\} \end{aligned}$$

The solution can be computed by a straightforward fixed-point computation, using a workset algorithm. Existence of a plan for I is then determined using Lemma 4. The fixed-point computation can easily be augmented to store additional information that provides a plan for I, if a plan for I exists.

Now we analyze the algorithm's time complexity. Let $g = |goal|$. Each node in V contains at most g roles, because the search starts with the state $goal$, and the definition of UA_1 ensures that $|UA_1| \leq |UA_2|$. There are $\binom{|R|}{g}$ sets containing exactly g roles, and each of these sets has 2^g subsets, so $|V|$ is $O(2^g\binom{|R|}{g})$, which is $O(2^g|R|^g)$. $|E|$ is $O(|V|^2)$ which is $O(2^{2g}|R|^{2g})$. Processing each possible node or edge takes $O(|I|)$ time, so the running time of Stage 1 is $O(|E||I|)$. In Stage 2, each node in V gets labeled with a set of subsets of $Irrev$, and propagating each subset along each edge takes $O(|I|)$ time, so the running time of Stage 2 is $O(|E|2^{2^{|Irrev|}}|I|)$. Thus, the overall running time is $O(2^{2g}|R|^{2g}2^{2^{|Irrev|}}|I|)$. This shows that reachability for problem instances with $|ppre| \leq 1$ is fixed-parameter $|Irrev|$-tractable and $|goal|$-polynomial.

This algorithm and complexity result can be extended to handle a class of policies that do not satisfy $|ppre| \leq 1$. Extending the algorithm is easy: interpret p as a set of positive preconditions, instead of a single positive precondition, and in the definition (3) of pred, replace "$\cup \{p\}$" with "$\cup\, p$". The consequence is that generated states may contain more than $|goal|$ roles. A policy is *cycle free* if the *posPre* relation (defined in Section 3) is acyclic. Consider any path P in the generated graph from a state UA_1 to a state UA_2, for a cycle free policy. Since the policy is cycle free, the number of edges in P labeled by a selected rule t is bounded by $|UA_2|$. Note that $|UA_1|$ is larger than $|UA_2|$ due to uses of t by at most $|ppre(t)| \times |UA_2|$. Thus, the maximum size of a state in the graph is bounded (loosely) by $\pi \times |goal|$, where $\pi = \Pi_{t \in can\_assign} \max(1, |ppre(t)|)$. The complexity analysis proceeds as above, except with g replaced with $\pi \times |goal|$. This shows that the extended algorithm is fixed-parameter $|Irrev|$-tractable and $(\pi \times |goal|)$-polynomial, for cycle free policies. Note that $\pi = 1$ when $|ppre| \leq 1$, so this result specializes to the previous result, although only for cycle-free policies.

Partial-Order Reduction. The graph construction can be optimized with a partial-order reduction [9, 4]. It is trickier than in Section 3, because the graph construction is a backward search, and the states corresponding to nodes in the graph are only partly determined during the backward search. A straightforward adaptation of the reduction in Section 3 is unsound. That reduction executes transitions whose target is a non-negative role as soon as they are enabled, hence those transitions appear early in the plan. A straightforward adaptation of it for backward search is to defer processing of such transitions whenever other transitions are backwards enabled; this also causes such transitions to appear early in the generated plan. However, this is unsound: it sometimes defers such transitions too much. For example, consider a reachability problem instance with $can\_assign = \{\langle r_0, r_1\rangle, \langle \neg r_0, r_2\rangle, \langle true, r_0\rangle, \langle \neg r_2, r_3\rangle\}$, $can\_revoke = \emptyset$, $UA_0 = \emptyset$, and $goal = \{r_1, r_2\}$. The technique proposed above defers processing of rules whose target is the non-negative role r_1, so only the rule that adds r_2 would be explored backwards from the goal state. Therefore, the algorithm would fail to find the plan for this problem instance, namely, the plan that adds r_2, then r_0, and then r_1.

To avoid this problem, our reduction takes a different approach: it identifies transitions that can be processed eagerly during the backward search, causing them to appear late in the resulting plan. A role r is *backwards invisible* in a state UA if every transition $t = \langle p \wedge \neg N, r\rangle$ with target r satisfies $(p = true \vee p \in UA \vee p \notin NR_I) \wedge N \cap Irrev = \emptyset$, and at least one transition with target r is backwards enabled in UA_2. The first conjunct in the formula ensures that backward execution from UA of a transition with target r does not disable backwards-enabled transitions with other targets (so the backward algorithm can process those transitions after processing a transition for r). The second conjunct in the formula ensures that transitions for r will not be disabled by irrevocable roles added to the state in stage 2.

To incorporate the reduction into the graph construction, modify the definition of E so that, if some role r in UA_2 is backward invisible in UA_2 and the stack proviso is satisfied (see below), then only can_assign rules for r are explored from UA_2, otherwise all backward enabled can_assign rules are explored from UA_2. The stack proviso ensures that no transitions are completely ignored even when the state space contains cycles; it is satisfied if at least one transition backward explored from the current state leads to a state not on the DFS search stack [9, Chapter 6]. Note that slicing can increase the benefit of this reduction, by turning negative roles into non-negative roles.

This reduction is not a special case of traditional partial-order reductions [9, 4], because it uses an extra condition to deal with the fact that the search is split into two stages, and because it exploits the fact that the eagerly executed transitions are left-movers, while traditional partial-order algorithms exploit only full commutativity (independence) of transitions.

5. BEYOND THE SEPARATE ADMINISTRATION RESTRICTION

This section considers policy analysis without the separate administration restriction. First, we generalize the reduction-based algorithm in Section 3 to eliminate its dependence on this restriction. Second, we identify a condition under which policy analysis without the separate administration assumption can be reduced to policy analysis with the separate administration assumption.

5.1 Fixed-Parameter Tractability of Reachability

Without the separate administration restriction, reachability analysis must consider plans that may contain administrative actions that change the role memberships of any user in U_0, not only the target user. To accommodate this, we describe how to generalize the reduction-based algorithm in Section 3 to track the role sets of multiple users. The worst-case time complexity is exponential in the number of those users. This demonstrates that reachability analysis is fixed-parameter tractable with respect to the number of negative roles and $|U_0|$.

Generalizing the partial-order algorithm in this way is straightforward. We now deal with full user assignments $UA \subseteq U \times R$, not simplified user assignments $UA \subseteq R$. We strengthen the definition of enabledness of an action in a state γ to require (in addition to the conditions in Section 2.2) that the administrative role r_a (the first argument to the $UserAssign$ or $UserRevoke$ action) is a role of some user in U_0 in γ. The definitions of visible and invisible transitions and closure are unchanged. The reduced transition relation is defined by: $\gamma_1 \overset{a}{\rightsquigarrow} \gamma_2$ iff there exists a state γ such that $\gamma_1 \xrightarrow{a} \gamma$ and $\gamma_2 = \text{closure}(\gamma)$ and $\gamma_1 \neq \gamma_2$ and a is $UserAssign(r_a, u, r)$ or $UserRevoke(r_a, u, r)$ for some administrative role r_a, some user u in U_0, and some negative role r. Theorem 1 still holds, provided we replace $goal \subseteq \gamma$ with $goal \subseteq \gamma(u_t)$, where $\gamma(u) = \{r \mid \langle u, r\rangle \in \gamma\}$.

The proof of fixed-parameter tractability of this algorithm with respect to the number of negative roles and $|U_0|$ is analogous to the proof of Theorem 2, except that each state in G_{NR} is now a $|U_0|$-tuple of subsets of NR, so the size of G_{NR} is bounded by a function of and $|NR|$ and $|U_0|$, so by the same argument as before, the size of G_{red} is also bounded by some function of $|NR|$ and $|U_0|$, *i.e.*, the size of G_{red} is $O(f(|NR_I|, |U_0|))$ for some function f. It follows that the worst-case time complexity of constructing the reduced state graph is $O(f(|NR|, |U_0|)|I|^c)$, for some function f and some constant c.

5.2 Hierarchical Role Assignment

We say that an administrative role r is "available" in a state if some user in U_0 is a member of r in that state.

The separate administration restriction simplifies policy analysis primarily because it ensures that the set of available administrative roles does not change. Here, we identify a condition under which changes to the set of available administrative roles, although possible, are not useful (for reaching a goal) and hence can be ignored.

The condition ensures that, in the initial state, the users in U_0 already have all of the administrative permissions of administrative roles to which they could assign themselves. This is achieved by requiring that the users in U_0 are implicit or explicit members of all those administrative roles in the initial state.

A miniARBAC policy has *hierarchical role assignment* with respect to a set A of administrative roles if: for all $\langle ar, c, r\rangle$ in can_assign, if ar is in A and r is an administrative role,[4] then $ar \succeq r$. This implies that a user who can assign users to r is an implicit member of r. We call this property "hierarchical role assignment" because it relates the role hierarchy with the can_assign relation.

THEOREM 5. *Let $I = \langle \gamma_0, \psi, U_0, u_t, goal\rangle$ be a reachability problem instance. Let A be the available administrative roles in γ_0. If the miniARBAC policy ψ has hierarchical role assignment with respect to A, then the goal is reachable iff the goal is reachable via a plan in which all assign and revoke actions act on the target user.*

PROOF:

It is not useful for a user u_1 in U_0 to assign a user u_2 other than u_t in U_0 to an administrative role ar, because u_1 is already an implicit member of ar, and making another user a member of r provides no additional administrative permissions to the group U_0 of users. Note that we allow assignment of administrative roles to u_t, because such assignments can truthify the precondition of a can_assign rule, and this might enable addition of u_t to a regular role in the goal. It is not useful to assign a user u_2 other than u_t to a regular role, or to revoke u_2 from any role, because the only potential benefit of such administrative actions would be to truthify the precondition of a rule allowing u_2 to be assigned to an administrative role, and we already showed that the latter role assignment would be useless. □

[4]Recall that the concept of administrative role is well-defined even without the separate administration restriction.

Our analysis algorithms in Sections 3 and 4 exploit separate administration only to avoid considering administrative actions that act on users other than the target user. Therefore, Theorem 5 implies that those algorithms work correctly for problem instances that satisfy hierarchical role assignment.

The same idea can be used to optimize reachability analysis for problem instances I with "partially hierarchical" role assignment, *i.e.*, when I satisfies hierarchical role assignment for a subset A' of the set A of available administrative roles in the initial state. Administrative actions of roles in $A \setminus A'$ might be useful in reaching the goal. By starting with those actions, and adding additional actions based on the dependencies induced by the can_assign relation, we can identify a set of administrative actions on non-target users that might be useful in reaching the goal; other administrative actions on non-target users can be eliminated.

6. OTHER ANALYSIS PROBLEMS

This section presents algorithms for some other analysis problems. These algorithms use algorithms for user-role reachability as a subroutine, so our results from Sections 3–5 are useful here. These algorithms can invoke specialized algorithms for reachability when the latter algorithm's restrictions are satisfied (*e.g.*, $|ppre| \leq 1$). It is straightforward to obtain fixed-parameter tractability results for these algorithms, based on our complexity results for user-role reachability.

Role containment. Role containment problem instances have the form [20]: in every state reachable from a given initial state, is every member of role r_1 also a member of role r_2? This problem can be reduced to user-role reachability by adding a new role r and adding a can_assign rule with precondition $r_1 \wedge \neg r_2$ and target r. The containment property holds iff, for every user u, there is not a reachable state in which u is a member of r. As an optimization, only users with distinct sets of initial roles need to be considered.

Weakest Preconditions. Weakest precondition queries return the minimal sets of initial role memberships of the target user for which a given reachability goal is achievable (for these queries, the initial state does not specify the initial roles of the target user). An example query is: what are the weakest preconditions for a user initially in DeptChair to assign the target user to HonorsStudent? For policies satisfying the conditions of the backward algorithm in Section 4, that algorithm can be modified to efficiently answer such queries. For each leaf node UA in the graph, we compute $airs(goal)$ taking UA as the only initial node, and if $airs(goal)$ is non-empty, then UA is a weakest precondition, unless we find another one that is a subset of UA.

7. CASE STUDIES

This section briefly describes the ARBAC policies that we used as case studies. Details of both policies are available from [31].

Our main case study is an ARBAC policy for selected aspects of a university. The policy includes rules for assignment of users to various student and employee roles. Student roles are undergraduate student, graduate student, teaching assistant, research assistant, grader, honors student, graduate student officer, graduate education committee (which has a student member), *etc*. Employee roles are admissions officer, assistant professor, dean, dean of admissions, department chair, facilities committee, graduate admissions committee, graduate education committee, honors program director, president, professor, provost, *etc*. The role hierarchy includes the relationships $\mathrm{President} \succeq \mathrm{Provost} \succeq \mathrm{Dean} \succeq \mathrm{DeptChair} \succeq \mathrm{Professor}$. Sample can_assign rules are: the honors program director can add undergraduates to the honors student role; the president can assign a professor who is not a department chair to the provost role.

The policy has some limitations. Our ARBAC framework does not support parameters (of role, permissions, *etc.*), so the policy is for a single department, a single class, *etc.* With a framework that does, we could easily add parameters (such as department name and class number) to the policy. We could analyze the resulting policy by instantiating the parameters with appropriate sets of values and then applying our current analysis algorithms; we plan to develop algorithms that handle parameters directly for efficiency. The current policy is only for user-role assignment; this seems like a good place to start, since the user assignment changes more often. Policies for administration of the permission-role assignment and the role hierarchy will be added later. Despite these limitations, this ARBAC policy is a substantial case study compared to others in the literature, as discussed in Section 9.

These limitations do not significantly affect the characteristics of the policy that guided this work, such as the size of preconditions. Experience with this policy guided our choice of complexity parameters and restrictions for the analysis algorithms in Sections 3–4 and suggested realistic values for some of the parameters used in random policy generation in Section 8. In particular, we note the following characteristics of the policy:

- Every can_assign rule has at most one positive precondition, so the analysis algorithm in Section 3 applies.
- The policy does not satisfy separate administration, but it has hierarchical role assignment with respect to most sets of administrative roles, so most reachability properties of the policy can be analyzed using the algorithms in Sections 3 and 4; the others can be analyzed using the algorithms in Section 5.
- About 1/3 of the roles are administrative (*i.e.*, have at least one administrative permission). After the transformation to eliminate role hierarchy (*cf.* first paragraph of Section 3), about 1/4 of the roles are negative, about 1/4 are mixed, and about 2/3 are positive. Since about 3/4 of the roles are non-negative, the reduction in Section 3 should be effective.
- Problem instances with hierarchical role assignment have at most two irrevocable roles, because admissions officers and the graduate admissions committee can accept but not expel students.

Formulating and checking properties of the policy helped uncover some flaws in it, for example, a place where we accidentally used Student instead of Undergrad, and places where we forgot to take role hierarchy into account, *e.g.*, places where we were thinking of Provost and President as Faculty only, forgetting that these roles also inherit (indirectly) from Staff.

Here are sample user-role reachability problem instances (with answers!) for this policy. Can a user initially in DeptChair and a user initially in Undergrad reach a state in which the latter user is in HonorsStudent? Yes, because the user in DeptChair can assign himself to HonorsProgramDirector, and then assign the undergrad to HonorsStudent. Can a user initially in Provost and a user initially in DeptChair reach a state in which latter user is in Dean? No, because the rule for assignment to Dean has precondition $\mathrm{Professor} \wedge \neg\mathrm{DeptChair}$. The Provost can remove users from DeptChair, but the Provost cannot add users to Professor (only the President can do that).

A sample role containment problem instance is: Is TA (*i.e.*, teaching assistant) contained in Grad (*i.e.*, graduate student)? No. Although Grad is a precondition for assignment to TA, a user can be revoked from Grad while remaining in TA. (This example illustrates that preconditions are not necessarily invariants.)

Our second case study is an ARBAC policy for a health care institution, based on the policy in [7], extended with some aspects of the policy in [3]. This case study is smaller, but we note that it shares most of the above characteristics of the university policy. In particular, every can_assign rule has at most one positive precondition, separate administration is not satisfied, about 1/3 of roles are negative, and problem instances with hierarchical role assignment have at most one irrevocable role. One difference is less hierarchical role assignment (*i.e.*, there are more sets of administrative roles for which the policy does not have hierarchical role assignment).

8. EXPERIMENTAL RESULTS

This section presents the results of experiments done to evaluate the performance of the forward and backward reachability algorithms. The forward algorithm always uses the reduced transition relation; the backward algorithm includes the partial-order reduction only when stated explicitly. The algorithms were applied to the case studies in Section 7 and randomly generated ARBAC policies. Our algorithm for random policy generation has several parameters, allowing control over the number of roles, percentage of negative roles, percentage of irrevocable roles, average number of positive and negative preconditions per rule, average number of rules per target role, *etc.* Generally, we used parameter values (*e.g.*, for number of roles) and distributions of values (*e.g.*, for number of rules per target role) similar to those in the university policy as a baseline, and then varied selected parameters to explore the effect. Each data point reported for randomly generated policies is an average over 32 policies generated using the same parameter values. In many cases, the standard deviation is comparable in magnitude to the average; this suggests that statistical fluctuations cause the observed aberrations in the trends (*e.g.*, there are fewer states and transitions for $|R| = 300$ than $|R| = 200$ in Table 1(b)) and suggests seeking additional parameters to control random policy generation more tightly. Running times were measured on a 2.8 GHz Pentium D with 1 GB RAM running Linux 2.6.20. All reported running times are measured in seconds.

Case Studies. Both policies satisfy the restriction $|ppre| \leq 1$ and hence both forward and backward reachability algorithms are applicable for queries satisfying hierarchical role assignment. For a variety of reachability queries with $|goal| \leq 2$, forward with slicing terminates in at most 0.01sec and backward with reduction terminates in at most 0.19sec. For each goal, the forward algorithm generates at most 2 states and 1 transition, and the backward algorithm with reduction generates at most 5 nodes and 4 transitions.

Evaluation of the Forward Algorithm. Table 1(a) shows the number of explored states, number of explored transitions, and running time of the forward algorithm (without slicing) on randomly generated policies with varying number of mixed roles and with other parameters held constant. All three cost metrics grow quickly as a function of the number of mixed roles. Table 1(b) shows the performance of the forward algorithm (without slicing) on randomly generated policies where the number of roles varies (the number of rules varies with it, since average number of rules per role is held constant), and the number of mixed roles is held constant. In this case, the cost grows much more slowly.

Slicing significantly improves the typical performance of the forward algorithm, although it does not change the worst-case perfor-

Mixed	State	Trans	Time
3	8	18	0.07
5	33	122	0.32
7	187	960	2.60
9	350	2238	5.62
11	1404	10699	24.7
13	5593	50511	128.10

(a)

Roles	State	Trans	time
100	42	152	1.09
200	54	193	4.42
300	41	159	5.40
400	57	218	13.65
500	74	266	17.63

(b)

Table 1: (a) Running time of forward algorithm vs. number of mixed roles with $|R| = 32$. (b) Running time of forward algorithm vs. number of roles with 5 mixed roles.

$\|goal\|$	Nodes	Trans	Time
1	30	125	0.01
2	377	3022	0.14
3	2128	23744	2.59
4	14395	215396	97.95

(a)

Roles	Nodes	Trans	Time
100	78	360	0.02
200	124	539	0.03
300	242	1095	0.10
400	329	1459	0.13
500	401	1789	0.25

(b)

Table 2: Performance of backward algorithm for (a) varying goal size and (b) varying number of roles with $|Irrev|/|R| = 0.05$ and $|goal| = 1$.

mance. For most policies, the algorithm explored just one or two states after slicing, terminating within 0.04 sec.

Evaluation of the Backward Algorithm. We evaluated the backward algorithm on randomly generated policies satisfying $|ppre| \leq 1$. Table 2(a) shows that the analysis cost grows quickly as a function of the size of the goal when other parameters are held constant. Table 2(b) shows that the analysis cost grows very slowly as a function of the number of roles (and rules), when the percentage of irrevocable roles and the goal size are held constant (at 5% and 1, respectively).

The reduction technique for backward algorithm reduces the state space and the running time of policies used in Table 2(a) by 24% and 19% on the average, respectively. It does not affect the state space and running time of policies used in Table 2(b).

Forward Algorithm with Slicing vs. Backward Algorithm. We applied the forward algorithm with slicing to the same policies used for the experiments reported in Table 2(a). The average execution times were 0.04 sec, 0.38 sec, 0.87 sec and 1.08 sec when $|goal|$ is 1, 2, 3, and 4, respectively. Observe that the average execution time for the forward algorithm increases slightly with $|goal|$, while the corresponding increase in the execution time for the backward algorithm is much more significant.

Note that slicing does not change the worst-case complexity of the forward algorithm. When $|goal| = 1$ and $|Irrev| \leq 1$, the backward algorithm has better time complexity than the forward algorithm, except when $|NR| = 0$ and both algorithms have similar (polynomial) complexity. For a set of randomly-generated policies with $|goal| = 1$, $|Irrev| = 2$, $|R| = 50$, and $|NR|$ varying between $0.6|R|$ and $0.9|R|$, the backward algorithm is 11 to 30 times faster than the forward algorithm. We observe that when $|goal|$ and $|Irrev|$ are small and fixed, the backward algorithm is superior to the forward algorithm in terms of analysis time and the size of the explored state space.

9. RELATED WORK

Policy Analysis. We classify related work on security policy analysis into three categories.

The first, and largest, category is analysis (including enforcement) of a fixed security policy. Some representative papers in this category include [15, 2, 11, 16, 10, 13]. Work in this category is less closely related to our work, so we do not discuss it further.

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 present algorithms to analyze the effects of a specified change to a SPKI/SDSI policy [16]. Fisler *et al.* [8] give algorithms to compute the semantic difference of two XACML 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 sequences of changes to a policy; the allowed changes are determined by parts of the policy that we call "administrative policy". Harrison, Ruzzo, and Ullman [12] present an access control model based on access matrices, which can express administrative policy, and show that the safety analysis problem is undecidable for that model. [5] Following this, a number of access control models were designed in which safety analysis is more tractable, *e.g.*, [22, 24]. While those models were designed mainly with tractability in mind, we aim to provide more practical results, by starting with more a realistic model, based on ARBAC97 [25], and identifying properties of typical policies that can be exploited for efficient analysis. Our framework allows features not considered in those papers, such as negative preconditions.

Finally, we focus on prior work on analysis of ARBAC policies.

Schaad and Moffett [29] use the Alloy analyzer [14] to check separation of duty properties for ARBAC97. They do not consider preconditions for any operations; this greatly simplifies the analysis problem. Since they leave the analysis to the Alloy analyzer, they do not present analysis algorithms or complexity results.

Li and Tripunitara [20] give algorithms and complexity results for various analysis problems—primarily safety, availability, and containment—for two restricted versions of ARBAC97, called AATU and AAR. Their results are based on Li, Mitchell, and Winsborough's results for analysis of trust management policies [19]. Our work goes significantly beyond their analyses for both AATU and AAR by allowing negative preconditions and thereby SMER (static mutually exclusive roles) constraints. This forces us to consider other (more realistic) restrictions, such as bounds on the size of preconditions, and to use fixed-parameter tractability to characterize the complexity of our algorithms. In addition, our work in Section 5 goes significantly beyond their analysis for AAR by dropping the separate administration restriction. Sistla and Zhou [30], like [19], consider trust management policies changing in accordance with *role restrictions* that indicate, for each role, whether arbitrary rules defining that role may be added, and whether they may be removed. The administrative policies we consider are finer-grained than such role restrictions.

Sasturkar *et al.* [27] present algorithms and complexity results for analysis of ARBAC policies subject to a variety of restrictions. Our work goes beyond theirs by providing efficient algorithms for larger and more realistic classes of policies, providing fixed-parameter tractability results to more accurately characterize the complexity of those algorithms, and giving analysis algorithms that do not rely on the the separate administration restriction, which is implicitly adopted throughout their paper. Also, they do not consider containment analysis.

Case Studies. Our ARBAC policy for a university contains significantly more can_assign rules than the ARBAC policies presented

[5]That result does not apply to ARBAC policy analysis, because the HRU model allows creation of subjects and objects, while ARBAC does not allow creation of users, roles, or permissions.

in [25, 28, 23, 29, 17, 5, 27, 20, 18], which typically contain about 4 administrative roles and the equivalent of 4 to 7 *can_assign* rules.[6] Some papers, such as [28, 17], sketch the general structure of RBAC and ARBAC policies of very large organizations, but only a few specific administrative roles and rules are presented in the paper (or otherwise made publicly available), and no analysis algorithms were applied to those policies. We analyzed our ARBAC policy for a university. The policy contains 11 administrative roles, 21 other roles, 28 *can_assign* rules (106 rules after the transformation to eliminate role hierarchy), *etc*.

[6]In some cases, a few rules are duplicated, *e.g.*, copied with change only to the name of the department; we did not count the duplicates, since we did not include such duplicates in our university policy. Also, we did not count other kinds of rules, *e.g.*, *can_assignp*; there are only a few of those, too.

10. REFERENCES

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

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

[3] M. Y. Becker. *Cassandra: Flexible Trust Management and its Application to Electronic Health Records*. PhD thesis, University of Cambridge, Oct. 2005.

[4] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[5] J. Crampton. Understanding and developing role-based administrative models. In *Proc. 12th ACM Conference on Computer and Communications Security (CCS)*, pages 158–167. ACM Press, 2005.

[6] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[7] M. Evered and S. Bögeholz. A case study in access control requirements for a health information system. In *Proc. Australasian Information Security Workshop 2004 (AISW)*, volume 32 of *Conferences in Research and Practice in Information Technology*, 2004.

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

[9] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[10] J. D. Guttman, A. L. Herzog, J. D. Ramsdell, and C. W. Skorupka. Verifying information flow goals in Security-Enhanced Linux. *Journal of Computer Security*, 13(1):115–134, 2005.

[11] J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 187–201. IEEE Computer Society Press, 2003.

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

[13] K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *In Proc. 13th ACM Conference on Computer and Communications Security (CCS)*, pages 134–143, Nov. 2006.

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

[15] 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.

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

[17] A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. In *Proc. 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM Press, 2003.

[18] N. Li and Z. Mao. Administration in role based access control. In *Proc. ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS)*, Mar. 2007.

[19] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM*, 52(3):474–514, 2005.

[20] N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security*, 9(4):391–420, Nov. 2006.

[21] R. J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, 1975.

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

[23] S. Oh and R. S. Sandhu. A model for role administration using organization structure. In *Proc. 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM Press, 2002.

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

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

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

[27] A. Sasturkar, P. Yang, S. D. Stoller, and C. Ramakrishnan. Policy analysis for administrative role based access control. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, July 2006.

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

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

[30] A. P. Sistla and M. Zhou. Analysis of dynamic policies. In *Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA)*, Aug. 2006. Full version to appear in *Information & Computation*.

[31] `www.cs.stonybrook.edu/~stoller/ccs2007/`.