Reachability Analysis for Role-based Administration of Attributes

Xin Jin
Institute for Cyber Security
Univ of Texas at San Antonio
San Antonio, Texas, USA
dfb616@my.utsa.edu

Ram Krishnan
Institute for Cyber Security
Univ of Texas at San Antonio
San Antonio, Texas, USA
ram.krishnan@utsa.edu

Ravi Sandhu
Institute for Cyber Security
Univ of Texas at San Antonio
San Antonio, Texas, USA
ravi.sandhu@utsa.edu

ABSTRACT

Attribute-based access control (ABAC) is well-known and increasingly prevalent. Nonetheless, administration of attributes is not well-studied so far. Recently, the Generalized User-Role Assignment model (GURA) was proposed to provide ARBAC97-style (administrative role-based access control) administration of user attributes. An *attribute* is simply a name-value pair, examples of which include clearance, group and affiliations. In GURA, user attributes are collectively administered by different administrative roles to enable distributed administration. Given an administrative policy that specifies the conditions under which administrative roles can modify user attributes, it is useful to understand whether an attribute of a particular user can *reach* a specific value because user attributes are used for security-sensitive activities such as authentication, authorization and audit. In this paper, we study the user-attribute reachability problems in a restricted GURA model called rGURA. We formalize rGURA as a state transition system and show that the reachability problems for its general cases are PSPACE-complete. However, we do find polynomial-time solutions to reachability problems for limited versions of rGURA that are still useful in practice. The algorithms not only answer reachability problem but also provide a plan of sequential attribute updates by one or more administrators in order to reach particular values for user attributes. rGURA is relatively simple and practical. It is likely that other proposals will subsume the functionality of rGURA and thereby subsume its complexity results.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection—Access controls

General Terms

Security

Keywords

Attributes; Administration; Reachability Analysis

DIM'13, November 8, 2013, Berlin, Germany.

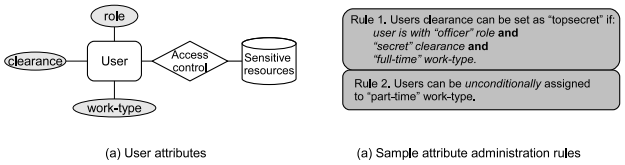
http://dx.doi.org/10.1145/2517881.2517891 .

1. INTRODUCTION

User attributes are functions which, given a particular user, return certain properties of the user. Examples are role, clearance, department, etc. User attributes are often used in sensitive activities such as authorization, authentication and audit [1, 18, 22, 23, 24, 33]. For example, in attribute-based access control (ABAC) [13, 18, 22], access decisions are made based on various user attributes compared to identity in discretionary access control (DAC) [30], clearance in mandatory access control (MAC) [28] and role in role based access control (RBAC) [29]. Generalized User-Role Assignment (GURA) model [17] was recently proposed for the purpose of attribute administration. The core idea is that permissions to modify user attributes are associated with administrative roles. Administrators are made members of these roles, thus obtaining associated permissions. GURA extends the user-role assignment (URA) component of the well-known ARBAC97 administrative model [26] by generalizing user attributes beyond role. In this paper we study a restricted version of GURA called rGURA.

A critical question regarding access control policies is whether they ensure certain security properties. In context of rGURA, as user attributes are further utilized for security-sensitive activities, it is important to ensure that every user can only be assigned appropriately valid attribute values. Although administrators might be trusted and expected to exercise due diligence in attribute assignments, it is nevertheless desirable to determine exactly what values of attributes can be assigned by a collection of administrators acting cooperatively. Such analysis can also provide guidance on the sequence of attributes modifications to achieve specific attribute assignments for specific users. It is not straightforward to understand administrative policies by simple inspection. Large number of attributes and policies and unexpected actions of administrators complicate the analysis. Take the scenario in figure 1 as an example. Figure 1(a) shows that users can only access sensitive resources when the three attributes reach certain values simultaneously. Suppose an administrative user in a manager role can assign a user to "topsecret" clearance if the user is with "officer" role, with "secret" clearance and not "part time" in work-type. It might seem that a user can never be "part time" and with "topsecret" clearance at the same time. The policy is summarized in figure 1(b). However, the policy may inadvertently allow that. A user may be "full time" and then assigned to "topsecret" clearance according to rule 1. After that, he can be assigned to "part time" work-type according to rule 2. Reachability analysis can reveal such anomalies which may not be explicitly considered or immediately obvious in policy design.

We explore attribute reachability analysis in this paper. Policy analysis, which allows policy designers to check whether their policies meet their security goals, has been recognized as important in access control [12, 20, 21, 25, 27, 31]. The closest work to this pa-



(a) User attributes (a) Sample attribute administration rules

Figure 1: *Example user attribute reachability problem*

per is role reachability analysis in ARBAC97 [11, 20, 31, 34, 36]. Since GURA is an extension to ARBAC which contains more than one attribute, it is not sufficient to directly use the results from ARBAC97 analysis. In ARBAC97 there is a single set-valued attribute called role. In rGURA on the other hand there are multiple attributes, some set-valued and some atomic-valued with possible constraints across multiple attributes.

Our main contributions are as follows. (1) The reachability problem asks whether a user can be assigned to specific attributes values by the actions of a set of administrators. We formally define user attribute reachability by abstracting the rGURA model as a state transition system. We consider different variations of queries for set-valued attributes than that in ARBAC study. In ARBAC97 analysis, a $goal$ (a set of roles) is reached if the user is assigned the roles in $goal$ or a superset of these. This is because the additional roles cannot reduce authorization in RBAC systems. With general attributes it is not clear how their values can impact authorization. Additional values in a set-valued attribute may possibly reduce authorization, depending on how the authorization policies are specified. Therefore, we also consider the case where set-valued attributes should be assigned exactly the same values as in $goal$ but not a superset. (2) By reductions from the role reachability problem in ARBAC97 and the planning problem in artificial intelligence, we show that general attribute reachability problems are PSPACE-complete, along with identified special cases. We also consider special NP-complete or NP-hard cases. (3) We identify polynomial time solvable cases which are useful in practice. As the input to the algorithms may be large and the analysis may need to be performed frequently, polynomial performance is always desirable. We further evaluate performance through simulations.

This paper focusses on user attributes. In general, attributes are also associated with other entities such as subjects, objects and environment in ABAC systems [13]. Our reachability analysis results apply to attributes of any such entity so long as they are managed using the rGURA style, that is each attribute is managed by administrative users who can modify it provided the entities' attributes satisfy specified preconditions. However, rGURA-style models may not be appropriate for attribute administration in all possible scenarios. For instance, one may allow subjects of regular users to modify object attributes, especially objects that they create. While the rGURA style is likely best suited to user attributes it does have broader applicability beyond the scope studied in this paper. For example, in a cloud computing scenario, the attributes of the same user may be administered distributively by different attribute providers and users may use these attributes to request the same online service [8]. These attributes may also be constrained by each other as modeled in GURA.

The remainder of this paper proceeds as follows. Section 2 introduces rGURA scheme. Section 3 formally defines reachability problems and overviews our analysis results. Section 4 provides algorithms and proofs of correctness and complexity. Section 5 outlines additional interesting results. Section 6 presents experimental results. We discuss related work in section 7 and conclude in section 8.

2. PRELIMINARIES

The user-attribute assignment model GURA (Generalized User-Role Assignment) was recently proposed [17]. It characterizes ARBAC97 style administration for user attributes. The core idea is that permissions to modify user attributes are associated with administrative roles. Administrators are made members of these roles and obtain the associated permissions. In this section, we provide an overview of rGURA and define relevant terminology.

2.1 User Attributes

User attributes characterize properties of users and are modeled as functions. We recognize two types of user attribute functions depending on the nature of the function's values: atomic-valued (*e.g.,* an attribute function Security Clearance that can take a single value such as *secret* or *confidential* for a given user) and set-valued (*e.g.,* an attribute function Role that can take multiple values such as *employee* and *manager*). For convenience we often say attribute rather than attribute function. More formally:

- U is the finite set of existing users in the system.
- ATTR is the finite set of user attribute functions, where $\text{attType} : \text{ATTR} \rightarrow \{\textbf{set}, \textbf{atomic}\}$.
- For each $att \in \text{ATTR}$, SCOPE_{att} is a finite set of atomic values which determines the range of att as follows.

$$\text{Range}(att) = \begin{cases} \text{SCOPE}_{att} & \text{if attType}(att) = \textbf{atomic} \\ \mathcal{P}(\text{SCOPE}_{att}) & \text{if attType}(att) = \textbf{set} \end{cases}$$

where $\mathcal{P}$ denotes the power set of a set.

DEFINITION 1 (**USER ATTRIBUTE**). *A* user attribute *is a function that maps each user to a value in the attribute's range, i.e.,*

$$\forall att \in \text{ATTR}.\ att : \text{U} \rightarrow \text{Range}(att)$$

Example 1. In Table 1 Clr (Clearance) and Dept (Department) are atomic-valued attributes, while Prj (Project) and Skill are set-valued. An example attribute assignment for user Alice is: Clr(Alice) = unclassified, Dept(Alice) = software, Proj(Alice) = {mobile, social, search} and Skill(Alice) = {web, security}.

2.2 Administrative Requests

In rGURA, *administrative requests* are made by members of administrative roles to modify attributes of users in U. A request takes effect only if it is authorized by an *administrative rule* introduced later in this section. An authorized request is called an *action*. In this paper we are concerned with the collective power of the administrative roles. We therefore do not distinguish specific members, and simply ascribe a request to an administrative role rather than to one of its members. Management of membership in administrative roles is outside the scope of rGURA. For simplicity we assume the finite set of administrative roles AR is flat, although as discussed later in the paper our analysis also applies to hierarchical AR.

Atomic-valued attributes are modified via an *assign* action which replaces the current value with a new value. For set-valued attributes, an *add* action is used to add a single atomic value to an existing attribute set, while a *delete* action is used to remove a specific atomic value from an existing set. More formally, for each $ar \in \text{AR}$, $u \in \text{U}$, $att \in \text{ATTR}$ and $val \in \text{SCOPE}_{att}$,

- $\textsf{assign}(ar,\ u,\ att,\ val)$ is a request made by (a member of) administrative role ar to assign value val to the atomic-valued attribute att of user u. Suppose AR = {gameleader, manager} and Alice's attribute assignments are as in example

Table 1: Example User Attributes

Attribute	Type	Scope	Range
Clr	**atomic**	$\text{SCOPE}_{\text{Clr}}$ = {unclassified, confidential, secret, topsecret}	Range(Clr) = $\text{SCOPE}_{\text{Clr}}$
Dept	**atomic**	$\text{SCOPE}_{\text{Dept}}$ ={software, hardware, finance, market}	Range(Dept) = $\text{SCOPE}_{\text{Dept}}$
Proj	**set**	$\text{SCOPE}_{\text{Proj}}$ ={search, game, mobile, social, cloud}	Range(Proj) = $\mathcal{P}(\text{SCOPE}_{\text{Proj}})$
Skill	**set**	$\text{SCOPE}_{\text{Skill}}$ ={web, system, server, win, linux, security}	Range(Skill) = $\mathcal{P}(\text{SCOPE}_{\text{Skill}})$

Table 2: Example Rules in rGURA_0 and rGURA_1 Schemes

N.	Relation	Admin Role	Precondition	Value
Example Rules in rGURA_0 Scheme				
1	$\text{can\_add}_{\text{Proj}}$	gameleader	mobile ∈ Proj(u) ∧ social ∈ Proj(u) ∧ ¬(cloud ∈ Proj(u))	game
2	$\text{can\_delete}_{\text{Proj}}$	gameleader	game ∈ Proj(u) ∧ social ∈ Proj(u)	game
3	$\text{can\_assign}_{\text{Dept}}$	manager	Dept(u) = software	market
4	$\text{can\_assign}_{\text{Dept}}$	manager	Dept(u) = hardware	market
Example Rules in rGURA_1 Scheme				
5	$\text{can\_add}_{\text{Proj}}$	gameleader	mobile ∈ Proj(u) ∧ social ∈ Proj(u) ∧ ¬(cloud ∈ Proj(u)) ∧ Dept(u) = software ∧ Clr(u) = unclassified ∧ web ∈ Skill(u) ∧ security ∈ Skill(u)	game
6	$\text{can\_delete}_{\text{Proj}}$	gameleader	game ∈ Proj(u) ∧ ¬(Clr(u) = topsecret)	game
7	$\text{can\_assign}_{\text{Dept}}$	manager	Dept(u) = software ∧ ¬(Clr(u) = unclassified) ∧ server ∈ Skill(u) ∧ win∈ Skill(u)	market
8	$\text{can\_assign}_{\text{Dept}}$	manager	Dept(u) = hardware ∧ ¬(Clr(u) = unclassified) ∧ server ∈ Skill(u) ∧ win∈ Skill(u)	market

1. The request assign(manager, Alice, Dept, hardware) is made by (a member of) administrative role manager to assign Alice to hardware department. If this request is authorized, Alice's Dept attribute will change from software to hardware.

- $\text{add}(ar,\ u,\ att,\ val)$ is a request made by ar to add value val to the set-valued attribute att of user u. E.g., add(manager, Alice, Proj, game) is a request made by manager to add Alice to the game project. If authorized, Proj(Alice) becomes {mobile, social, search, game}. The operation add has no effect if the value to be added is already present in the attribute.

- $\text{delete}(ar,\ u,\ att,\ val)$ is a request made by ar to delete val from attribute att of user u. E.g., delete(manager, Alice, Proj, game) is a request by manager to delete Alice from game project. If authorized, Proj(Alice) would revert back to {mobile, social, search}. As a set operation delete has no effect if the value being deleted is not present in the attribute.

DEFINITION 2 (**ADMINISTRATIVE REQUESTS**). Administrative requests *are made by members of administrative roles to modify user attributes.* $\text{REQ} = \text{ASSIGN} \cup \text{ADD} \cup \text{DELETE}$ *denotes the set of all possible administrative requests where,*

- $\text{ASSIGN} = \{\text{assign}(ar,\ u,\ att,\ val) \mid ar \in \text{AR} \wedge u \in \text{U} \wedge att \in \text{ATTR} \wedge \text{attType}(att) = \textbf{atomic} \wedge val \in \text{SCOPE}_{att}\}$
- $\text{ADD} = \{\text{add}(ar,\ u,\ att,\ val) \mid ar \in \text{AR} \wedge u \in \text{U} \wedge att \in \text{ATTR} \wedge \text{attType}(att) = \textbf{set} \wedge val \in \text{SCOPE}_{att}\}$
- $\text{DELETE} = \{\text{delete}(ar,\ u,\ att,\ val) \mid ar \in \text{AR} \wedge u \in \text{U} \wedge att \in \text{ATTR} \wedge \text{attType}(att) = \textbf{set} \wedge val \in \text{SCOPE}_{att}\}$

2.3 Administrative Rules

An administrative request takes effect only if it is authorized by an *administrative rule*. Administrative rules specify the necessary preconditions for authorizing administrative requests. A precondition is a logical formula expressed over user attributes that evaluates to **true** or **false** (*e.g.*, Clr(Alice) ≥ confidential ∧ game ∈ Proj(Alice)). The power of the administrative model lies in the expressive power of the preconditions, which can be specified by different grammars for defining preconditions. We define the grammars for two specific rGURA models later. An *administrative rule* specifies the authorization for administrative requests. Administrative rules are specified as follows.

DEFINITION 3 (**ADMINISTRATIVE RULES**). *Administrative rules are tuples in the following relations where* C *is a set of preconditions (specified by a formal grammar for each instance of a* GURA *model). For each atomic-valued attribute* $aua \in \text{ATTR}$*,*

$$\text{can\_assign}_{aua} \subseteq \text{AR} \times \text{C} \times \text{SCOPE}_{aua}$$

The rule $\langle ar,\ c,\ val\rangle \in \text{can\_assign}_{aua}$ *authorizes the requests* $\text{assign}(ar,\ u,\ aua,\ val)$ *if user* u *satisfies precondition* c*. For each set-valued attribute* $sua \in \text{ATTR}$*,*

$$\text{can\_add}_{sua} \subseteq \text{AR} \times \text{C} \times \text{SCOPE}_{sua}$$
$$\text{can\_delete}_{sua} \subseteq \text{AR} \times \text{C} \times \text{SCOPE}_{sua}$$

The rule $\langle ar,\ c,\ val\rangle \in \text{can\_add}_{sua}$ *authorizes the requests* $\text{add}(ar,\ u,\ sua,\ val)$ *if user* u *satisfies the precondition* c*, and the rule* $\langle ar,\ c,\ val\rangle \in \text{can\_delete}_{sua}$ *similarly authorizes requests* $\text{delete}(ar,\ u,\ sua,\ val)$ *if user* u *satisfies precondition* c*.*

Examples of administrative rules are shown in table 2. We explain selected rules here. Rule 1 authorizes members of gameleader role to add "game" to "project" attribute of any user who is currently in "mobile" and "social" projects but not in "cloud". Rule 4 states that manager role can assign a value of "market" to the "Dept" attribute of a user if the user currently belongs to the "hardware" department. We use can_assign to denote the collection of can_assign_{aua} relations for all atomic-valued attributes. That is,

$$\text{can\_assign} = \langle \text{can\_assign}_{\text{att}_1}, \ldots \text{can\_assign}_{\text{att}_m}\rangle,$$

where $\text{att}_1, \text{att}_2, \ldots, \text{att}_m$ are atomic-valued attributes. Similarly, we use can_add and can_delete as follows:

$$\text{can\_add} = \langle \text{can\_add}_{\text{att}'_1}, \ldots \text{can\_add}_{\text{att}'_n}\rangle$$
$$\text{can\_delete} = \langle \text{can\_delete}_{\text{att}'_1}, \ldots \text{can\_delete}_{\text{att}'_n}\rangle$$

where $\text{att}'_1, \text{att}'_2, \ldots, \text{att}'_n$ are set-valued attributes.

2.4 Restricted GURA (rGURA)

Our analysis focuses on a simple grammar for specifying the preconditions C, which is a subset of the precondition grammars defined for GURA [17]. Hence we call it the rGURA (restricted GURA) model. We expect practical administrative models to have richer precondition grammars, but these are likely to include the capabilities provided in rGURA. Our results therefore establish lower bounds on the analysis complexity for these richer models.

2.4.1 rGURA Scheme

An rGURA scheme is a state transition system where a state is an attribute assignment for each user and each attribute. State transitions are caused by authorized administrative requests to modify user attributes. We give the general definition for an rGURA scheme below, followed by two specific instantiations rGURA_0 and rGURA_1 with specific formal grammars for preconditions.

DEFINITION 4 (rGURA **SCHEME**). *An* rGURA *scheme is a state transition system* $\langle$U, ATTR, AR, Range[], attType[], SCOPE, Ψ, Γ, $\delta\rangle$ *where,*

- U, ATTR, AR, Range[] *and* attType[] *are defined above.*
- $\mathrm{SCOPE} = \langle \mathrm{SCOPE}_{att_1} \ldots \mathrm{SCOPE}_{att_n} \rangle$ *where* $att_i \in$ ATTR*, is the collection of the scopes of all attributes.*
- $\Psi = \langle$can_assign, can_add, can_delete$\rangle$*, is the collection of administrative rules for all attributes.*
- Γ *is the finite set of states. A* state $\gamma \in \Gamma$ *records assigned attribute values for each user. The user attribute assignment in state* γ*, denoted* UAA_γ*, contains tuples of the form* $\langle u, att, val\rangle$ *for every* $u \in$ U *and every* $att \in$ ATTR *such that att(u) = val. Formally,*

$$\begin{aligned}\mathrm{UAA}_\gamma = \{\langle u, att, val_1\rangle \mid u \in \mathrm{U} \wedge att \in \mathrm{ATTR} \wedge \\ val_1 \in \mathrm{Range}(att) \wedge (\forall val_2 \in \mathrm{Range}(att). \\ val_2 \neq val_1 \rightarrow \langle u, att, val_2\rangle \notin \mathrm{UAA}_\gamma)\}\end{aligned}$$

- $\delta : \Gamma \times \mathrm{REQ} \rightarrow \Gamma$ *is the transition function, where* REQ *is the set of all possible administrative requests defined above. Function* δ *is formally defined in table 3.*

2.4.2 The rGURA$_0$ Scheme

An rGURA_0 scheme is an instance of a rGURA scheme where the grammar for specifying preconditions is specified as follows. In each $\mathsf{can\_assign}_{aua}$ relation for each atomic-valued attribute, the preconditions in all rules are generated by the following grammar,

$$\begin{aligned}\varphi &::= \neg\, \varphi \mid \varphi \wedge \varphi \mid aua(u) = avalue \\ avalue &::= aval_1 \mid aval_2 \ \ldots \mid aval_n\end{aligned}$$

where $\mathrm{SCOPE}_{aua} = \{aval_1,\ aval_2,\ \ldots,\ aval_n\}$. In all rules in $\mathsf{can\_add}_{sua}$ and $\mathsf{can\_delete}_{sua}$ relations, the preconditions are formulas generated by the following grammar,

$$\begin{aligned}\varphi &::= \neg\, \varphi \mid \varphi \wedge \varphi \mid svalue \in sua(u) \\ svalue &::= sval_1 \mid sval_2 \mid \ \ldots \mid sval_m\end{aligned}$$

where $\mathrm{SCOPE}_{sua} = \{sval_1,\ sval_2,\ \ldots,\ sval_m\}$.

Example 2. The first part of table 2 shows example administrative rules in rGURA_0 based on the attributes listed in example 1. We assume that Bob is a member of administrative role "gameleader" and Alice is a user whose attributes are as given in example 1. Rule 1 authorizes Bob to add "game" to the Proj attribute of Alice. Bob is also authorized to delete "game" from Proj attribute of Alice based on rule 2. However, if Proj(Alice) = {mobile, social, cloud}, the add request will not take effect. Rules 3 and 4 are specified for the attribute Dept and they allow members of "manager" role to assign users who are currently in "software" or "hardware" department to "market" department. Note that multiple rules can be specified for the same attribute name and value combinations.

2.4.3 The rGURA$_1$ Scheme

An rGURA_1 scheme is an instance of a rGURA scheme where the preconditions in all rules can be specified using the grammar,

$$\varphi ::= \neg\varphi \mid \varphi \wedge \varphi \mid aua(u) = avalue \mid svalue \in sua(u)$$

where $avalue$ and $svalue$ are non-terminals and could be any value from the scope of any atomic-valued and set-valued attribute respectively. Similarly, non-terminal symbols aua and sua can be any atomic-valued and set-valued attribute respectively of the user u. Note that each attribute should be compared with a value from its respective scope, otherwise, the formulas return **false**.

Example 3. The second part in table 2 shows example administration rules in rGURA_1 based on attributes in table 1. Unlike rGURA_0, the preconditions of administrative rules in rGURA_1 can be specified using any attributes of the user. We assume that Alice has the same attribute assignment as in example 1. Bob is a member of "gameleader" role. Rule 5 authorizes Bob to add value "game" to Alice's Proj attribute. If Clr(Alice) = topsecret, the above request will not be authorized. Similarly, according to rule 7, Bob is not authorized to assign Alice to "market" department since Clr(Alice) = unclassfied. Other rules are self-explanatory.

3. PROBLEM DEFINITION AND RESULTS OVERVIEW

In this section, we define reachability problems and provide an overview of the complexity analysis results.

3.1 Attribute Reachability Problem

The attribute reachability problem (or simply the reachability problem) is informally defined as follows. Given an initial state with an assignment of each attribute for a particular user, can members of a set of administrative roles issue one or more administrative requests that transition the system to a target state with specific attribute assignments for that user? Before formally defining the reachability problems in the context of the rGURA_0 and rGURA_1 schemes defined earlier, we note two simplifications. Firstly, reachability analysis questions are about one user. Since modifications to attributes of one user have no impact on potential changes to the attributes of other users, we only consider the attribute assignment of a single user of interest in a state. That is, we assume U = {**u**} in all of our analysis. Secondly, reachability problems ask about the power of members of a set of administrative roles SUBAR $\subseteq$ AR. In this case, the administrative rules specified for roles not in SUBAR need not be considered for the analysis. We assume that the scheme is provided with Ψ which only contains administrative rules for roles in SUBAR, that is, AR = SUBAR.

The above simplification eases our presentation without loss of generality. We now define the notion of a *query* which is concerned about whether a particular state "satisfies" specific attribute assignments for a given user.

DEFINITION 5 (**REACHABILITY QUERY**). *A* Reachability Query *specifies value-assignments for selected attributes of a user in the target state. Let* Q *denote the set of queries. Each query* $q \in$ Q *is a subset of* UAA_γ.

Table 3: State Transition Function $\delta : \Gamma \times \text{REQ} \rightarrow \Gamma$

(1) Let the source state be γ_1. In all requests in the first column, $ar \in \text{AR}$, $u \in \text{U}$, $att \in \text{ATTR}$, $val' \in \text{SCOPE}_{att}$.
(2) Satisfy: $\text{U} \times \text{C} \times \Gamma \rightarrow \{\textbf{true}, \textbf{false}\}$, returns **true** if user $u \in U$ satisfies precondition $c \in \text{C}$ in state $\gamma \in \Gamma$, else **false**.

Request	Target State
assign(ar, u, att, val')	**if** $\exists\ \langle ar,\ c,\ val\rangle \in \mathsf{can\_assign}_{att} \wedge \text{Satisfy}(u,\ c,\ \gamma_1)$ **then** transition to target state γ_2 where: $\text{UAA}_{\gamma_2} = \text{UAA}_{\gamma_1} \setminus \langle u, att, val\rangle \cup \langle u, att, val'\rangle$ **else** remain in γ_1
add(ar, u, att, val')	**if** $\exists\ \langle ar,\ c,\ val\rangle \in \mathsf{can\_add}_{att} \wedge \text{Satisfy}(u,\ c,\ \gamma_1)$ **then** transition to target state γ_2 such that $\text{UAA}_{\gamma_2} = \text{UAA}_{\gamma_1} \setminus \langle u, att\ , setval\rangle \cup \langle u, att\ , setval'\rangle$ where $att(u) = setval$ in state γ_1 and $setval' = setval \cup \{val'\}$, **else** remain in γ_1
delete(ar, u, att, val')	**if** $\exists\ \langle ar,\ c,\ val\rangle \in \mathsf{can\_delete}_{att} \wedge \text{Satisfy}(u,\ c,\ \gamma_1)$ **then** transition to target state γ_2 where: $\text{UAA}_{\gamma_2} = \text{UAA}_{\gamma_1} \setminus \langle u, att\ , setval\rangle \cup \langle u, att\ , setval'\rangle$ where $att(u) = setval$ in state γ_1 and $setval' = setval \setminus \{val'\}$, **else** remain in γ_1

We say a query is satisfied at a "strict" level if every attribute assignment specified in the query is exactly the same as that in the concerned state. A query can be satisfied at a "relaxed" level if in the concerned state every atomic-valued attribute assignment specified in the query is exactly the same but if every set-valued attribute assignment in the concerned state is a superset of the corresponding set-valued attribute assignment specified in the query. Note that in both strict and relaxed levels the atomic-valued attributes in the concerned state should exactly match the query. The distinction arises in the values of set-valued attributes since the values in the concerned state can either exactly equal or simply contain (superset) the value specified in the query. Since atomic-valued attributes do not affect query satisfaction levels, we illustrate the difference on set-valued attributes. For instance, let ATTR = {Proj} and U = {Alice}. An example query is a user attribute assignment: q = ⟨ Alice, Proj, {cloud, game}⟩. In $\text{RP}_{=}$ or strict query type, q can be satisfied only by states γ where UAA_{γ} = {⟨ Alice, Proj, {cloud, game}⟩}. While in $\text{RP}_{\supseteq}$ or relaxed query type, q can be satisfied by any state γ' where $\text{UAA}_{\gamma'}$ = {⟨Alice, Proj, $setval$⟩} and {cloud, game} $\subseteq setval$. We formally specify the queries and the satisfaction levels below.

DEFINITION 6 (**REACHABILITY QUERY TYPES**). *Given a scheme* ⟨U, ATTR, AR, Range[], attType[], SCOPE, Ψ, Γ, δ⟩, *we define two* Reachability Query Types*:*

- $\text{RP}_{=}$ *queries have the entailment function* $\vdash_{\text{RP}_{=}} : \Gamma \times \text{Q} \rightarrow \{\textbf{true}, \textbf{false}\}$ *which returns* **true** *(i.e.,* $\gamma \vdash_{\text{RP}_{=}} q$*) if* $\forall$ ⟨*u, att, val*⟩ $\in q$. ⟨*u, att, val*⟩ $\in \text{UAA}_{\gamma}$.
- $\text{RP}_{\supseteq}$ *queries have the entailment function* $\vdash_{\text{RP}_{\supseteq}} : \Gamma \times \text{Q} \rightarrow \{\textbf{true}, \textbf{false}\}$ *which returns* **true** *(i.e.,* $\gamma \vdash_{\text{RP}_{\supseteq}} q$*) if* $\forall$ ⟨*u, att, val*⟩ $\in q$*: (1)* $\langle u,\ att,\ val\rangle \in \text{UAA}_{\gamma}$ *if* attType*(att)* = **atomic** *and (2)* $\exists\langle u,\ att,\ val'\rangle \in \text{UAA}_{\gamma}$ *where* $val' \supseteq val$ *if* attType*(att)* = **set**.

DEFINITION 7 (**PLAN**). *A* plan *is a sequence of authorized administrative requests that causes a transition from one state to another. Given a scheme* ⟨U, ATTR, AR, Range[], attType[], SCOPE, Ψ, Γ, δ⟩ *and states* γ, $\gamma' \in \Gamma$, *a sequence of authorized requests* $\langle\ req_1, req_2, \ldots, req_n\rangle$ *where* $req_i \in \text{REQ}$ $(1 \leq i \leq n)$ *is called a* plan *to transition from an initial state* γ *to the target state* γ' *if:* $\gamma \overset{req_1}{\rightarrow} \gamma_1 \overset{req_2}{\rightarrow} \gamma_2 \ldots \overset{req_n}{\rightarrow} \gamma'$. *The arrow denotes a successful transition from one state to another in response to an administrative request* req_i *that is authorized by rules in* Ψ. *For convenience, we also write* $\gamma \overset{plan_{\Psi}}{\rightsquigarrow} \gamma'$.

The reachability problem is concerned about whether it is possible to transition an initial state to some target state where the attribute-value assignments satisfy a particular query.

DEFINITION 8 (**REACHABILITY PROBLEMS**). *Given a scheme* ⟨U, ATTR, AR, Range[], attType[], SCOPE, Ψ, Γ, δ⟩*:*

- *An* $\text{RP}_{=}$ Reachability Problem *instance* I *is of the form* ⟨γ, q⟩ *where* $\gamma \in \Gamma$ *and* $q \in \text{Q}$ *and asks does there exist a plan* P *for problem instance* I *such that* $\gamma \overset{P_{\Psi}}{\rightsquigarrow} \gamma'$ *and* $\gamma' \vdash_{\text{RP}_{=}} q$.
- *An* $\text{RP}_{\supseteq}$ Reachability Problem *instance* I *is of the form* ⟨γ, q⟩ *where* $\gamma \in \Gamma$ *and* $q \in \text{Q}$ *and asks does there exist a plan* P *for problem instance* I *such that* $\gamma \overset{P_{\Psi}}{\rightsquigarrow} \gamma'$ *and* $\gamma' \vdash_{\text{RP}_{\supseteq}} q$.

3.2 Overview of Analysis Results

It is evident from the definitions, given the same scheme and problem instance, if the $\text{RP}_{=}$ problem has a positive answer then so does the $\text{RP}_{\supseteq}$ problem, but not vice versa. The size of input for each problem instance I is the sum of size of each set in I. Our analysis proves the complexity of attribute reachability problems for rGURA schemes in general is PSPACE-complete. However, we have identified instances of the rGURA scheme with some restrictions on the precondition specification in administrative rules that have more practical time complexity. Moreover, these instances have many practical applications as will be discussed later. The following restrictions are considered:

- **No negation** ($\overline{N}$): Ψ satisfies $\overline{N}$ if no rules in Ψ use negation in preconditions.
- **No deletion** ($\overline{D}$): Ψ satisfies $\overline{D}$ if for all set-valued attributes sua in ATTR, $\mathsf{can\_delete}_{sua}$ is *empty*. This restriction applies only to problem instances containing set-valued attributes. It implies that once a value is added, it can never be deleted.
- **Single rule** (SR): Ψ satisfies SR if: (1) for each atomic-valued attribute $aua \in$ ATTR, there is at most one precondition associated with a particular value assignment in the $\mathsf{can\_assign}_{\mathsf{aua}}$ and (2) for each set-valued attribute $sua \in$ ATTR, there is at most one precondition associated with a particular value assignment in each of the $\mathsf{can\_add}_{\mathsf{sua}}$ and $\mathsf{can\_delete}_{\mathsf{sua}}$. That is (1) $\forall att \in \text{ATTR} \wedge \text{attType}(att) =$ **atomic**. $\forall val \in \text{SCOPE}_{att}, |\{c \mid \langle ar, c, val\rangle \in \mathsf{can\_assign}_{\mathsf{att}}\}| \leq 1$ and (2) $\forall\ att \in \text{ATTR} \wedge \text{attType}(att) =$ **set**. $\forall\ val \in \text{SCOPE}_{att}$, $|\{c \mid \langle ar, c, val\rangle \in \mathsf{can\_add}_{att}\}| \leq 1$ and $|\{c \mid \langle ar, c, val\rangle \in \mathsf{can\_delete}_{att}\}| \leq 1$.

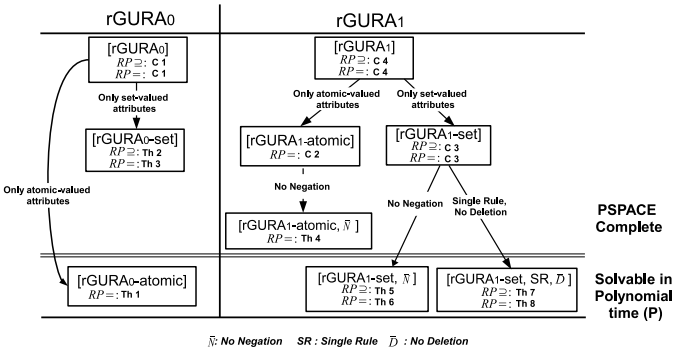


Figure 2: *Complexity Results for Different Classes of Reachability Problems*

The SR restriction means that the preconditions in can_assign, can_add or can_delete rules specified for each attribute-value pair are unique. However, the corresponding rules could still be assigned to different administrative roles. For instance, suppose another rule is in the $\text{can\_add}_{\text{Proj}}$ in item 1 in table 2 as follows.

$$(\text{manager}, \text{mobile} \in \text{Proj(u)} \wedge \text{social} \in \text{Proj(u)} \wedge \neg(\text{cloud} \in \text{Proj(u)}), \text{game})$$

The SR restriction is still satisfied since given an attribute and value pair (the attribute being "Proj" and the value being "game"), the preconditions remain the same even though there are multiple rules that allow adding the same value to that attribute. Here, the two rules allow members of different administrative roles to add "game" to the attribute "Proj". If the above rule is modified as follows

$$(\text{manager}, \text{mobile} \in \text{Proj(u)} \wedge \neg(\text{cloud} \in \text{Proj(u)}), \text{game})$$

the SR restriction is no longer satisfied. Similar considerations apply to rules in $\text{can\_delete}_{\text{att}}$ and $\text{can\_assign}_{\text{att}}$.

Another restriction is the type of attributes contained in the system. It's possible that a system deals only with atomic-valued or set-valued attributes. These restrictions are also considered. Many attribute semantics and applications work well with the above restrictions to be of practical use. The positive results are that in such situations reachability analysis can be performed efficiently. For instance, consider a scenario to express necessary prerequisites to register for a course in a university. Let a "course" attribute keep track of the set of courses a student has earned credits for. The preconditions to register for a course in this scenario are mostly positive which commonly check whether the student has successfully completed all the prerequisite courses. This would satisfy the $\overline{N}$ restriction. Consider a "Skill" attribute that keeps track of user skills (*e.g.,* web, system, etc. as mentioned in table 1) which may never need to be deleted after add and hence satisfies the $\overline{D}$ restriction. The SR restriction can be satisfied in situations where there are no alternative preconditions that allow a particular value to be assigned/added/deleted to/from an attribute. That is, there is exactly one and only way an attribute can obtain a particular value.

We use [rGURA_x-[atomic, set], `Restriction`] to denote a specialized rGURA scheme on which we perform reachability analysis. The subscript x takes a value of 0 or 1 representing an rGURA_0 or rGURA_1 scheme and [atomic, set] means that the scheme contains only the specified type of attributes (if not specified, it represents the general case where both types of attributes are included). `Restriction` represents possible combinations of SR, $\overline{D}$ and $\overline{N}$ meaning that the rules in the scheme satisfies those restrictions. Thus [rGURA_1-atomic, $\overline{N}$] denotes an rGURA_1 scheme $\langle \text{U}, \text{ATTR}, \text{AR}, \text{Range}[], \text{attType}[], \text{SCOPE}, \Psi, \Gamma, \delta \rangle$ where ATTR contains only atomic-valued attributes and Ψ satisfies $\overline{N}$.

Figure 2 summarizes our analysis using the above notation. The left column shows the results for rGURA_0 variations and the right column for rGURA_1 variations. Each scheme is specified in a box which includes theorem and corollary number (abbreviated Th and C respectively) in section 4 that provides the proof for that specific scheme and query type. An arrow from a box to another indicates that the restrictions specified on the arrow when applied to the former box lead to the latter. Note that for schemes that contain only atomic-valued attributes, only $\text{RP}_=$ queries are considered. For schemes that only contain set-valued attributes or both types of attributes, both $\text{RP}_=$ and $\text{RP}_\supseteq$ are considered. Reachability analysis in general is intractable except $\text{RP}_=$ in [rGURA_0-atomic]. While the PSPACE-hardness results are not highly surprising, several of them are interesting and insightful.

(1) In the rGURA_0 column, [rGURA_0-atomic] is the only scheme that has a polynomial-time algorithm without further restrictions. Since this scheme only has atomic-valued attributes, only the $\text{RP}_=$ query is considered. However, $\text{RP}_\supseteq$ and $\text{RP}_=$ in [rGURA_0-set] abruptly escalate to PSPACE-complete. This big jump is caused by negative preconditions and delete operations. Since [rGURA_0-set] is similar to ARBAC97, many of the results from prior work on reachability analysis in ARBAC97 can be adapted here [31]. Thus if $\overline{N}$ is enforced, $\text{RP}_\supseteq$ is polynomial-time solvable. If $\overline{D}$ is enforced, $\text{RP}_\supseteq$ for [rGURA_1-set, $\overline{D}$] is in NP (see section 5).

(2) In rGURA_1, the complexity for reachability problems differs sharply from rGURA_0 for systems containing only atomic-valued attributes (from polynomial-time to intractable) while not so much for systems containing only set-valued attributes (both are PSPACE-complete). The huge increase in complexity for schemes containing only atomic-valued attributes is caused by mutual constraints of attributes on each other in the preconditions for rGURA_1. In order to satisfy a query, attribute values may need to be reassigned a large number of times. In addition, each attribute needs to be satisfied simultaneously which also increases the complexity.

(3) Also interestingly, similar restrictions work fairly differently on atomic-valued and set-valued attributes in rGURA_1. A notable example is $\overline{N}$. $\text{RP}_=$ in [rGURA_1-atomic, $\overline{N}$] is surprisingly intractable while $\text{RP}_=$ and $\text{RP}_\supseteq$ for [rGURA_1-set, $\overline{N}$] are in P.

(4) The [rGURA_1-set] scheme is intractable for both $\text{RP}_=$ and $\text{RP}_\supseteq$. However, two types of restrictions yield polynomial-time analysis results: [rGURA_1-set, $\overline{N}$] and [rGURA_1-set, SR, $\overline{D}$].

4. FORMAL PROOFS

In this section, we walk through proofs for the results in figure 2. For schemes with PSPACE-complete complexity, we show reduction to a known problem in that class. For schemes in the polynomial-time solvable class, we provide polynomial-time algorithms with correctness proofs. There are two parts to a PSPACE-complete proof: a proof to show that the scheme is solvable in PSPACE and a proof to show that the scheme is PSPACE-hard. The first part is the same for all schemes in figure 2.

LEMMA 1. *Every scheme in figure 2 is in PSPACE.*

PROOF. Given any problem instance, a Non-deterministic Turing Machine can simulate the following algorithm. In each state, the Turing machine stores the current user-attribute assignments, attributes scopes, query and administrative policies. These are needed to guess the next possible states. In each step, it guesses the next possible states it can enter (there maybe more than one possible

next states) by running the administrative policies against the current user-attribute assignments. For each next state, the Turing machine checks it against the query. If the query is satisfied, the process stops. Otherwise, it repeats the same process. The space needed for each state is polynomial to the input size which includes the initial user-attribute assignments, attributes scopes[1], query and administrative policies. Thus, all problems are in NPSPACE and thus also in PSPACE as implied by Savitch's theorem [32]. □

4.1 rGURA$_0$ Schemes

Consider the [rGURA$_0$-atomic] scheme which only contains atomic-valued attributes. In an rGURA$_0$ scheme recall that modifications to one attribute have no impact on future modifications to other attributes (see section 2.4.2). Hence it is sufficient to confine our analysis to the set of rules specified for a single attribute, repeating this process for each attribute in turn. In other words, we can analyze the reachability of each attribute in the query independently and the combine the results. Note that this strategy does not work for rGURA$_1$ schemes.

THEOREM 1. RP$_=$ *for* [rGURA$_0$-*atomic*] *is solvable in P.*

PROOF. The reachability query for each single attribute is transformed to a path search problem between two nodes in a directed graph. In this graph, the nodes are the values from the scope of this attribute. A directed edge from node n_1 to node n_2 represents an action of assigning n_2 to this attribute if the current value of that attribute is n_1. Detailed proof is provided in the appendix. □

THEOREM 2. RP$_\supseteq$ *for* [rGURA$_0$-*set*] *is PSPACE-complete.*

PROOF. Given lemma 1, it suffices to show PSPACE-hardness. As earlier, it is feasible to analyze the complexity of each attribute independent of each other. Our proof is to show the reduction from the role reachability problem. Combining the above observation, the fact that role can be treated as simply another set-valued user attribute and [rGURA$_0$-set] has the same expressive as miniARBAC97 [31], the reduction is straight forward as presented in the appendix. □

The miniARBAC97 work is useful for reduction with respect to RP$_\supseteq$ queries. However, it does not deal with RP$_=$ for which we utilize the SAS planning problem [5] from artificial intelligence.

THEOREM 3. RP$_=$ *for* [rGURA$_0$-*set*] *is PSPACE-complete.*

PROOF. Detailed proof is provided in the appendix. □

COROLLARY 1. RP$_\supseteq$ *and* RP$_=$ *for* [rGURA$_0$] *are PSPACE-complete.*

PROOF. Since RP$_\supseteq$ and RP$_=$ in [rGURA$_0$] can also be answered by querying each attribute separately and this scheme supports both atomic and set-valued attributes, this result follows from Lemma 1 and Theorems 1, 2 and 3. □

This completes the left hand side of figure 2.

4.2 rGURA$_1$ Schemes

For rGURA$_1$-atomic schemes we have the following results for RP$_=$. (Recall that RP$_\supseteq$ does not apply to rGURA$_1$-atomic schemes.)

[1]Although some of the attributes ranges may be encoded with smaller space than the number of values it may take (e.g., a integer with a range of [1,n] can be encoded by O(1) rather than O(n)), the space needed in the input is still polynomial because the administrative policies cannot be encoded in the same method.

THEOREM 4. RP$_=$ *for* [rGURA$_1$-*atomic,* $\overline{N}$] *is PSPACE-complete.*

PROOF. By lemma 1, it suffices to show PSPACE-hardness. The SAS planning problem under the U restriction is PSPACE-complete [5]. We give a reduction from [SAS planning, U] to [rGURA$_1$-atomic, $\overline{N}$]. In the former, only positive conditions are allowed in the operators (pre and post) which is accommodated by the $\overline{N}$ restriction. Consider an instance of the SAS planning problem $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$. (1) Map each state variable to one user attribute whose scope corresponds to the domain of the variable. Thus, s_0 and s_* map to two different attribute assignments. (2) Since each operator updates only one variable, it can be mapped to one rule in can_assign$_{att}$ where att is the corresponding user attribute of the variable updated in the operator. The value to be assigned is the same as that in post in the operator. The preconditions pre and prv only specifies positive precondition (no negative comparisons between state variables and values from their domains) and they can be specified using conjunctions. Negation is not required. The reduction process takes O($|\mathcal{V}| + |\mathcal{O}|$). This establishes that RP$_=$ for [rGURA$_0$-atomic, $\overline{N}$] is PSPACE-hard. □

COROLLARY 2. RP$_=$ *for* [rGURA$_1$-*atomic*] *is PSPACE-complete.*

PROOF. Follows from Theorem 4 and the fact that [rGURA$_1$-atomic, $\overline{N}$] is a sub-problem of [rGURA$_1$-atomic]. □

For rGURA$_1$-set we begin with the following observations.

COROLLARY 3. RP$_\supseteq$ *and* RP$_=$ *for* [rGURA$_1$-*set*] *are PSPACE-complete.*

PROOF. This follows from Theorems 2 and 3 and the fact that RP$_\supseteq$ and RP$_=$ for [rGURA$_0$-set] are sub-problems of RP$_\supseteq$ and RP$_=$ for [rGURA$_1$-set] respectively. □

COROLLARY 4. RP$_\supseteq$ *and* RP$_=$ *for* [rGURA$_1$] *are PSPACE-complete.*

PROOF. This follows from Corollaries 2 and 3. □

We now consider the RP$_\supseteq$ problem for [rGURA$_1$-set, $\overline{N}$] which can be solved in polynomial-time by Algorithm 1. The algorithm works as follows. For RP$_\supseteq$, we only need to consider issuing add requests to the set-valued attributes. This is because only positive preconditions are allowed, the rules cannot specify addition of new values to the set-valued attribute based on absence of certain values in the existing set. So existing values need not be removed in order to successfully add a new value. Thus, we only need to investigate can_add rules and completely ignore can_delete rules. Furthermore, additional values can be added to the attribute even if the desired set value is reached because it deals with the RP$_\supseteq$ problem. Starting from the given state, we traverse all rules in can_add and try the add requests allowed by any rule if the corresponding attribute value is not yet in the current set. Algorithm 1 terminates either when (1) the current set can no longer be augmented by the rules in can_add, or (2) all attributes are assigned with all values from their scope. The outer while loop is required because a value added to one or more of the attributes in an earlier round can potentially enable new additions in subsequent rounds. This is due to rGURA$_1$ preconditions where attributes can constrain each other.

THEOREM 5. RP$_\supseteq$ *for* [rGURA$_1$-*set,* $\overline{N}$] *is in P.*

PROOF. Algorithm 1 provides a polynomial-time solution.
Correctness. (1) Assume that Algorithm 1 returns a $plan$. If the $plan$ is empty, the query q is trivially satisfied in γ. Otherwise, it is ensured that if executed sequentially, there exists at least one

Algorithm 1 Plan Generation for $\mathrm{RP}_{\supseteq}$ in [rGURA_1-set, $\overline{N}$]

1: **Input:** problem instance I = $\langle \gamma, q \rangle$ **Output:** plan or false
2: $plan$ = $\langle\rangle$;
3: Begin with state s=γ;
4: **if** $s \vdash_{\mathrm{RP}_{\supseteq}} q$ **then return** $plan$
5: **while** $true$ **do**
6: **Let** $Save$ = UAA_s;
7: **for** each $att \in$ ATTR and $\langle ar, c, val \rangle \in$ can_add$_{att}$ **do**
8: **if** s satisfies precondition $c \wedge (\nexists \langle u, att, sv \rangle$
9: $\in \mathrm{UAA}_s$ such that $val \in sv$) **then**
10: Suppose that $\langle u, att, setval \rangle \in \mathrm{UAA}_\gamma$;
11: Go to state t such that
12: UAA_t == $\mathrm{UAA}_s \setminus \{\langle u, att, setval \rangle\}$
13: $\cup \{\langle u, att, setval \cup \{val\}\}$;
14: s = t;
15: $plan$ = $plan$.**append**(add(ar, u, att, val));
16: **end if**
17: **end for**
18: **if** UAA_s == $Save$ **then break; else** $Save$ = UAA_s;
19: **end if**
20: **end while**
21: **if** $s \vdash_{\mathrm{RP}_{\supseteq}} q$ **then return** $plan$ **else return false end if**

rule in can_add that authorizes each action in the $plan$. Thus, the $plan$ takes γ to another state that satisfies q. (2) When Algorithm 1 returns $false$, there does not exist a $plan$. We use contradiction. Assume $plan$ = $\langle \mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_n \rangle$ is a valid plan of length n and is not detected by algorithm 1. Without loss of generality, we assume a_k ($1 \leq k \leq n$) is not detected and the state before a_k is cur$'$. Thus, according to line 8 in algorithm 1, either (a) there does not exist a rule whose preconditions are satisfied in cur$'$ to authorize a_k, or (b) cur$'$ already contains the attribute value to be added in request a_k. In either case such an a_k cannot exist. (3) Algorithm 1 always terminates because the number of attributes and the attribute values to be added to γ are bounded. **Complexity**. The complexity is determined by the number of times all the rules in can_add are traversed. In the worst case, one value is added to one attribute during each round of rule traversing. Thus, the complexity of Algorithm 1 is $\mathrm{O}((\sum_{att \in \mathrm{ATTR}} |\mathrm{SCOPE}_{att}|) \times |\mathsf{can\_add}|)$ and it is polynomial. $|\mathsf{can\_add}|$ represents the size of all rules for all attributes. □

Next we consider the $\mathrm{RP}_{=}$ problem for [rGURA_1-set, $\overline{N}$].

THEOREM 6. $\mathrm{RP}_{=}$ *for* [rGURA_1*-set,* $\overline{N}$] *is in P.*

PROOF. The proof is by reduction to the STRIPS planning problem [7] shown in the appendix. □

We now consider $\mathrm{RP}_{\supseteq}$ for [rGURA_1-set, SR, $\overline{D}$] for which Algorithm 2 provides a polynomial-time solution. The $\overline{D}$ restriction obviates the need to include the rules from can_delete relations. The SR restriction provides a critical advantage in our analysis. Since a unique precondition facilitates addition of a value to an attribute, the number of paths in the search space is greatly reduced. The algorithm works by traversing backwards from the target state as follows. Assume that the problem instance is $\langle \gamma, q \rangle$. For at least one attribute, q requires a value which is a superset of that in γ (otherwise, q is already satisfied by γ). For those attribute values in q not in γ, there must be a corresponding add action in the plan if it exists. In addition, in order to add those values, attribute values which appear as positive preconditions in the administrative rules which authorize those add actions must also be added and so on. Thus, the basic idea is to use backward search to compute all attribute values that are required to be added in order to satisfy q. This is done by tracing rules in can_add for attributes and values

Algorithm 2 Plan Generation for $\mathrm{RP}_{\supseteq}$ in [rGURA_1-set, SR, $\overline{D}$]

1: **Input:** problem instance I = $\langle \gamma, q \rangle$ **Output:** plan or false
2: $toadd$ = $\emptyset$; $npre$ = $\emptyset$; cur = $\emptyset$; $plan$ = $\langle\rangle$;
3: **if** $\gamma \vdash_{\mathrm{RP}_{\supseteq}} q$ **then return** $\langle\rangle$
4: $\forall$ ⟨u, att, vset⟩ $\in \mathrm{UAA}_\gamma$. $\forall$ val $\in$ vset.
5: $cur' = cur \cup \{$(att, val)$\}$;
6: $\forall$ ⟨u, att, vset⟩ $\in q$. $\forall$ val $\in$ vset.
7: $toadd' = toadd \cup \{$(att, val)$\}$;
8: **Repeat**:
9: $\forall$ (att, val) $\in toadd$
10: $ppre$ = {(att$_1$, val$_1$) | $\exists \langle ar, c, val \rangle \in can\_add_{att}$ such
11: that $val_1 \in att_1(u)$ is a conjunct in c};
12: $toadd' = toadd \cup ppre \setminus cur$;
13: **Until** $toadd$ **does not change**
14: **if** $\exists$ (att, val) $\in toadd$ such that $\nexists \langle ar, c, val \rangle \in can\_add_{att}$
15: **then return false end if**
16: $\forall$ $(att, val) \in toadd$
17: $npre' = npre \cup$ {att$_1$, val$_1$) | $\exists \langle ar, c, val \rangle \in can\_add_{att}$
18: such that $\neg(val_1 \in att_1(u))$ is a conjunct in c};
19: **if** $npre \cap cur$!= $\emptyset$ **then return false end if**
20: Construct a directed graph $G = \langle V, E \rangle$;
21: V = $toadd$; E = $\emptyset$;
22: **for** each pair of nodes $((att, val), (att_1, val_1)) \in$ V **do**
23: **if** ($\exists \langle ar, c, val_1 \rangle \in can_add_{att_1}$such that $val \in att(u)$
24: is a conjunct in c) $\vee$($\exists \langle ar, c, val \rangle \in can\_add_{att}$ such
25: that $\neg(val_1 \in att_1(u))$ is a conjunct in c)
26: **then** E$'$ = E $\cup$ $\{\langle (att, val), (att_1, val_1) \rangle\}$; **end if**
27: **end for**
28: $plan$ = Topological ordering on graph G;
29: **if** topological ordering is successful **then return** $plan$;
30: **else return false**; **end if**

that need to be added and recursively for those values required in the previous state (line 5 - line 9). Till now, only positive preconditions have been considered. If negative preconditions of any add request for values to be added are not satisfied in γ, q cannot be satisfied (since they can never be deleted). Otherwise, negative preconditions can only be introduced during each step of adding new values. Thus, those add actions need to be ordered based on mutual dependencies. This is achieved by creating a directed graph which reflects dependencies of attribute values (line 16 - line 21). A plan is a topological ordering of the graph (line 22). If there are cycles in the graph, q can never be satisfied.

THEOREM 7. $\mathrm{RP}_{\supseteq}$ *for* [rGURA_1*-set,* SR*,* $\overline{D}$] *is in P.*

PROOF. Algorithm 2 provides a polynomial-time solution.
Correctness. (1) *If algorithm 2 returns a plan* and it is empty, query q is satisfied in γ. If the $plan$ is not empty, we assume $plan$ = $\langle \mathrm{a}_1, \mathrm{a}_2, \ldots \mathrm{a}_n \rangle$. We prove that the plan is valid. We first prove that the first action a_1 is allowed to be executed. Firstly, its positive precondition is satisfied in γ. The $repeat$ loop (line 5) only stops when $toadd$ does not change. It means positive preconditions for all vertices already in $toadd$ are either satisfied by $ppre$ or cur. Since a_1 is the first action, its positive precondition is satisfied in cur. Secondly, its negative precondition is satisfied in γ (otherwise, line 14 returns false). If request a_k ($1 \leq k \leq n$) is authorized, then a_{k+1} is also allowed because a_{k+1}'s positive precondition may be satisfied in cur or action a_k (and its negative precondition is satisfied by both cur and a_k). Thus, sequentially executing the plan leads to a set of reachable states. q is satisfiable because in the first round of $repeat$, $toadd$ contains attribute values in q while not in γ. (2) *If algorithm 2 returns false*, there are several reasons: (a) no can_add rule for some of the attribute values in $toadd$; (b) some of the negative preconditions are not satisfied in γ; (c) there are loops in the graph created by lines 17-21. We show that at the least all at-

tribute values in $toadd$ should be added to reach q. Assume $toadd$ = {(a_1, v_1), (a_2, v_2) ... (a_n, v_n)}. Without loss of generality, we let $vset$ = $toadd \setminus$ {(a_k, v_k)} which if added, transitions γ to a state in which q is satisfied. Thus (a_k, v_k) is in $ppre$ and is not in cur in some round of $repeat$ through lines 5 to 9. If it is not added, it means $\langle ar, c, val\rangle \notin \mathsf{can\_add}_{a_{k-1}}$ and (a_{k-1}, val) cannot be added. Thus, other attribute values which depend on these attribute values are unreachable. Hence, q will not be satisfied. This suffices to prove that situations (a), (b) and (c) are all correct. (3) *Algorithm 2 always terminates.* The only loop is from line 5 to line 9. It always ends because the number of rules and conjuncts in preconditions is finite. **Complexity**. The graph can be created in polynomial time and the topological sort also takes polynomial time. The total complexity is $O((\sum_{att\in \mathrm{ATTR}}|\mathrm{SCOPE}_{att}|) \times |\Psi|)$. □

A minor extension to algorithm 2 can solve $\mathrm{RP}_{=}$ for [rGURA_1-set, SR, $\overline{D}$]. Since in this problem, q requires that a state has exactly the same values for each attribute, adding attribute values which are not specified in q is not allowed (attribute values in q should be superset of corresponding attribute values in γ, otherwise, q is not satisfiable). Before topologically sorting the graph, we do the following preprocessing: (1) vset is a set of attribute values which is in q while not in γ and (2) detect in the created graph whether there exists a vertex in vset which contains incoming edges from vertices not in vset. If yes, return false. Otherwise, remove all other vertices not in vset. A topological ordering of the vertex in vset is a valid plan for the problem.

THEOREM 8. $\mathrm{RP}_{=}$ *for* [rGURA_1*-set,* SR, $\overline{D}$] *is in P.*

PROOF. **Correctness**. The only change to algorithm 2 is in the last step (line 22). Because of the nature of $\mathrm{RP}_{=}$, if there does not exist such a vset as explained earlier, q is never satisfiable. **Complexity**. As described above, there is only one additional process compared to algorithm 2: to detect vset which takes $O(\sum_{att\in \mathrm{ATTR}}|\mathrm{SCOPE}_{att}|)$. Thus, the total complexity is $O((\sum_{att\in \mathrm{ATTR}}|\mathrm{SCOPE}_{att}|) \times |\Psi|)$ which is polynomial. □

5. ADDITIONAL RESULTS

Earlier we've shown rGURA schemes for which reachability problems are either PSPACE-complete or P. Here we briefly go over additional schemes for which the $\mathrm{RP}_{\supseteq}$ is NP-complete and NP. Firstly we look at additional results for $\mathrm{RP}_{\supseteq}$ for [rGURA_0-set]. Sizeable results on role reachability in miniARBAC97 can be borrowed directly and utilized for $\mathrm{RP}_{\supseteq}$ for [rGURA_0-set]. The reason is that [rGURA_0-set] is designed with the same expressive power as ARBAC97 (considering role as one user attribute). Even though there are multiple attributes in [rGURA_0-set], their management is independent of each other [20, 31, 34].

For $\mathrm{RP}_{\supseteq}$ in [rGURA_1-set, $\overline{N}$], we look at a relaxed restriction compared to $\overline{N}$, PosCanAdd which is defined as: in all rules in can_add, only positive preconditions are allowed. Thus [rGURA_1-set, PosCanAdd] is solvable in P (follows trivially from Theorem 5). For $\mathrm{RP}_{\supseteq}$ in [rGURA_1-set, SR, $\overline{D}$], if we take out SR restriction, the complexity increases to NP-complete.

THEOREM 9. $\mathrm{RP}_{\supseteq}$ *for* [rGURA_1*-set,* $\overline{D}$] *is NP-complete.*

NP-hardness is proved through a reduction from role reachability problem in miniARBAC97 policies without revocation which is NP-complete. In addition, The length of any plan is bounded by $\sum_{att\in \mathrm{ATTR}}|\mathrm{SCOPE}_{att}|$ as each attribute value can be added at most once. Any plan can be verified in polynomial time. Interestingly, even if we further loosen the $\overline{D}$ restriction to $\overline{CD}$ (No Conditional Deletion), the complexity remains to be NP-complete. $\overline{CD}$ is defined as follows: the can_delete relation is empty except for a certain set of values for which the delete rules are unconditionally **true** for some administrative roles in AR.

THEOREM 10. $\mathrm{RP}_{\supseteq}$ *for* [rGURA_1*-set,* $\overline{\mathrm{CD}}$] *is NP-complete.*

The proof is borrowed from earlier results of other schemes as shown in [rGURA_1-set, $\overline{D}$]. Details are shown in appendix.

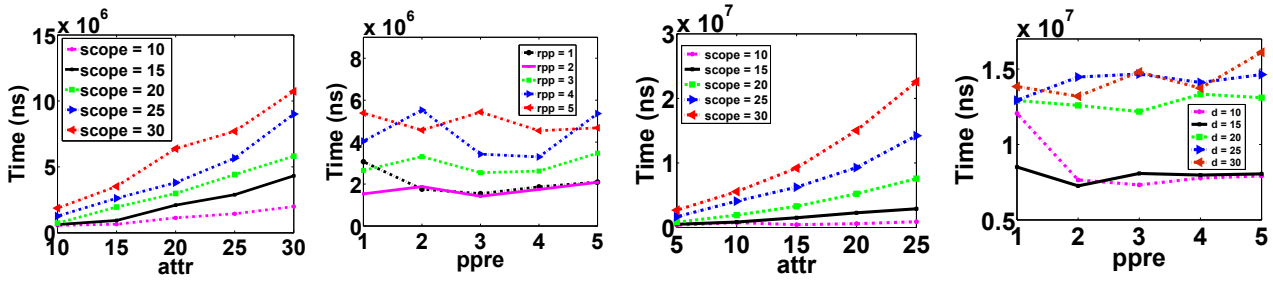
Recall that we assumed the roles are flat in AR. However, the analysis results also apply in hierarchies AR. The rules specified for each administrative role $ar \in \mathrm{AR}_h$ are prorogated to roles which are senior to ar as they are implicitly assigned with ar. The restrictions defined in this paper will not be violated by the above process because no new preconditions are introduced.

6. EXPERIMENTAL RESULTS

This section presents the experiments to evaluate the performance of algorithms 1 and 2. We automate administrative rule generation as follows. There are several parameters: $attr$ represents the number of attributes, $scope$ represents the size of each attribute scope, $ppre$ and $npre$ represent the number of positive and negative conjuncts in a precondition respectively, rpp represents the fixed number of can_add rules for each attribute-value pair. For each randomly generated query, d represents the total number of desired attribute and value pairs specified in the query where the desired values are not already available in the initial state. For instance, suppose that ATTR = {Prj, Skill} and U ={Alice}. In the initial state, Prj(Alice) ={search} and Skill(Alice) ={web}. If the query requires Prj(Alice) ={game, mobile}, Skill(Alice) ={web, system, server}, then d would be 4 which is the number of attribute and value pairs not reached in the initial state. We vary all these parameters to generate instances for both algorithms except that rpp applies to only algorithm 1 and $npre$ applies to only algorithm 2. We generate at least one rule for each attribute and value pair (In practice, it is possible that no rule is specified for some attribute and value pairs). Each data reported is an average over 16 instances generated using the same parameter values. In all 16 problem instances, the query is satisfiable and a plan is returned. Times were measured on a 2.53 GHz dual-core CPU with 2 GB RAM.

Results for Algorithm 1. The results are in figure 3a and 3b. Figure 3a shows the impact of the sizes of $attr$ and $scope$ on execution time. We plot the number of attributes on the x-axis and time consumed on y-axis. We plot a curve for different values of $scope$. In all instances, we use $ppre$ = 5, rpp = 3, d = 20. As expected, the general trend is that the execution time increases with the increase in attribute number and scope size and the change is faster as they become larger. For instance, the execution time for $attr$ = 30 and $scope$ = 30 is nearly 6 times that of when $attr$ = 10 and $scope$ = 30. The major reason is that the total number of rules are increased (recall that we generate at least one rule for each attribute and value pair). However, we believe that the number of the parameters are reasonably small (for example, we do not expect a user to carry 100 attributes) in practical systems and hence the reachability problems can be solved in a very reasonable amount of time.

Figure 3b evaluates the impact of $ppre$ and rpp parameters. We plot $ppre$ on x-axis and time consumed on y-axis. We plot multiple curves for different rpp values. In all problem instances, we use $attr$ = 20, $scope$ = 50 and d = 10. Our result shows that there is no trend of time increase with the size of $ppre$ given the same size of rpp. However, the total time increases with rpp given the same $ppre$. The major reason is that the total number of rules affects the algorithm complexity and it stays the same when rpp remains the same. (The distance between initial state and the final state



(a) Varying $attr$ and $scope$ (Alg 1) (b) Varying $ppre$ and rpp (Alg 1) (c) Varying $attr$ and $scope$ (Alg 2) (d) Varying $ppre$ and d (Alg 2)

Figure 3: Performance Evaluation of Algorithm 1 and Algorithm 2 With Various Parameters

satisfying the query is also a factor. Since we set d to be constant, its impact is not visible here.) Note that since the problem instances are randomly generated, given the same $ppre$, the time for solving the case of a bigger rpp may be lesser than when the rpp is smaller.

Results for Algorithm 2. The results are in figure 3c and 3d. Figure 3c shows the impact of $attr$ and $scope$ on the execution time. The x-axis shows the number of attributes and we plot multiple curves for different values of scope. In all problem instances, we use $npre$ = 1, d = 5 and $ppre$ = 5. The time consumption increases with attribute and scope size and it increase faster with larger $attr$ and $scope$. The major reason is that a larger number of attribute and values pairs may need to be added to satisfy the query.

Figure 3d evaluates the impact of $ppre$ and d. We plot the positive precondition size on x-axis and consider d varying from 10 to 30. In all problem instances, we use $npre$ = 1, $attr$ = 20 and $scope$ = 30. We can see from the figure that the time does not increase significantly with $ppre$ given the same d. However, the time increases with d given the same $ppre$. The major reason is that when the difference between the expected attribute values in the query and the values in the initial state is larger, more attribute and value pairs need to be added (recursive back tracking of can_add relations results in more attribute and value pairs to be added). Again, note that in some instances (e.g. black and pink curves in the figure), the time for a bigger d value is lesser than that when d is higher due to the randomness in generation of our administrative rules.

7. RELATED WORK

The closest category of work is the role reachability analysis for ARBAC97. Li et al [20] presented algorithms and complexity results for analysis of two restricted versions of ARBAC97–AATU and AAR. However, this work did not consider negative preconditions. Sasturkar et al [31] and Jha et al [16] presented algorithms and complexity results for analysis of ARBAC policies subject to a variety of restrictions on how the policy can be specified. Stoller et al [34] proposed the first fixed-parameter-tractable algorithm for analyzing ARBAC policies. However, the algorithm only applies to rules with one positive precondition and unconditional role revocation. Stoller et al [35, 36] analyzed security on parameterized RBAC and ARBAC97. Although the parameters of role can be considered as user attributes, all parameters are treated as atomic-valued and are only changed together with the modification of role. Similar works are [3, 4] which presented symbolic analysis for attribute RBAC models. Our work is fundamentally different from these in consideration of administration of multiple attributes including atomic valued attributes, whereas the ARBAC97 analysis only deals with a single set-valued attribute called role. The second category is policy analysis in attribute based models. Gupta et al [11] proposed rule-based administrative policy model that controls addition and removal of both rules and facts, and a symbolic analysis algorithm for answering reachability queries. The facts of users may be termed as attributes. However, the model does not distinguish atomic and set valued attributes and the current version of the algorithm is incomplete. Li et al [19] discussed security analysis in role based trust management (RT). It is different from our work in that the focus is on delegation and trust. In addition, only one attribute, i.e. role, is considered. Jajodia et al [15] proposed a policy language to express positive and negative authorizations and derived authorizations, and they give polynomial-time algorithms to check consistency and completeness of a policy. [6] showed how to eliminate policy mis-configurations using data mining. [10] presented security constraint patterns for modelling security system architecture and verifying whether required security constraints are correctly enforced. However, this framework facilitates design and deployment of security polices rather than run-time security analysis. [14] developed a graphical constraint expression model to simplify constraints specification and make safety verification practical, but does not ensure polynomial time safety checking.

While there are many works on constraints on policies [2, 9], constraints can help mitigate reachability issues when a security architect can plan ahead and disallow certain values of attributes under certain scenarios. However, reachability analysis is still an important issue since not all scenarios can be planned ahead.

8. CONCLUSION AND FUTURE WORK

This paper presents attribute reachability analysis in rGURA. We formally define the problem and prove that it is in general intractable. Further, we provide restrictions on the precondition specification in administrative rules to show polynomial time solutions. We provide algorithms which determines reachability as well as generates plans for the query. There are considerable future works. The first direction is more polynomial time solutions. For instance, tractable solutions for the scheme rGURA_1-atomic and rGURA_1 remain to be explored. Secondly, the rGURA scheme itself can be extended in many directions. For instance, administrators could be treated as regular users so administrative role is just another user attribute. User attributes are utilized in determining administrative privileges as well as in precondition specification. Precondition in rules could allow specification of other users' attributes thus connecting related users. Thirdly, more kinds of queries can be defined. Except for the examples introduced in other related work (*e.g.,* existence of length-bounded plan), queries can also be specified on the relationships between the attributes of the same user.

Acknowledgment

The authors are partially supported by grants from NSF grant CNS-1111925 and AFOSR MURI grant FA9550-08-1-0265.

9. REFERENCES

[1] OASIS, Extensible access control markup language (XACML), v2.0 (2005).

[2] G. J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM TISSEC*, 3(4):207–226, 2000.

[3] F. Alberti, A. Armando, and S. Ranise. Efficient symbolic automated analysis of administrative attribute-based RBAC-policies. In *ACM ASIACCS*, pages 165–175, 2011.

[4] A. Armando and S. Ranise. Automated and efficient analysis of role-based access control with attributes. *Data and Applications Security and Privacy XXVI*, pages 25–40, 2012.

[5] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[6] L. Bauer, S. Garriss, and M. K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. In *ACM SACMAT*, pages 185–194, 2008.

[7] T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, pages 165–204, 1994.

[8] D.W. Chadwick. Authorisation using attributes from multiple authorities. In *WETICE*, pages 326–331, 2006.

[9] J. Crampton. Specifying and enforcing constraints in role-based access control. In *SACMAT*, 2003.

[10] Y. Deng, J. Wang, J. JP Tsai, and K. Beznosov. An approach for modeling and analysis of security system architectures. *IEEE TKDE*, 15(5):1099–1119, 2003.

[11] P. Gupta, S. D. Stoller, and Z. Xu. Abductive analysis of administrative policies in rule-based access control. In *ICISS*, volume 7093, pages 116–130. Springer-Verlag, 2011.

[12] M. A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Comm. of the ACM*, pages 461–471, 1976.

[13] V. C. Hu and D. Ferraiolo et al. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. In *NIST SP 800-162 DRAFT*, 2013.

[14] T. Jaeger and J. Tidswell. Practical safety in flexible access control models. *ACM TISSEC*, pages 158–190, 2001.

[15] S. Jajodia, P. Samarati, and VS Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on S&P*, pages 31–42, 1997.

[16] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 5(4):242–255, 2008.

[17] X. Jin, R. Krishnan, and R. Sandhu. A role-based administration model for attributes. In *Proceedings of WSARS*, pages 7–12. ACM, 2012.

[18] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *DBSec*, pages 41–55, 2012.

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

[20] N. Li and M.V. Tripunitara. Security analysis in role-based access control. In *ACM SACMAT*, pages 126–135, 2004.

[21] N. Li and W. H. Winsborough. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *IEEE Symposium on S&P*, pages 123–139, 2003.

[22] Ninghui Li, John C Mitchell, and William H Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on S&P*, pages 114–130, 2002.

[23] J. Park and R. Sandhu. The UCON ABC usage control model. *ACM TISSEC*, pages 128–174, 2004.

[24] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters. Secure attribute-based systems. In *ACM CCS*, pages 99–112, 2006.

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

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

[27] R. S. Sandhu. The typed access matrix model. In *IEEE Symposium on S&P*, pages 122–136, 1992.

[28] R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 1993.

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

[30] R. S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Com. Mag.*, 1994.

[31] A. Sasturkar, P. Yang, S.D. Stoller, and CR Ramakrishnan. Policy analysis for administrative role based access control. In *IEEE CSFW*, 2006.

[32] W. J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

[33] C. Schläger, M. Sojer, B. Muschall, and G. Pernul. Attribute-based authentication and authorisation infrastructures for e-commerce providers. In *E-Commerce and Web Technologies*, 2006.

[34] S. D. Stoller, Ping Yang, C R. Ramakrishnan, and Mikhail I. Gofman. Efficient policy analysis for administrative role based access control. In *ACM CCS*, pages 445–455, 2007.

[35] S.D. Stoller, P. Yang, M.I. Gofman, and CR Ramakrishnan. Symbolic reachability analysis for parameterized administrative role-based access control. *Computers & Security*, 30(2):148–164, 2011.

[36] S.D. Stoller, Ping Yang, Mikhail Gofman, and C. R. Ramakrishnan. Symbolic reachability analysis for parameterized administrative role based access control. In *ACM SACMAT*, pages 165–174, 2009.

APPENDIX

Proof for Theorem 1. The following simple algorithm can be used for solving the $\mathrm{RP}_{=}$ query (recall that $\mathrm{RP}_{\supseteq}$ query does not apply to this scheme). The plan to the original query on all attributes can be obtained by combining the plan for satisfying each attribute. Based on this observation, the reachability problem instance I = $\langle \gamma, q \rangle$ for the scheme [rGURA_0-atomic] can be reduced to finding whether, for an attribute $att \in \mathrm{ATTR}$, it is possible to reach a state γ' from γ that satisfies the condition that the value of att is the same as the corresponding value specified in the query q. A directed graph TG = $\langle$V, E$\rangle$ is constructed based on the rules in $\mathsf{can\_assign}_{att}$. In this graph, V = $\mathrm{Range}(att)$. For each val_1 and val_2 in V, an edge $\langle \mathrm{val}_1, \mathrm{val}_2 \rangle$ is added to E if $\langle ar, c, val_2 \rangle \in \mathsf{can\_assign}_{att}$ and $att(u) = val_1$ is a conjunct in c. A query on att is equivalent to a path search problem between the corresponding two nodes in the graph which can be solved using well-known search algorithms such as depth first search (DFS). It is straight forward to generate a plan if the target value is reachable.

We assume that DFS is used. (1) **Correctness.** The algorithm is well-known to be correct. (2) **Complexity.** We first discuss the complexity for querying a single attribute $att \in \mathrm{ATTR}$. The graph can be created by traversing each rule once. Each rule adds at most

$|\mathrm{Range}(att)|$ edges to the graph. The complexity of DFS on a graph $\langle \mathrm{V}, \mathrm{E}\rangle$ is O(|V| + |E|). Thus, the complexity for solving $\mathrm{RP}_{=}$ for attribute att is O($|\mathrm{Range}(att)| \times |\mathsf{can\_assign}_{att}|$). For problem instances containing |ATTR| attributes, the overall complexity is O($|\mathrm{ATTR}| \times |\mathrm{Range}(att)| \times |\mathsf{can\_assign}_{att}|$), where att has the largest $|\mathrm{Range}(att)| \times |\mathsf{can\_assign}_{att}|$ amongst all attributes.

Proof for Theorem 2. Our proof shows the reduction that the $\mathrm{RP}_{\supseteq}$ problem for [rGURA_0-set] is at least as hard as the role reachability problem in miniARBAC97 which is PSPACE-complete [31]. Its problem instance can be understood as a 3-tuple $\langle \gamma, goal, \psi\rangle$ where γ is an initial state with role assignments for a particular user, $goal$ is the desired set of roles for that user in some future state and ψ is a set of administrative rules that guides user-role assignments by a set of administrative roles. The reachability question asks whether a given set of administrative roles AR can act with the permissions associated with their roles in ψ and transition γ to a future state γ' such that the desired role assignments specified in $goal$ for a particular user is satisfied in γ'. In addition to can_assign and can_revoke relations for roles, the miniARBAC97 also considers SMER (Static Mutually Exclusive Roles) constraints which represents a set of conflicting roles that cannot be assigned to the same user at any time.

The core idea of our construction is to treat roles in miniARBAC97 as a user attribute called role. We assume that SMER = $\emptyset$ since it can be expressed in can_add rules using negative preconditions [31]. The scope of the role attribute is the same as the set of roles in miniARBAC97. It is straight-forward to specify each of the can_assign and can_revoke rules in miniARBAC97 using corresponding $\mathsf{can\_add}_{att}$ and $\mathsf{can\_delete}_{att}$ rules in [rGURA_0-set] since the pre-condition grammar of rGURA_0 is similar to that of miniARBAC97. User-role assignment in the initial state in miniARBAC97 can be mapped to attribute assignment in [rGURA_0-set] and the query can be specified. The reduction process takes O($|\gamma| + |goal| + |\psi|$). For a problem instance containing |ATTR| number of attributes, the total complexity is the sum of complexity of reduction for each attribute which is polynomial. This establishes that $\mathrm{RP}_{\supseteq}$ for [rGURA_0-set] is PSPACE-hard.

Proof for Theorem 3. Per lemma 1, it suffices to show PSPACE-hardness. We use the result from SAS planning problem [5]. An instance of SAS planning problem is a tuple $\langle \mathcal{V}, \mathcal{O}, \mathrm{s}_0, \mathrm{s}_*\rangle$, where $\mathcal{V}$ represents a finite set of state variables with pre-specified domains for each variable, $\mathcal{O}$ represents a finite set of operators, s_0 and s_* represent initial and goal states and they are both total states (i.e., each variable is assigned with a value from its domain). An operator $\langle$pre, prv, post$\rangle$ updates state variables in post if the conditions pre and prv are satisfied in the current state. The conditions pre, prv and post are members of partial state space (state variables are allowed to be unspecified). The problem is given an initial state s_0, does there exist a sequence of operators (a plan) which transition s_0 to s_*? The plan-existence for the SAS planning problem under U (each operator changes only a single state variable) and B (boolean domain for state variables) restrictions is PSPACE-complete [5].

We show that $\mathrm{RP}_{=}$ for [rGURA_0-set] is at least as hard as the [SAS planning, U, B] problem. As earlier, we consider the complexity of reachability of one attribute $att \in \mathrm{ATTR}$ independent of others. The reduction is as follows, given any SAS planning problem satisfying U and B. (1) Each state variable is mapped to one value in the scope of att. Thus, the scope of att is a set of values whose size is the same as $\mathcal{V}$. In each state, if a state variable is set to **true**, the corresponding value is added to the attribute att. Thus, s_0 is specified using attribute assignment of att and s_* is specified as a query. (2) The operator which updates a state variable to **true** is mapped to one rule in $\mathsf{can\_add}_{att}$ and the operator which sets a state variable to **false** is mapped to one rule in $\mathsf{can\_delete}_{att}$ (att is mapped to the state variable in the operator). A precondition in an operator can be specified as the precondition in each administrative rule. The complexity of the reduction process is O($|\mathcal{V}| + |\mathcal{O}|$). This establishes that $\mathrm{RP}_{=}$ for [rGURA_0-set] is PSPACE-hard.

Proof for Theorem 6. We use the result from STRIPS planning problem [7]. An instance of STRIPS planning problem is 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 (each take the value **true** or **false**), $\mathcal{O}$ is a finite set of operators pre $\Rightarrow$ post, where post updates the conditions to either positive or negative if pre is satisfied. The pre and post are satisfiable conjunctions of positive and negative conditions. Any state can be specified by a subset of $\mathcal{P}$, indicating that each element in the subset is **true** and all others are **false** in the state. $\mathcal{G}$ called $goal$ is a satisfiable conjunction of positive and negative conditions. $\mathcal{S}$ is a goal state if $\mathcal{S}$ all positive conditions in goal is in $\mathcal{S}$ and none of the negative conditions in goal appears in $\mathcal{S}$. STRIPS planing explores a sequence of operators which transition the initial state $\mathcal{I}$ to a state in which $\mathcal{G}$ is satisfied. PLANSAT is defined as determining whether an instance of STRIPS planing is satisfiable.

[7] shows that [PLANSAT, $*$+preconds, 1 postcond] is polynomial time solvable . Here, only positive preconditions are allowed and each operator only modifies one condition, setting it as either positive or negative. We show the reduction: plan existence in [PLANSAT, $*$+ preconds, 1 postcond] is at least as hard as $\mathrm{RP}_{=}$ in [rGURA_1-Set, $\overline{N}$]. The reduction is as follows. Given any [rGURA_1-set, $\overline{N}$] scheme: (1) each attribute and value pair $(att, value)$ is mapped to a corresponding condition; (2) to specify a state in [rGURA_1-set], for each attribute and value pair, the corresponding condition is set to true. To specify a query in [rGURA_1-set], for each attribute and value pair in the query, the corresponding condition is set to true. For all other attribute and value pairs not in the query, their corresponding conditions are set to false. This ensures the query is only satisfiable with exact the same value for each attribute; (3) each rule in $\mathsf{can\_add}_{att}$ is specified as a positive operator which updates the corresponding condition for the specified attribute and value pair. The precondition is specified as pre, the value to be added is specified in post; and finally (4) each rule in $\mathsf{can\_delete}_{att}$ rule is specified as a negative operator. The reduction process takes O($(\sum_{att \in \mathrm{ATTR}} |\mathrm{SCOPE}_{att}|)$+$|\Psi|$) where $|\Psi|$ is the number of all administrative rules (Ψ contains only can_add and can_delete relations) which is polynomial.

Proof for Theorem 10. We assume that DV = $\{(att, val) \mid att \in \mathrm{ATTR} \wedge val \in \mathrm{SCOPE}_{att}\}$ is a set of attribute values that can be deleted without preconditions. If any attribute value appears in any of the preconditions as negative conjuncts, it is safe to remove it from the precondition of those rules if it is also in DV for the purpose of our analysis. Members of administrative roles can delete the values at any time for all users, it is no need to specify them in preconditions. We pre-process all rules in can_add. There are two situations when the pre-process is finished: (1) if there are no negation in can_add rules, the problem is equivalent to $\mathrm{RP}_{\supseteq}$ in [rGURA_1-Set, PosCanAdd] and it has been shown earlier to be solvable in P; (2) if negation exists in the preconditions of some of the rules in can_add, the problem is then equivalent to $\mathrm{RP}_{\supseteq}$ in [rGURA_1-set, $\overline{D}$]. The complexity is proved to be NP-complete in theorem 9.