Reachability Analysis for Attributes in ABAC With Group Hierarchy

Maanak Gupta, *Member, IEEE*, Ravi Sandhu, *Fellow, IEEE*, Tanjila Mawla, and James Benson

Abstract—Attribute-based access control (ABAC) models are widely used to provide fine-grained and adaptable authorization based on the attributes of users, resources, and other relevant entities. Hierarchical group and attribute based access control (HGABAC) model was recently proposed which introduces the novel notion of attribute inheritance through group membership. GURA_G was subsequently proposed to provide an administrative model for user attributes in HGABAC, building upon the ARBAC97 and GURA administrative models. The GURA model uses administrative roles to manage user attributes. The reachability problem for the GURA model is to determine what attributes a particular user can acquire, given a predefined set of administrative rules. This problem has been previously analyzed in the literature. In this article, we study the user attribute reachability problem based on directly assigned attributes of the user and attributes inherited via group memberships. We first define a restricted form of GURA_G, called rGURA_G scheme, as a state transition system with multiple instances having different preconditions and provide reachability analysis for each of these schemes. In general, we show PSPACE-complete complexity for all rGURA_G schemes. We further present polynomial time algorithms with empirical experimental evaluation to solve special instances of rGURA_G schemes under restricted conditions.

Index Terms—Access control, ABAC model, reachability analysis, group hierarchy, attributes inheritance, attributes administration

1 Introduction

Attribute-based access control (ABAC) is considered as an important authorization system among practitioners and researchers. The system offers fine-grained and adaptable access control solutions based on the characteristics, referred to as attributes, of several entities. ABAC systems provide a flexible and scalable approach to secure resources in distributed environments and overcome some of the shortcomings of traditional discretionary access control (DAC)[1], mandatory access control (MAC) [2] and role based access control (RBAC) [3] models. Several attribute based access control models have been formulated [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21] but a strong consensus on its definitive characteristics is still to be achieved. More recently, hierarchical group and attribute based access control model (HGABAC) [22] was proposed, which introduced the notion of user and object groups to assign attributes to users and objects respectively. In this model, besides the direct assignment of attributes to users or objects, groups are also assigned attributes, which are then assigned to users and objects through corresponding group memberships. The most important advantage of this model is the ease of administration, since multiple attributes can be assigned or removed from users or objects through single administrative operation. The administrative model for HGABAC, referred as GURA_G, was defined in [23], to control user attribute assignment based on specified precondition rules and administrative roles. This model has three sub-models user attribute assignment (UAA), user group attribute assignment (UGAA) and user to user-group assignment (UGA) which assigns attributes to users directly or indirectly through groups. The model extends well-known ARBAC97 [24] administrative model and recently published GURA administrative model [25] by introducing administration of attributes for user-groups and managing user to groups memberships.

In ABAC, the attributes of an entity are critical in determining its permissions. Therefore, it is an important question to compute the attribute values that an entity can acquire through the combination of administrative roles and rules. In the context of GURA_G, it is imperative to understand the set of attribute values a user can get based on direct assignment or via group memberships. Group hierarchy also exists in the HGABAC operational model which further complicates computation of the possible effective attribute values of a user. Although security administrators are trusted to assign attributes correctly, it is still desirable to understand the eventual set of attribute values that a user can acquire through multiple direct and indirect assignments. Such analysis can also help to identify a sequence of administrative actions required by administrators to assign certain attribute values to the users. It further allows administrators to know the future attribute values an entity can achieve based on predefined administrative rules, which can help them to understand if certain permissions can ever be granted to an entity.

As the number of attributes, attribute values and administrative rules become large, certain anomalies become hard

- *Maanak Gupta and Tanjila Mawla are with the Department of Computer Science, Tennessee Tech University, Cookeville, TN 38501 USA. E-mail: {mgupta, tmawla42}@tntech.edu.*
- *Ravi Sandhu and James Benson are with the Institute for Cyber Security and Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249 USA. E-mail: {ravi.sandhu, james.benson}@utsa.edu.*

Manuscript received 19 January 2021; revised 24 November 2021; accepted 17 January 2022. Date of publication 25 January 2022; date of current version 16 January 2023.
This work was supported by NSF under Grants HRD-1736209 at UTSA and 2025682 at TTU.
(Corresponding author: Maanak Gupta.)
Digital Object Identifier no. 10.1109/TDSC.2022.3145358

to detect just by simple inspection. For example, suppose an administrative user having role RoomAdmin is allowed to add user attribute roomAcc with value 1.02 to a user only if the user's attribute status has value Grad and the user does not currently have roomAcc 2.01. Further, a user can be assigned status attribute with value Grad only through group G_1 membership since there is no direct assignment administrative rule for status attribute. Another administrative rule assigns roomAcc 2.01 to a group junior to G_1, thereby getting G_1 with roomAcc value 2.01. Now if a user is assigned to user-group G_1, she will get all G_1's attributes, including roomAcc with value 2.01. It might seem that user will not be able to get roomAcc value 1.02 and 2.01 together. However, it is possible if the junior group to G_1 is assigned value 2.01 after the user is assigned to user-group G_1. Such security policy anomalies can be discovered with the help of reachability analysis, which checks if entities can get certain values together or whether the entity will get particular values based on the set of administrative rules defined through administrative models.

In this paper we analyze the attribute reachability analysis focusing on the effective attributes of the user achieved through direct assignment and through user-group memberships. This work extends the reachability analysis [26] done for GURA administrative model [25], where the attributes were only directly assigned to users without the concept of group memberships. In our analysis, we have defined a restricted $\mathrm{GURA_G}$ model, called $\mathrm{rGURA_G}$, which considers a subset of preconditions which can be created in $\mathrm{GURA_G}$. We abstract $\mathrm{rGURA_G}$ into a state transition system and specify three separate instances—$\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_{1+}}}$—to cover different set of prerequisite conditions for attributes assignments to a user or a group, and also for user to group membership assignment. Our reachability analysis primarily focuses on the effective set of attributes of users which is the union of direct attributes and attributes attained by group membership. We have defined reachability queries which is the required set of effective attributes a user can achieve in any target state. Two different types of reachability queries are discussed, one with the exact values and another with the superset of attribute values. We will show that the general reachability problem for $\mathrm{rGURA_G}$ schemes is PSPACE-complete. We further identify certain more restricted cases of $\mathrm{rGURA_G}$ schemes where the reachability problem can be solved in polynomial time. For such instances we will provide algorithms and a sequence of administrative requests (referred as reachability plan) to satisfy the reachability query.

The rest of this paper is organized as follows. Section 2 reviews the related work. In Section 3, we review the HGABAC model and $\mathrm{GURA_G}$ administrative model. Section 4 discusses the generalized restricted $\mathrm{rGURA_G}$ scheme and its instances. In Section 5, we formally define our user attribute reachability problem. Formal proofs for general $\mathrm{rGURA_G}$ schemes are discussed in Section 6. Section 7 presents polynomial algorithms for some restricted versions of $\mathrm{rGURA_G}$ schemes followed by example problems instances in Section 8 and experimental results in Section 9. Section 10 concludes this paper.

2 Related Work

Reachability analysis for user attributes was first studied by Jin *et al.* [26], based on the GURA administrative model [25]. In this analysis, attribute values are assigned to users directly based on certain attribute-based prerequisite conditions and by administrators assuming roles. This work proves PSPACE-complete complexity for generalized GURA scheme and also presents polynomial algorithms for some conditional cases. Our work extends the aforementioned reachability analysis where attributes are assigned to users as well as to groups to which users are members. This assignment of attributes to groups provides administrative benefits in addition and removal of multiple attributes to users with a single administrative operation.

Security policies have been widely analysed in several works including [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37]. The safety analysis problem goes back to 1970's. In general, the safety of access control matrix (ACM) model was shown to be undecidable in [27]. Tripunitara and Li presented an important theoretical comparison of expressive powers of different access control models in [28]. Many of our notations in this paper are adapted from this work. The same authors in [29] defined restricted forms of ARBAC97 (AATU and AAR) and provided algorithms for analysis problems including safety and availability in restricted forms. This work extends results from trust management policies in [30] where safety and availability security analysis on delegation of authority is discussed. The schematic protection model (SPM) [31] introduced typed security entities where each entity is associated with a security type, which remains unchanged. Sasturkar *et al.* [32] analyse ARBAC97 administrative policies to determine reachability and availability problems, by establishing connections between artificial intelligence planning problem. Jha *et al.* [36] classified analysis problems related to RBAC and claimed PSPACE-complete solutions for unrestricted classes whereas NP-complete and polynomial time algorithms for restricted subclasses. Lipton *et al.* [34] presented a linear time algorithm for take and grant system. Alloy language is used for specification of role based system and analysis is done using Alloy constraint analyser in [35]. Recently, Rajkumar and Sandhu discussed safety problem for pre-authorization sub-model for $\mathrm{UCON_{ABC}}$ in [37].

Jajodia *et al.* [38] presented a logical language to express positive, negative and derived authorization policies, and provided polynomial algorithms to check completeness and consistency. Cholvy and Cuppens [39] discussed the problem of policy consistency and offered a methodology to solve it. They further suggested the use of roles priorities to resolve normative conflicts in policies. [40] provides a method to transform policy specifications into event calculus based formal notation. It further describes the use of abductive logical reasoning to perform a priori analysis of various policy specifications. Jaeger *et al.* [41] presented the concept of access control space and its use in managing access control policies. These spaces are used to represent permission assignment to subjects or roles. Authors in [42] presented decision diagram based algorithms to analyze XACML based policies and compute the semantic differencing information between versions of policies. Stoller *et al.* [43] provided algorithms for

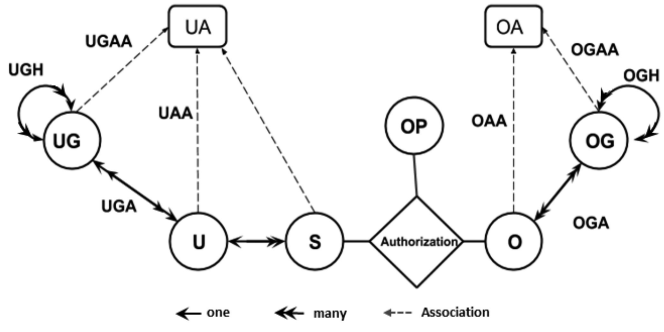


Fig. 1. HGABAC conceptual model.

ARBAC97 policies limited to rules with one positive precondition and unconditional role revocations. Same authors in [44] defined PARBAC (parameterized ARBAC) and determined user-reachability problem as undecidable over an infinite types of parameter. It further assumed all parameters as atomic-valued and are changed when the role is modified. Gupta *et al.* [45] discussed rule-based administrative model to control addition and removal of facts (attributes) and rules. It further proposed an abductive algorithm which can analyse policies even when the facts (attributes) are unavailable based on computation of minimal sets of facts. The work in [46] provides analysis of expressive power of generalized temporal role-based access control (GTRBAC) which offers a set temporal constraints to specify fine grained time based policies.

Several works [23], [24], [25], [47] have been presented to discuss administrative models for well known access control models. ARBAC97 [24] discusses the user to role assignment based on the administrative rules comprising of administrative roles and prerequisite conditions based on roles. The $\mathrm{GURA_G}$ administrative model [23] provides a generalized administrative model for attributes based access control models by asserting role as one of the several user attributes. These works define attribute based preconditions and administrative roles to assign and remove attributes from users and groups. Crampton and Loizou [47] also presented an administrative work related to RBAC model and developed models for role hierarchy administration.

3 Background

In this section, we will provide an overview of reformalized hierarchical group and attribute based access control (HGABAC) model. We will further discuss the $\mathrm{GURA_G}$ model [23] and its three sub models user attribute assignment (UAA), user-group attribute assignment (UGAA) and user to user-group assignment (UGA). The main objective of this section is to lay the foundation of our reachability analysis and make the reader familiar with relevant terminologies and concepts.

3.1 HGABAC Model

This subsection discusses the reformalized HGABAC model as defined in [23]. We have formulated this model in style of ABAC_α [4] to help in our administrative model and reachability analysis. The model is notationally different but equivalent to HGABAC model provided by Servos *et al.* [22]. We begin with an informal overview of the model

TABLE 1
HGABAC Formal Model (User Attributes Only)

Basic Sets and Relations

– U, S (finite sets of users and subjects respectively).
– UG (finite set of user groups).
– UA (finite set of user attribute functions).
– For each $att \in \mathrm{UA}$, SCOPE_{att} is a finite set of atomic values and $\mathrm{Range}(att) = \mathcal{P}(\mathrm{SCOPE}_{att})$ where $\mathcal{P}$ denotes the powerset.
– $\mathrm{UGH} \subseteq \mathrm{UG} \times \mathrm{UG}$, a partial order relation $\succeq_{ug}$ on UG.

Defined (Direct) Functions

– For each $att \in \mathrm{UA}$. att: $\mathrm{U} \cup \mathrm{UG} \rightarrow \mathrm{Range}(att)$, maps each user and user group to a subset of values in SCOPE_{att}.
– directUg : $\mathrm{U} \rightarrow 2^{\mathrm{UG}}$, maps each user to a subset of user groups in UG.

Derived (Effective) Functions

– effUg : $\mathrm{U} \rightarrow 2^{\mathrm{UG}}$, defined as
$$\mathrm{effUg(u)} = \mathrm{directUg(u)} \cup \Big(\bigcup_{\mathrm{ug_i} \in \mathrm{directUg(u)}} \{\mathrm{ug_j} \mid \mathrm{ug_i} \succeq_{ug} \mathrm{ug_j}\}\Big).$$
– For each $att \in \mathrm{UA}$,
- effUG_{att} : UG $\rightarrow$Range(att), defined as
$$\mathrm{effUG}_{att}(\mathrm{ug_i}) = att(\mathrm{ug_i}) \cup \Big(\bigcup_{\mathrm{g} \in \{\mathrm{ug_j} \mid \mathrm{ug_i} \succeq_{ug} \mathrm{ug_j}\}} \mathrm{effUG}_{att}(\mathrm{g})\Big).$$
- effU_{att} : $\mathrm{U} \rightarrow \mathrm{Range}(att)$, defined as
$$\mathrm{effU}_{att}(\mathrm{u}) = att(\mathrm{u}) \cup \Big(\bigcup_{\mathrm{g} \in \mathrm{directUg(u)}} \mathrm{effUG}_{att}(\mathrm{g})\Big).$$

followed by formal definitions of components of HGABAC relevant to our reachability analysis.

3.1.1 Model Overview

Fig. 1 shows the conceptual HGABAC model. The basic components include traditional access control entities like Users (U), Objects (O), and Subjects (S). A user is a human being interacting directly with a computer whereas subject is an active entity (like an application or a process) created by the user to access resources or objects. A user can create multiple subjects but each subject must belong to a single user. OP represents the set of operations which can be performed by subjects on objects. The novel approach introduced by HGABAC model is the notion of user groups (UG) and object groups (OG), which are a collection of users or objects respectively. The set of user and object attributes is defined by UA and OA respectively. Each attribute in set UA and OA is a set-valued function, which takes different entities, like users, objects, user-groups or object-groups, and return values from the attribute range. As the attributes are assigned to groups also, the prime advantage of this assignment is the inheritance of attributes to the group's user or object members. For example, if a user-group ug with attribute skills having values c and java, is assigned to user u, then u will inherit attribute skill with values c and java from ug. Group hierarchy also exists in HGABAC (defined using a partial relation and shown as self loops in Fig. 1) where senior groups inherit all the attributes from their junior groups. For example, suppose a junior group to ug, say ug', is assigned value c++ for attribute skill, then ug will inherit this value and its effective values for skill will be c, java and c++. In this case user u already assigned to user group ug will get all three values for skill attribute. Similar assignments can be done for object and object groups also. It should be noted that each user or object can be assigned to multiple user or object groups and vice versa. A subject inherits all or subset of the effective attributes of the creator user. Each operation $op \in \mathrm{OP}$ will have an associated boolean authorization function which specifies the policies under which a

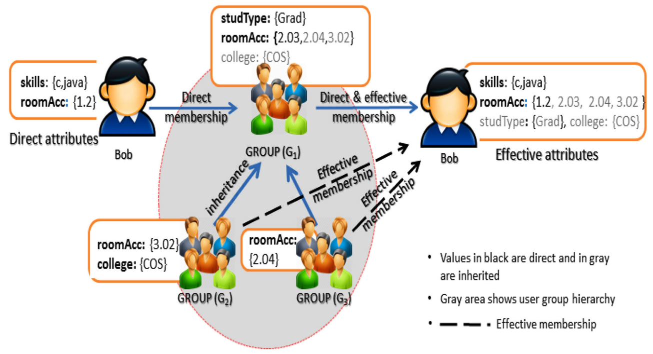


Fig. 2. Example user and user group attributes.

subject is allowed to perform operation *op* on the objects. These policies are specified as propositional logic formulas using the model's policy language and are defined by the security architect at the time of system creation. A subject is allowed to perform operations on an object if the effective attributes of subjects and objects satisfy the boolean authorization function.

3.1.2 Formal Definitions (User Attributes Only)

The GURA_G administrative model [23] deals with the user side of HGABAC model reflecting the administrative relations for users and user groups to modify their attributes. Similar administrative model can also be extended for objects but is out of the scope of the paper. Our reachability analysis also considers only the effective attributes of the user, and therefore, we will only formalize the relevant sets, relations and functions pertinent to HGABAC and required in our analysis. Table 1 defines the formal HGABAC model covering the required definitions. An example configuration with respect to these definitions is shown in Fig. 2 and Table 2.

Basic sets and relations as shown in Table 1 include U, S and UG representing the set of users, subjects and user groups in the system. UA represents the set of user attribute functions for user and user groups where each attribute function in UA is set valued. These attribute functions can assign values to user or user groups from the set of atomic values, represented as SCOPE_{att}. The power set of SCOPE_{att} is defined by $\text{Range}(att)$. Example definitions for these sets is shown in first part of Table 2. User group hierarchy (UGH) is a partial order relation on UG, defined as $\succeq_{ug}$, where $ug_1 \succeq_{ug} ug_2$ represents ug_1 is senior to ug_2 or ug_2 is junior to ug_1. As shown in gray the area of Fig. 2, UGH = $\{(\text{G}_1, \text{G}_1), (\text{G}_2, \text{G}_2), (\text{G}_3, \text{G}_3), (\text{G}_1, \text{G}_2), (\text{G}_1, \text{G}_3)\}$. This UGH relation results in inheritance of attributes from junior to senior group (will discuss in a moment).

Attribute values can be directly assigned to user and user groups which is denoted by function *att* in UA. As defined in Fig. 2 and Table 2, user Bob is directly assigned {c, java} for attribute function skills. Similarly, other direct attributes are given for Bob and user groups $\text{G}_1, \text{G}_2, \text{G}_3$. The function directUg specifies the user groups to which the user is directly assigned. In our example, Bob is directly assigned to user group G_1. We also define the effective user groups of the user (denoted by effUg), which states all the groups to which the user is either directly or indirectly assigned via UGH relation. Effective user group for Bob will be $\{\text{G}_1, \text{G}_2, \text{G}_3\}$, since Bob is directly assigned to G_1 and G_1 has junior groups as G_2 and G_3.

TABLE 2
Example Configuration as Defined in Fig. 2

Basic Sets and Relations
– U = {Bob}, UG ={$\text{G}_1, \text{G}_2, \text{G}_3$}.
– UA = {skills, roomAcc, studType, college}.
– SCOPE of each att in UA, denoted by SCOPE_{att}:
studType = {Grad, UnderGrad}, college = {COS, COE, BUS}
skills = {c, c++, java}, roomAcc = {1.2, 2.03, 2.04, 3.02}.
– UGH is given in Figure 2, highlighted in gray area.
Direct Attributes
skills(Bob) = {c, java}, roomAcc(Bob) = {1.2},
roomAcc(G_2) = {3.02}, college(G_2) = {COS},
roomAcc(G_3) = {2.04}, studType(G_1) = {Grad},
roomAcc(G_1) = {2.03}.
Direct User Groups
directUg(Bob) = {G_1}.
Effective User Groups
effUg(Bob) = {G_1, G_2, G_3}
Effective User Group Attributes
$e\_\text{roomAcc}(\text{G}_2)$ = {3.02}, $e\_\text{college}(\text{G}_2)$ = {COS},
$e\_\text{roomAcc}(\text{G}_3)$ = {2.04}, $e\_\text{studType}(\text{G}_1)$ = {Grad},
$e\_\text{roomAcc}(\text{G}_1)$ = {2.03, 2.04, 3.02}, $e\_\text{college}(\text{G}_1)$ = {COS}.
Effective User Attributes
$e\_\text{skills}(\text{Bob})$ = {c, java},
$e\_\text{roomAcc}(\text{Bob})$ = {1.2, 2.03, 2.04, 3.02},
$e\_\text{studType}(\text{Bob})$ = {Grad}, $e\_\text{college}(\text{Bob})$ = {COS}.

The effective values of an attribute *att* (effUG_{att}) for a user group is the union of the user's direct attribute values and the effective values of all its junior groups in UGH relation. Note that this definition is well formed since $\succeq_{ug}$ is a partial order. For the minimal groups ug_j in this ordering, we have $\text{effUG}_{att}(ug_j) = att(ug_j)$, giving us base cases for this recursive definition. For simplicity, we defined $e\_att(ug)$ = $\text{effUG}_{att}(ug)$ for $ug \in \text{UG}$. Therefore, for attribute roomAcc, effective values for user group G_2 is $e\_\text{roomAcc}(\text{G}_2)$ = {3.02}. This value is same as its direct value for roomAcc attribute, since G_2 has no junior group in UGH. For user group G_1, $e\_\text{roomAcc}(\text{G}_1)$ = {2.03, 2.04, 3.02} as it inherits values from G_2 and G_3. The function effU_{att} maps the user to the effective values for attribute *att*, which is the union of its direct values and the effective values of *att* for all its direct groups. For convenience we defined $e\_att(u) = \text{effU}_{att}(u)$ for user $u \in$ U and as shown in Table 2, the effective values for attribute roomAcc for user Bob, written as $e\_\text{roomAcc}(\text{Bob})$ = {1.2, 2.03, 2.04, 3.02} which is the union of its directly assigned value for roomAcc and values inherited from group G_1. Similarly other effective attributes for user Bob can be calculated. The prime benefit of HGABAC model, which is easy assignment of multiple attributes to a user with user group memberships, is reflected in this function where user u is assigned multiple attributes with direct group membership of G_1.

Subject s $\in$ S created by the user u $\in$ U will then assume a subset of all effective attributes of user u. Similar effective attributes can be assigned to objects, which is out of scope of our reachability analysis and is not discussed. Authorization policies are pre-defined in the system, using propositional logic formula, for each operation in OP (set of operations) by security administrators, which determine if a subject is allowed to perform operations on objects, based on their effective attributes.

Note. HGABAC only allows set-valued attributes. ABAC models generally allow set-valued as well as atomic-valued

TABLE 3
Administrative Requests

In the following requests: $ar \in \text{AR},\ att \in \text{UA},\ val \in \text{SCOPE}_{att},\ u \in \text{U},\ ug \in \text{UG}$
– For User Attributes $\textsf{add}(ar,\ u,\ att,\ val)$ $\textsf{delete}(ar,\ u,\ att,\ val)$ – For User Group Attributes $\textsf{add}(ar,\ ug,\ att,\ val)$ $\textsf{delete}(ar,\ ug,\ att,\ val)$ – For User to User-Group Membership $\textsf{assign}(ar,\ u,\ ug)$ $\textsf{remove}(ar,\ u,\ ug)$

TABLE 4
GURA_G Administrative Model

– User Attribute Assignment (**UAA**):
For each att in UA,

$$\textsf{canAddU}_{att} \subseteq \text{AR} \times \text{C} \times \text{SCOPE}_{att}$$
$$\textsf{canDeleteU}_{att} \subseteq \text{AR} \times \text{C} \times \text{SCOPE}_{att}$$

– User Group Attribute Assignment (**UGAA**):
For each att in UA,

$$\textsf{canAddUG}_{att} \subseteq \text{AR} \times \text{C} \times \text{SCOPE}_{att}$$
$$\textsf{canDeleteUG}_{att} \subseteq \text{AR} \times \text{C} \times \text{SCOPE}_{att}$$

– User to User Group Assignment (**UGA**):

$$\textsf{canAssign} \subseteq \text{AR} \times \text{C} \times \text{UG}$$
$$\textsf{canRemove} \subseteq \text{AR} \times \text{C} \times \text{UG}$$

attributes (for example [4]). Inheritance of values via group membership for an atomic-valued attribute is problematic since such attributes can have only one value. Hence, while the GURA administrative model allows both atomic and set valued attributes the HGABAC only allows set values.

3.2 GURA_G Administrative Model

The GURA_G administrative model [23] was proposed to regulate the assignment of user attribute values in HGABAC model via direct user attributes, user-group attributes and user to group memberships. For convenience we understand the term "assignment of attributes" to mean "assignment of attribute values." The model is inspired by ARBAC97 [24] and GURA [25] administrative models, where administrative roles and current attributes of user and groups or user to group memberships are considered to make future attributes or groups assignments. Administrative role hierarchy also exists in the system where senior administrator roles inherit permissions from junior roles. The GURA_G model has three sub models (shown in Fig. 1): user attribute assignment (UAA), user group attribute assignment (UGAA) and user to group assignment (UGA), which regulates the direct and effective attributes of users. It should be noted that user group hierarchy (UGH) is considered fixed in the system and is not modified. Each of these sub models have different sets of administrative relations and preconditions definition using policy language as discussed in following subsections.

The main difference between GURA and GURA_G is that GURA_G includes the assignment of attributes to groups and user to group memberships. Further, the prerequisite conditions specified in GURA_G are more expressive, as it also checks the current effective attributes or effective group memberships of entities to make future assignments.

3.2.1 Administrative Requests

Definition 1 (Administrative Requests). *The attributes and group memberships of entities are changed by administrative request made by administrators with certain administrative roles as defined in Table 3, where AR is the finite set of administrative roles. The administrative request* $\textsf{add}(ar,\ u,\ att,\ val)$ *is made by administrator with role* ar *to add value* val *to attribute* att *of user* u. *Similar administrative requests are used for groups also. Administrative requests* assign *and* remove *are required for managing group memberships. Each administrative request can add or delete a single attribute value from a user or group.*

3.2.2 Administrative Rules

Definition 2 (Administrative Rules). *Administrative rules are tuples in administrative relations which specify conditions under which administrative requests are authorized. Each of the three sub-models (UAA, UGAA, UGA) in* GURA_G *model have administrative relations to define these rules.*

The UAA sub-model deals with addition or deletion of attributes from the user. It has two administrative relations shown in Table 4, where a rule $\langle ar,\ c,\ val\rangle \in \textsf{canAddU}_{att}$ *authorizes request* $\textsf{add}(ar,\ u,\ att,\ val)$ *if user* u *satisfies precondition* c. *Similarly, rule* $\langle ar,\ c,\ val\rangle \in \textsf{canDeleteU}_{att}$ *authorizes* $\textsf{delete}(ar,\ u,\ att,\ val)$ *requests if user* u *satisfies precondition* c. *In UAA, the precondition* $c \in \text{C}$ *includes only current direct and effective attributes of user* u. *Similar relations also exist for administering attributes of user groups as discussed in sub-model UGAA. In UGAA,* $c \in \text{C}$ *involves current direct or effective attributes of the group whose attributes are modified.*

The UGA sub-model has two relations shown in lower part of Table 4. The rule $\langle ar,\ c,\ ug\rangle \in \textsf{canAssign}$ *authorizes user to group assignment request* $\textsf{assign}(ar,\ u,\ ug)$ *if user* u *satisfies the precondition* c. *Similarly rule* $\langle ar,\ c,\ ug\rangle \in \textsf{canRemove}$ *authorizes remove request* $\textsf{remove}(ar,\ u,\ ug)$ *if user* u *satisfies precondition* c. *The precondition* $c \in \text{C}$ *involves both current direct or effective attributes and groups of user* u.

The expressive power of the GURA_G *model is primarily determined by the richness of the policy language used to define the preconditions C in Table 4. The most general language for this purpose is defined in [23], similar to the most general language of [25] (but without atomic attributes).*

Note. *In the original* GURA_G *definition [23], the administrative relations of Table 4 are defined with* $2^{\text{SCOPE}_{att}}$ *substituted for* SCOPE_{att} *and* 2^{UG} *substituted for* UG. *With the modification of Table 4 the administrative relations can grow linearly in the size of* SCOPE_{att} *and* UG. *This does not materially impact the complexity analysis of the reachability problem.*

3.2.3 GURA_G Scheme

For purpose of our reachability analysis, we express the GURA_G model according to the notations developed in [28], following the treatment in [26]. The GURA_G scheme is presented as a state transition system where each state consists of direct attribute assignments for each attribute of every user and group, and also each user to groups membership. A

TABLE 5
Transition Function

(1) γ_1 and γ_2 are the source and target states respectively.
(2) Let : $ar \in \mathrm{AR},\ u \in \mathrm{U},\ ug \in \mathrm{UG},\ att \in \mathrm{UA},\ val' \in \mathrm{SCOPE}_{att},\ ug' \in \mathrm{UG}$.
(3) $\mathrm{Satisfy}_u$: $\mathrm{U} \times \mathrm{C} \times \Gamma \rightarrow \{\mathbf{true}, \mathbf{false}\}$, returns $\mathbf{true}$ if user $u \in \mathrm{U}$ satisfies precondition $c \in \mathrm{C}$ in state $\gamma \in \Gamma$, else $\mathbf{false}$.
(4) $\mathrm{Satisfy}_{ug}$: $\mathrm{UG} \times \mathrm{C} \times \Gamma \rightarrow \{\mathbf{true}, \mathbf{false}\}$, returns $\mathbf{true}$ if user group $ug \in \mathrm{UG}$ satisfies precondition $c \in \mathrm{C}$ in state $\gamma \in \Gamma$, else $\mathbf{false}$.
(5) $\mathrm{Satisfy}_{u\text{–}ug}$: $\mathrm{U} \times \mathrm{C} \times \Gamma \rightarrow \{\mathbf{true}, \mathbf{false}\}$, returns $\mathbf{true}$ if user $u \in \mathrm{U}$ satisfies precondition $c \in \mathrm{C}$ in state $\gamma \in \Gamma$, else $\mathbf{false}$.

Request	Pre-Conditions	Target State
add(ar, u, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in \mathsf{canAddU}_{att}$. ($\mathrm{Satisfy}_u(u,\ c,\ \gamma_1) \wedge$ $val' \notin att_{\gamma_1}(u)$)	$att_{\gamma_2}(u) = att_{\gamma_1}(u) \cup \{val'\}$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\mathrm{directUg}_{\gamma_2}(u) = \mathrm{directUg}_{\gamma_1}(u)$, $\mathrm{UAA}_{\gamma_2} = \mathrm{UAA}_{\gamma_1} \setminus \langle u, att, att_{\gamma_1}(u) \rangle \cup \langle u, att, att_{\gamma_2}(u) \rangle$.
delete(ar, u, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in \mathsf{canDeleteU}_{att}$. ($\mathrm{Satisfy}_u(u,\ c,\ \gamma_1) \wedge$ $val' \in att_{\gamma_1}(u)$)	$att_{\gamma_2}(u) = att_{\gamma_1}(u) \setminus \{val'\}$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\mathrm{directUg}_{\gamma_2}(u) = \mathrm{directUg}_{\gamma_1}(u)$, $\mathrm{UAA}_{\gamma_2} = \mathrm{UAA}_{\gamma_1} \setminus \langle u, att, att_{\gamma_1}(u) \rangle \cup \langle u, att, att_{\gamma_2}(u) \rangle$.
add(ar, ug, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in \mathsf{canAddUG}_{att}$. ($\mathrm{Satisfy}_{ug}(ug,\ c,\ \gamma_1) \wedge$ $val' \notin att_{\gamma_1}(ug)$)	$att_{\gamma_2}(ug) = att_{\gamma_1}(ug) \cup \{val'\}$, $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $\mathrm{directUg}_{\gamma_2}(u) = \mathrm{directUg}_{\gamma_1}(u)$, $\mathrm{UGAA}_{\gamma_2} = \mathrm{UGAA}_{\gamma_1} \setminus \langle ug, att, att_{\gamma_1}(ug) \rangle \cup \langle ug, att, att_{\gamma_2}(ug) \rangle$.
delete(ar, ug, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in \mathsf{canDeleteUG}_{att}$. ($\mathrm{Satisfy}_{ug}(ug,\ c,\ \gamma_1) \wedge$ $val' \in att_{\gamma_1}(ug)$)	$att_{\gamma_2}(ug) = att_{\gamma_1}(ug) \setminus \{val'\}$, $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $\mathrm{directUg}_{\gamma_2}(u) = \mathrm{directUg}_{\gamma_1}(u)$, $\mathrm{UGAA}_{\gamma_2} = \mathrm{UGAA}_{\gamma_1} \setminus \langle ug, att, att_{\gamma_1}(ug) \rangle \cup \langle ug, att, att_{\gamma_2}(ug) \rangle$.
assign(ar, u, ug')	$\exists \langle ar,\ c,\ ug' \rangle \in \mathsf{canAssign}$. ($\mathrm{Satisfy}_{u\text{–}ug}(u,\ c,\ \gamma_1) \wedge$ $ug' \notin \mathrm{directUg}_{\gamma_1}(u)$)	$\mathrm{directUg}_{\gamma_2}(u) = \mathrm{directUg}_{\gamma_1}(u) \cup \{ug'\}$ $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\mathrm{UGA}_{\gamma_2} = \mathrm{UGA}_{\gamma_1} \setminus \langle u, \mathrm{directUg}_{\gamma_1}(u) \rangle \cup \langle u, \mathrm{directUg}_{\gamma_2}(u) \rangle$.
remove(ar, u, ug')	$\exists \langle ar,\ c,\ ug' \rangle \in \mathsf{canRemove}$. ($\mathrm{Satisfy}_{u\text{–}ug}(u,\ c,\ \gamma_1) \wedge$ $ug' \in \mathrm{directUg}_{\gamma_1}(u)$)	$\mathrm{directUg}_{\gamma_2}(u) = \mathrm{directUg}_{\gamma_1}(u) \setminus \{ug'\}$ $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\mathrm{UGA}_{\gamma_2} = \mathrm{UGA}_{\gamma_1} \setminus \langle u, \mathrm{directUg}_{\gamma_1}(u) \rangle \cup \langle u, \mathrm{directUg}_{\gamma_2}(u) \rangle$.

transition between states occurs when an authorized administrative request changes either direct user or group attribute, or changes user to group membership. The general definition for $\mathrm{GURA_G}$ scheme is as follows.

Definition 3 ($\mathrm{GURA_G}$ Scheme). *A $\mathrm{GURA_G}$ scheme is a state transition system $\langle$U, UA, AR, SCOPE, UG, $\succeq_{ug}$, Ψ, Γ, $\delta\rangle$ where,*

- *(i)* U, UA, AR, UG, $\succeq_{ug}$ *are as defined in Tables 1 and 3.*
- *(ii)* $\mathrm{SCOPE} = \langle \mathrm{SCOPE}_{att_1} \ldots \mathrm{SCOPE}_{att_n} \rangle$ *where* $att_i \in$ UA, *is the collection of scopes of all attributes.*
- *(iii)* Ψ *is the collection of all administrative rules in UAA, UGAA and UGA sub-models.*
- *(iv)* Γ *and* δ *are set of states and transition function respectively, defined in following parts of this subsection.*

3.2.4 Direct State

Γ is the finite set of states where each state $\gamma \in \Gamma$ records directly assigned attributes of each user and user group, along with user to groups membership. The direct user attribute assignment in state γ, denoted by UAA_γ, contains tuples of the form $\langle u, att, val \rangle$ for every $u \in \mathrm{U}$ and every $att \in \mathrm{UA}$ such that $att(u) = val$ and $val \in \mathrm{Range}(att)$ in state γ. To ensure uniqueness of user attribute values we require the following:

$$\langle u, att, val_1 \rangle \in \mathrm{UAA}_\gamma \wedge \langle u, att, val_2 \rangle \in \mathrm{UAA}_\gamma \Rightarrow val_1 = val_2.$$

Similarly, direct user group attribute assignment in state γ, denoted by UGAA_γ, contains tuples of the form $\langle ug, att, val \rangle$ for every $ug \in \mathrm{UG}$ and every $att \in \mathrm{UA}$ such that $att(ug) = val$ and $val \in \mathrm{Range}(att)$ in state γ, with the following uniqueness requirement:

$$\langle ug, att, val_1 \rangle \in \mathrm{UGAA}_\gamma \wedge \langle ug, att, val_2 \rangle \in \mathrm{UGAA}_\gamma \Rightarrow val_1 = val_2.$$

Finally, direct user to group assignment in state γ, denoted UGA_γ, contains tuples of the form $\langle u, val \rangle$ for every $u \in \mathrm{U}$ such that $\mathrm{directUg}(u) = val$ and $val \in 2^{\mathrm{UG}}$ in state γ, with the following uniqueness requirement

$$\langle u, val_1 \rangle \in \mathrm{UGA}_\gamma \wedge \langle ug, val_2 \rangle \in \mathrm{UGA}_\gamma \Rightarrow val_1 = val_2.$$

Note that information in a state can be used to calculate the effective attributes for user or group and effective user to groups membership in that state. For convenience we understand the notation $att_\gamma(u)$, $att_\gamma(ug)$ and $\mathrm{directUg}_\gamma(u)$ to denote the values of these functions in state γ for $u \in \mathrm{U}$ and $ug \in \mathrm{UG}$.

3.2.5 Transition Function

Any change in the direct state records ($\mathrm{UAA}_\gamma, \mathrm{UGAA}_\gamma, \mathrm{UGA}_\gamma$) will transform the current state to a new state. The transition function specifies the change from one state to another in a $\mathrm{GURA_G}$ system based on current direct or effective values and administrative requests, as shown in Table 5. Formally, $\delta : \Gamma \times \mathrm{REQ} \rightarrow \Gamma$, where REQ is the set of possible administrative requests.

4 Restricted GURA$_\mathrm{G}$ (rGURA$_\mathrm{G}$)

In this section, we introduce a restricted form of $\mathrm{GURA_G}$ administrative model, called $\mathrm{rGURA_G}$, used in our attribute reachability analysis. This restricted form allows a subset of the precondition language defined for $\mathrm{GURA_G}$ [23], whereby our analysis also establishes lower bound results on the

TABLE 6
Example Rules in $\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_{1+}}}$ Schemes

Relation	Admin Role	Pre-requisite Condition	Value
		Rules in $\mathbf{rGURA_{G_0}}$ scheme	
canAddU_{skills}	DeptAdmin	c $\in e\_skills(u)$ $\wedge \neg$ (java $\in skills(u)$)	c++
canDelete$\mathsf{U}_{roomAcc}$	BuildAdmin	3.02 $\in e\_roomAcc(u)$	1.2
canAdd$\mathsf{UG}_{college}$	UnivAdmin	$\neg$ (COE $\in college(ug)$)	COS
canDelete$\mathsf{UG}_{roomAcc}$	BuildAdmin	2.04 $\in e\_roomAcc(ug)$	2.03
		Rules in $\mathbf{rGURA_{G_1}}$ scheme further add	
canAdd$\mathsf{U}_{studType}$	DeptAdmin	java $\in e\_skills(u)$ $\wedge$ 2.03 $\in roomAcc(u)$	Grad
canDelete$\mathsf{U}_{roomAcc}$	BuildAdmin	3.02 $\in roomAcc(u)$ $\wedge$ COS $\in college(u)$	3.02
canAddUG_{skills}	DeptAdmin	COS $\in college(ug) \wedge$ UnderGrad $\in e\_studType(ug)$	java
canDelete$\mathsf{UG}_{college}$	UnivAdmin	2.04 $\in e\_roomAcc(ug) \wedge$ 2.03 $\in e\_roomAcc(ug)$	BUS
		Rules in $\mathbf{rGURA_{G_{1+}}}$ scheme further add	
canAssign	DeptAdmin	1.02 $\in e\_roomAcc(u) \wedge \neg$ (BUS $\in college(u)) \wedge \mathrm{G_2} \in \mathrm{directUg}(u)$	$\mathrm{G_1}$
canRemove	GroupAdmin	$\mathrm{G_1} \in \mathrm{effUg}(u) \wedge \mathrm{G_2} \in \mathrm{directUg}(u)$	$\mathrm{G_2}$

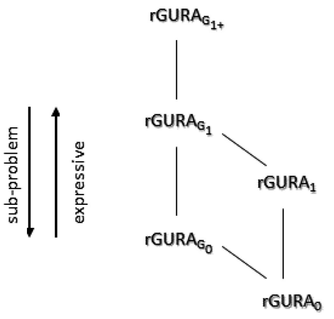


Fig. 3. $\mathrm{rGURA_G}$ (left side) and rGURA (right side) schemes.

complexity analysis for richer $\mathrm{GURA_G}$ model. We first present a generalized policy language for $\mathrm{rGURA_G}$, followed by three specific instances—$\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$, and $\mathrm{rGURA_{G_{1+}}}$.

The left side of Fig. 3 shows the relation between these schemes, while the right side shows the rGURA schemes discussed in [26]. At a high level, $\mathrm{rGURA_{G_0}}$ and $\mathrm{rGURA_{G_1}}$ add group attributes respectively to $\mathrm{rGURA_0}$and $\mathrm{rGURA_1}$, while $\mathrm{rGURA_{G_{1+}}}$ further adds administration of user membership in groups. Thereby, in $\mathrm{rGURA_{G_0}}$ and $\mathrm{rGURA_{G_1}}$ the administrative relations canAssign and canRemove are empty whereas they are populated in $\mathrm{rGURA_{G_{1+}}}$. Table 6 provides example administrative rules for each $\mathrm{rGURA_G}$ instance, as will be explained below.

Definition 4 ($\mathrm{rGURA_G}$ Scheme). *The $\mathrm{rGURA_G}$ scheme uses the policy grammar below, to specify preconditions C in Table 4*

$$\varphi ::= \neg\, \varphi \mid \varphi \wedge \varphi \mid svalue \in \text{direct} \mid svalue \in \text{effective}$$
$$svalue ::= sval_1 \mid sval_2 \mid \ \dots \ \mid sval_m,$$

where $\mathrm{SCOPE}_{att} = \{sval_1,\ sval_2,\ \dots,\ sval_m\}$. The two non-terminals direct *and* effective*, are individually defined in its three instances—$\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_{1+}}}$—in following subsections.*

4.1 The $\mathrm{rGURA_{G_0}}$ Scheme

In $\mathrm{rGURA_{G_0}}$ scheme, preconditions for rules in canAddU_{att} and canDeleteU_{att} relations only allow the same attribute att whose value is added or deleted from user. Therefore, conditions for user u have non-terminals direct and effective defined as follows:

$$\text{direct} ::= att(u) \quad \& \quad \text{effective} ::= e\_att(u).$$

Similarly, the administrative relations in canAddUG_{att} and canDeleteUG_{att} for user group ug have direct and effective defined as follows:

$$\text{direct} ::= att(ug) \quad \& \quad \text{effective} ::= e\_att(ug).$$

The examples for $\mathrm{rGURA_{G_0}}$ shown in Table 6 conform to these restrictions. Note that the attribute being updated is given as the subscript in the Relation column and the conditions in the Pre-requisite Condition column only involve this attribute.

4.2 The $\mathrm{rGURA_{G_1}}$ Scheme

In $\mathrm{rGURA_{G_1}}$ scheme, the precondition can include any attribute from the set of attributes. Therefore, conditions in rules for canAddU_{att} and canDeleteU_{att} relations for user u have direct and effective defined as follows where $att_i \in \mathrm{UA}$

$$\text{direct} ::= att_i(u) \quad \& \quad \text{effective} ::= e\_att_i(u).$$

Similarly, the conditions for user group ug in relations canAddUG_{att} and canDeleteUG_{att} have non-terminals direct and effective defined as follows:

$$\text{direct} ::= att_i(ug) \quad \& \quad \text{effective} ::= e\_att_i(ug).$$

The added rules for $\mathrm{rGURA_1}$in Table 6 illustrate this, where the preconditions involve attributes other than the one being updated. The earlier rules for $\mathrm{rGURA_{G_0}}$ continue to be valid for $\mathrm{rGURA_1}$.

4.3 The $\mathrm{rGURA_{G_{1+}}}$ Scheme

The $\mathrm{rGURA_{G_{1+}}}$ scheme allows changes in user group memberships besides modifying the attributes of user and user groups. Therefore, in addition to the grammar supported by $\mathrm{rGURA_{G_1}}$ scheme, $\mathrm{rGURA_{G_{1+}}}$ also includes user's direct or effective group memberships as preconditions in rules for canAssign and canRemove administrative relations. The additional grammar to specify such preconditions is specified below:

$$\varphi ::= ug \in \mathrm{directUg}(u) \mid ug \in \mathrm{effUg}(u).$$

In Table 6, rule in canAssign includes effective values for $roomAcc$, direct values for $college$ attribute and direct groups of user u.

5 Reachability Problem Definition

In this section, we provide a formal definition of our attribute reachability problem along with the reachability query and different query types supported in our analysis. The general approach is similar to that of [26], except that atomic-valued attributes are excluded (as noted in Section 3.1.2) and reachability is defined with respect to effective rather than direct attributes (Research in [26] does not have the notion of effective attributes).

The user attribute reachability analysis problem (or reachability problem) is based on the effective attributes of the user. Informally, the problem can be stated as: Given an initial transition system state with a set of attribute assignments of the user, the user's group memberships and the attributes of all the user's member groups, can administrators with a given set of administrative roles issue one or more administrative requests, which transition to a target state having the set of specified effective attributes for that user? We highlight some simplifications in our reachability analysis process. First, as the changes made to the attributes or group memberships of one user do not affect the attributes or group memberships of another user, our analysis will only determine the effective attributes of a single user of interest and hence will only consider attribute assignment of that user, its group memberships and attributes of these groups. Formally, we assume U = {u} in our analysis [26]. Second, as the reachability analysis focuses on powers of a certain set of administrative roles SUBAR $\subseteq$ AR, we do not consider the administrative rules specified for roles outside of SUBAR. In other words, we can assume AR = SUBAR. These simplifications gives our analysis more convenient statements without losing generality.

Definition 5 (Reachability Query). *A reachability query $q \in$ Q specifies a subset of effective values of a user for some attributes in any target state. Formally*

$$q \subseteq \{\langle u, e\_att, vset\rangle | u \in \mathrm{U}, att \in \mathrm{UA}, vset \in \mathrm{Range}(att)\}.$$

In the example problems discussed in Section 8, we will use the following notation to specify our query, which is equivalent to the notation defined above

$$q \subseteq \{e\_att(u) = vset | u \in \mathrm{U}, att \in \mathrm{UA}, vset \in \mathrm{Range}(att)\}.$$

For example,

$q = \{\langle \mathrm{u, e\_roomAcc}, \ \{2.04\}\rangle, \langle \mathrm{u, e\_skills}, \ \{c\}\rangle,$
$\langle \mathrm{u, e\_college}, \ \{COS, COE\}\rangle\}$ is equivalent to
$q = \{\mathrm{e\_roomAcc(u)} = \{2.04\}, \mathrm{e\_skills(u)} = \{c\},$
$\mathrm{e\_college(u)} = \{COS, COE\}\}.$

Two types of reachability query are defined in the system. A query is called "strict" satisfied if every effective attribute value specified in the query is exactly the same as that in the target state. A query is called "relaxed" satisfied by the user if in the target state every effective attribute value of the user is a superset of the corresponding attribute values specified in the reachability query. For example, let UA = {*skills*}, U = {*Bob*} *and reachability query q* = ⟨*Bob, e_skills,* {*c, java*}⟩. *For strict query type, q can be satisfied in states $\gamma' \in \Gamma$ where $e\_skills_{\gamma'}(u)$ = {c, java}. In relaxed query type, q can be satisfied by any state $\gamma'' \in \Gamma$ where $e\_skills_{\gamma''}(u)$ = $setval$ and {c, java} $\subseteq$ $setval$. For ease of understanding, we represent the effective value of attribute att for user u in state $\gamma \in \Gamma$ as $e\_att_{\gamma}(u)$. The formal definition for reachability query types is given below:*

Definition 6 (Reachability Query Types). *For any* $\mathrm{rGURA_G}$ *scheme* ⟨U, UA, AR, SCOPE, UG, $\succeq_{ug}$, Ψ, Γ, δ⟩, *we formally define two Reachability Query Types as:*

- $\mathrm{RP}_=$ *or strict satisfied queries have the entailment function* $\vdash_{\mathrm{RP}_=} : \Gamma \times \mathrm{Q} \rightarrow$ {**true**, **false**} *which returns* **true** *(i.e.,* $\gamma \vdash_{\mathrm{RP}_=} q$*) if* $\forall \langle u, e\_att, vset\rangle \in q.$ $e\_att_{\gamma}(u) = vset.$
- $\mathrm{RP}_\supseteq$ *or relaxed satisfied queries have the entailment function* $\vdash_{\mathrm{RP}_\supseteq} : \Gamma \times \mathrm{Q} \rightarrow$ {**true**, **false**} *which returns* **true** *(i.e.,* $\gamma \vdash_{\mathrm{RP}_\supseteq} q$*) if* $\forall \langle u, e\_att, vset\rangle \in q.$ $e\_att_{\gamma}(u) \supseteq vset.$

It is clear that given a scheme and problem instance, if $\mathrm{RP}_=$ query problem is satisfied then $\mathrm{RP}_\supseteq$ problem is also satisfied, but not vice versa. The following two definitions are same as defined in [26], but we will state them for the sake of completeness.

Definition 7 (Reachability Plan). *A Reachability Plan or plan is a sequence of authorized administrative requests to transition from initial state to the target state. For any* $\mathrm{rGURA_G}$ *scheme* ⟨U, UA, AR, SCOPE, UG, $\succeq_{ug}$, Ψ, Γ, δ⟩ *and states* $\gamma_0, \gamma' \in \Gamma$, *reachability plan is a sequence of authorized requests* $\langle req_1, req_2, \ldots, req_n\rangle$ *where* $req_i \in \mathrm{REQ}$ $(1 \leq i \leq n)$, *to transition from an initial state* γ_0 *to target state* γ' *if:* $\gamma_0 \overset{req_1}{\rightarrow} \gamma_1 \overset{req_2}{\rightarrow} \gamma_2 \ldots \overset{req_n}{\rightarrow} \gamma'$. *The arrow denotes a successful transition from one state to another due to an administrative request* req_i *authorized by rules in* Ψ. *We write* $\gamma_0 \overset{plan_\Psi}{\rightsquigarrow} \gamma'$ *to abbreviate the complete plan.*

Informally, a reachability problem deals if there exists a reachability plan to transition from an initial state to some target state where the effective attribute values of the user satisfy a particular reachability query. Formally,

Definition 8 (Reachability Problems). *Given any* $\mathrm{rGURA_G}$ *scheme* ⟨U, UA, AR, SCOPE, UG, $\succeq_{ug}$, Ψ, Γ, δ⟩, *the attribute reachability problem is as follows:*

- $\mathrm{RP}_=$ *or strict reachability problem instance* I *is of the form* $\langle \gamma_0, q\rangle$ *where* $\gamma_0 \in \Gamma$, $q \in \mathrm{Q}$ *and checks if there exist a reachability plan P such that* $\gamma_0 \overset{P_\Psi}{\rightsquigarrow} \gamma'$ *and* $\gamma' \vdash_{\mathrm{RP}_=} q$.
- $\mathrm{RP}_\supseteq$ *or relaxed reachability problem instance* I *is of the form* $\langle \gamma_0, q\rangle$ *where* $\gamma_0 \in \Gamma$, $q \in \mathrm{Q}$ *and checks if there exist a reachability plan P such that* $\gamma_0 \overset{P_\Psi}{\rightsquigarrow} \gamma'$ *and* $\gamma' \vdash_{\mathrm{RP}_\supseteq} q$.

6 PSPACE-Complete Reachability

In this section, we present our reachability analysis results for different $\mathrm{rGURA_G}$ schemes shown in Fig. 3. These results are extensions to the results from GURA reachability analysis [26] and also considers groups for assigning attributes to its member users. Our analysis will prove that rGURA_G schemes in Fig. 3 in general are PSPACE-complete. For such schemes we will first show that all rGURA_G schemes are in PSPACE and then reduce a known PSPACE-complete problem to our problem schemes. In the next section, we will provide polynomial algorithms for some restricted rGURA_G problem classes.

Lemma 1. *Reachability problem for every* rGURA_G *scheme in Fig. 3 is in PSPACE.*

Proof. Each state of a non-deterministic Turing machine stores some information to predict future states. This information takes polynomial amount of space and therefore all instance are in PSPACE. This proof is similarly stated for GURA schemes in [26] and more details are presented in appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2022.3145358. □

Since all rGURA_G schemes are in PSPACE, it will now be sufficient to prove that all rGURA_G schemes are PSPACE-hard, which will conclude that the schemes are PSPACE-complete.

Corollary 1. *Reachability query types* $\text{RP}_\supseteq$ *and* $\text{RP}_=$ *for* rGURA_G *schemes in general is PSPACE-complete.*

Proof. Recall that Fig. 3 defines the relation between different rGURA_G schemes and rGURA_0. The reachability analysis for rGURA_0 scheme discussed in [26] describes the scheme is PSPACE-complete. This scheme only allows change in attributes of the user. With respect to $\text{rGURA}_{\text{G}_0}$, it can be said that rGURA_0 scheme is a sub-problem without user groups. Therefore, the reduction from known PSPACE-complete problem (rGURA_0) to $\text{rGURA}_{\text{G}_0}$ is straightforward, which makes $\text{rGURA}_{\text{G}_0}$ as PSPACE-hard. Further, using Lemma 1, it is justified to claim that $\text{rGURA}_{\text{G}_0}$ is in PSPACE-complete.

Similar claim can also be made for $\text{rGURA}_{\text{G}_1}$ scheme where $\text{rGURA}_{\text{G}_0}$ is its sub-problem involving only the same attribute in preconditions for rules (Ψ). Therefore, $\text{rGURA}_{\text{G}_1}$ is PSPACE-hard and using Lemma 1, it is also PSPACE-complete. The analysis for $\text{rGURA}_{\text{G}_{1+}}$ is also alike the above two schemes where $\text{rGURA}_{\text{G}_1}$ is a sub-problem of $\text{rGURA}_{\text{G}_{1+}}$, therefore, $\text{rGURA}_{\text{G}_{1+}}$ is in PSPACE-hard and hence PSPACE-complete also. More details are present in appendix, available in the online supplemental material. □

7 Polynomial Reachability for Restricted Cases

In previous section, we proved that attribute reachability for any rGURA_G scheme in general is PSPACE-complete. However, we have identified some instances of rGURA_G schemes which can be solved in polynomial time under precondition restrictions on administrative rules (Ψ). The practicality and use of these restricted instances are discussed in the example problem use-case discussed in Section 8. Similar to [26], the following restrictions are considered where $\overline{\text{D}}$ and SR_d are always imposed together:

- *No negation* ($\overline{\text{N}}$): Ψ satisfies $\overline{\text{N}}$ if no administrative rules in Ψ use negation in preconditions.
- *No deletion* ($\overline{\text{D}}$): Ψ satisfies $\overline{\text{D}}$ if for each attribute $att \in \text{UA}$, canDeleteU_{att} and canDeleteUG_{att} are empty. Further, canRemove rules are also empty, meaning, attribute values or groups once added cannot be deleted.
- *Single rule with direct values* (SR_d): Ψ satisfies SR_d if for each attribute $att \in \text{UA}$, there is at most one precondition associated with a particular value assignment in rules of $\text{canAddU}_\text{att}$ or $\text{canAddUG}_\text{att}$. Therefore, an attribute value pair can either be added through user directly or through groups but not both. Similar, condition also exists for canAssign rules. Further, only direct conjuncts i.e., $val \in att_i(u)$, $val \in att_i(ug)$ or $ug \in \text{directUg}(u)$ are allowed in prerequisite condition.

These restrictions are important in different kinds of attributes and scenarios. For instance, *No negation* ($\overline{\text{N}}$) restrictions have significance when attributes like $course$ or $degree$ are added to entities. It is likely that adding a new value for $course$ attribute do not require negation of another course as the precondition. Similarly, *No deletion* ($\overline{\text{D}}$) restriction can apply for attributes like $skills$ where a value once added to any entity will never be deleted. Ideally, an individual attaining some skill-set will never loose them. The SR_d restriction allows only unique preconditions in administrative relations for user and user groups. This restriction essentially separates set of attributes into two parts, one which can be assigned only to user directly and others assigned through groups. For example, attribute like $roomAccess$ can be assigned through group as it is usually common to all users with certain characteristics, and if value changes for one user, it will change for all others too. Attribute like $advisor$ is assigned individually to each user as change for one user may not change it in others. Therefore, these restrictions are relevant in applications.

We now discuss reachability analysis for restricted rGURA_G schemes. The notation [$\text{rGURA}_{\text{G}x}$, `Restriction`] specifies special instances of rGURA_G scheme where subscript x takes a value in 0, 1 or 1+ representing – $\text{rGURA}_{\text{G}_0}$, $\text{rGURA}_{\text{G}_1}$ or $\text{rGURA}_{\text{G}_{1+}}$ and `Restriction` represents combinations of $\overline{\text{N}}$, $\overline{\text{D}}$ and SR_d specifying that administrative rules Ψ in the scheme satisfy these restrictions. For example, [$\text{rGURA}_{\text{G}_0}$– $\overline{\text{N}}$] denotes $\text{rGURA}_{\text{G}_0}$ scheme where rules in Ψ satisfy $\overline{\text{N}}$.

As shown in Fig. 3, $\text{rGURA}_{\text{G}_{1+}}$ scheme is the most expressive scheme where new attribute values are achieved by direct assignment to the user or to its effective groups, and also by changing user to group memberships. It is clear from the previous discussions that the scheme covers $\text{rGURA}_{\text{G}_1}$ and $\text{rGURA}_{\text{G}_0}$, which only allow change in attributes of the user or its effective groups. Therefore, we will only discuss algorithm for restricted $\text{rGURA}_{\text{G}_{1+}}$ scheme which can be easily used for other two schemes by simply ignoring irrelevant administrative rules.

7.1 Reachability Plan for $\text{RP}_=$ in [$\text{rGURA}_{\text{G}_{1+}}$ – $\overline{\text{N}}$]

First we will discuss reachability query type $\text{RP}_=$ for [$\text{rGURA}_{\text{G}_{1+}}$ –$\overline{\text{N}}$] scheme which can be solved in polynomial time by Algorithm 1. This algorithm extends the algorithm discussed for rGURA_1 [26] by including user group attribute assignments and also modification in user to group

Algorithm 1. Plan Generation for $\mathrm{RP}_=$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}\,]$

```
1:  Input: problem instance I = $\langle \gamma_0, q \rangle$ Output: $plan$ or false
2:  $plan := \langle\rangle$;                                                      ▷Initialize plan
3:  $s := \gamma_0$;                                                               ▷Initialize with state s
4:  if ($\exists\, att \in \mathrm{UA}\ \exists\, \langle u, e\_att, vset\rangle \in q).\ e\_att(u) - vset \neq \emptyset$ then
      return false;                                                  ▷Check if state s has more values than query
    ▷ Assign attribute values required in query to the user or its effective groups
5:  while ($s \nvdash_{\mathrm{RP}_=} q\ \wedge$
6:    (($\exists\, att' := att \in \mathrm{UA}\ \exists\, rule := \langle ar, c, val\rangle \in \mathsf{canAddU}_{att'}$).
      ($\mathrm{Satisfy}_u(u,\ c,\ s) \wedge val\ \notin\ att'(u) \wedge \exists\, \langle u, e\_att', vset\rangle \in q.\ val \in vset$ ))
7:    $\vee$
8:    (($\exists\, att' := att \in \mathrm{UA}\ \exists\, rule := \langle ar, c, val\rangle \in \mathsf{canAddUG}_{att'}$). ($\exists\, ug' := ug \in \mathrm{effUg}(u)$. $\mathrm{Satisfy}_{ug}(ug',\ c,\ s) \wedge val\ \notin\ att'(ug') \wedge$
9:                              $\exists\, \langle u, e\_att', vset\rangle \in q.\ val \in vset$))
10:   $\vee$
11:   (($\exists\, ug'' := ug \in \mathrm{UG}\ \exists\, rule := \langle ar, c, ug''\rangle \in \mathsf{canAssign}$).
      ($\mathrm{Satisfy}_{u-ug}(u,\ c,\ s) \wedge ug''\ \notin\ \mathrm{directUg}(u)\ \wedge$
12:       $\forall\, att \in \mathrm{UA}\ \ \exists \langle u, e\_att, vset\rangle \in q.\ e\_att(ug'') \subseteq vset$))) do
13:     $s := s \ll rule$;                                                   ▷apply rule on state s
14:   switch                                                 ▷append administrative request to plan
15:     case $rule \in \mathsf{canAddU}_{att'}$:
16:       $plan := plan$.append($\mathsf{add}(ar, u, att', val)$);
17:     break;
18:     case $rule \in \mathsf{canAddUG}_{att'}$:
19:       $plan := plan$.append($\mathsf{add}(ar, ug', att', val)$);
20:     break;
21:     case $rule \in \mathsf{canAssign}$:
22:       $plan := plan$.append($\mathsf{assign}(ar, u, ug'')$);
23:     break;
24: end while
25: if $s \vdash_{\mathrm{RP}_=} q$ then return $plan$ else return false end if      ▷check if reachability query is satisfied
```

memberships. The added restriction to this scheme ($\overline{\mathrm{N}}$) requires preconditions in rules without negation conjuncts and therefore, administrative rules cannot specify addition of new attributes based on the absence of some other values. Hence, the current attribute values of user or groups are not required to be removed for adding new values or group, which precludes the need for investigating any $\mathsf{canDeleteU}_{att}$, $\mathsf{canDeleteUG}_{att}$ and $\mathsf{canRemove}$ rules.

The algorithm starts with the current set of attribute values and group memberships for user, and the attribute values for its member groups. It traverse all relevant $\mathsf{canAddU}_{att}$, $\mathsf{canAddUG}_{att}$ or $\mathsf{canAssign}$ rules to add new values to the attributes of user or to its effective user groups and also add new groups to the user. Since, the query type is restricted, the algorithm first checks if the current effective attributes of user are not more than what required in the query (line 4). If the current values are extra, the algorithm returns false, since there are no delete administrative relations to delete such values. The while loop (line 5–24) terminates when either the query is satisfied or when no other values can be added from the rules in $\mathsf{canAddU}_{att}$ and $\mathsf{canAddUG}_{att}$ or no new groups can be added to the user using $\mathsf{canAssign}$ rules. When adding a new value to the user or its effective groups, the corresponding value must be checked against the query. If the value is present in the query, the addition is allowed. Similar check is also done to add new groups the user, where all the attributes present in the group should also be a part of the query. The order to add these values or new groups is independent to each other, since no negation conjuncts are required and presence of extra values in user or group will not stop from adding new values. Also, if later a new value is added to an entity, the while loop will again consider the relevant rules to add values based on the already added values. When a new attribute is added to user, its effective groups or a new group is assigned to the user, its corresponding administrative request is appended to the reachability plan $plan$. If the query is satisfied, the algorithm returns the corresponding reachability plan $plan$ or returns false stating that the query is unsatisfiable and user will not achieve desired effective attributes as mentioned in query.

Theorem 1. *Reachability query type* $\mathrm{RP}_=$ *for scheme* $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}]$ *is P.*

Proof. Algorithm 1 describes the polynomial time algorithm.

Complexity. The complexity is determined by the number of times the administrative rules in $\mathsf{canAddU}_{att}$, $\mathsf{canAddUG}_{att}$ or $\mathsf{canAssign}$ are traversed. If only one value is added by each of the rules, the complexity of Algorithm 1 is $\mathcal{O}(|\mathsf{canAssign}| \times |\mathrm{UG}| + ((\sum_{att \in \mathrm{UA}} |\mathrm{SCOPE}_{att}|) \times (|\mathsf{canAddU}_{att}| + |\mathsf{canAddUG}_{att}| \times |\mathrm{UG}|)))$, where $|\mathsf{canAddU}_{att}|$, $|\mathsf{canAddUG}_{att}|$ and $|\mathsf{canAssign}|$ represents number of the administrative rules in these relations and $|\mathrm{UG}|$ represents the maximum number of groups assigned to the user. Clearly, the complexity of algorithm is polynomial. □

The $\mathrm{RP}_\supseteq$ query type for $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}]$ also has a polynomial algorithm, where the extra conditions to check the

Algorithm 2. Group Assignment Plan Generation for $\mathrm{RP}_=$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_\mathrm{d}]$

```
1: Input: problem instance I = ⟨$\gamma_0$, $q$⟩ Output: $plan_{ug}$
2: if $\gamma_0 \vdash_{\mathrm{RP}_=} q$ then return $plan_{ug} := \langle\rangle$;                ▷Check initial
   state
3: $G_{ug} := \langle V_{ug}, E_{ug}\rangle$; $V_{ug} := \{ug \mid \exists\, ug \in \mathrm{UG}.\ \exists\, ug \notin \mathrm{directUg}(u).\ \exists\, \langle ar, c, ug\rangle \in$ canAssign$(u)$. $\forall\, att \in \mathrm{UA}\ \exists\langle u, e\_att, vset\rangle \in q.\ e\_att(ug)$
   $\subseteq vset\}$; $E_{ug} := \emptyset$;                ▷Construct a directed graph
4: for each pair of nodes $((ug_1, ug_2) \in V_{ug})$ do
5:     if $((\exists\langle ar, c, ug_2\rangle \in$ canAssign. "$(ug_1 \in \mathrm{directUg}(u))$"
       is a conjunct in $c) \vee$
6:     $(\exists\langle ar, c, ug_1\rangle \in$canAssign. "$\neg(ug_2 \in \mathrm{directUg}(u))$"
       is a conjunct in $c$))
7:     then $E_{ug} := E_{ug} \cup \{\langle ug_1, ug_2\rangle\}$; end if                ▷Add edges
8: end for
9: if graph $G_{ug}$ has cycles then remove the cyclic paths and
   $plan_{ug}$ := sequence of assign requests corresponding to the
   topological sort of $G_{ug}$;
```

query before adding new values is removed since we can have values even if they are not required in the query. The complexity will remain the same as shown in Theorem 1. Similar algorithm can also be devised for $\mathrm{RP}_=$ and $\mathrm{RP}_\supseteq$ query type in $[\mathrm{rGURA}_{\mathrm{G}_1} - \overline{\mathrm{N}}]$ and $[\mathrm{rGURA}_{\mathrm{G}_0} - \overline{\mathrm{N}}]$ schemes where canAssign rules will not be considered into the while loop for adding new groups to the user. Hence these schemes can be also solved in polynomial time.

7.2 Reachability Plan for $\mathrm{RP}_=$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_\mathrm{d}]$

We will now consider another restricted instance for $\mathrm{rGURA}_{\mathrm{G}_{1+}}$, $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_\mathrm{d}]$ which can be solved by Algorithms 2 and 3. The scheme has two restrictions, $\overline{\mathrm{D}}$ which removes the need to consider delete administrative relations – canDeleteU$_{att}$, canDeleteUG$_{att}$ and canRemove. The SR_d restriction allows single preconditions for each attribute value pair or user group, with only direct values as conjuncts in preconditions. This restriction results in rules which can be either satisfied by user or any of its effective groups but not both. We have divided the algorithm into two algorithm for ease of understanding and to show how these algorithms can be reused in other schemes also.

Algorithm 2 is used to add new groups to the user. Since the preconditions only involves user's direct groups as conjuncts (SR_d restriction), the addition of groups is independent of the attributes and can be calculated separately. The administrative rules in this scheme can have negation conjuncts in preconditions, therefore, the order of assigning new groups can be mutually dependent. The algorithm first creates a directed graph where vertices V_{ug} are user groups and edges E_{ug} are directed based on conjuncts in precondition of rules in canAssign. In line 3, before adding a group to V_{ug}, it is checked that all the attributes in group are required in query, as no extra attributes are allowed in $\mathrm{RP}_=$ and deletion is not allowed. Line 4 – 8 creates edges in the graph, if a user group ug_1 is a negation conjunct to add another group ug_2 or ug_2 is a precondition for ug_1, then edge is drawn from ug_2 to ug_1, signifying that ug_2 should be added before ug_1. If cycles exists in the created graph then remove the cyclic paths and create topological sort on the remaining graph. The set of administrative requests based on the sort will provide the $plan_{ug}$ for user to groups assignment. Once the requests are executed in order, new effective groups are calculated for the user and computation continues from Algorithm 3.

Algorithm 3 extends algorithm defined in [26], which checks the final set of values required to satisfy the reachability query and find canAddU$_{att}$ or canAddUG$_{att}$ rules to add those values. Further to add the values in precondition of rules, it may in-turn need some other rules and values and so on. Therefore, algorithm traverses backward to find the set of values required to satisfy the query. Since the values can be achieved by user directly or from any of its effective groups, this backward search is done for user and all its effective groups as calculated by Algorithm 2.

The algorithm starts by checking if the query is satisfied in the current state, in that case empty plan is returned signifying that with only new group assignments query is satisfied. Otherwise, it creates a set of attribute value pair for values required in query q and also for current attributes of user and its effective groups (line 4-5). Line 6 checks if the union of current values of the user or its effective groups is not more than values required in q. If extra values are there, the algorithm returns false, as no delete rules are allowed. The algorithm calculates all positive precondition attribute value pairs required by user or its effective groups to get values in *toadd* (line 7-17). Therefore, the final set of values required includes the values in query (*toadd*) and positive preconditions in user or its effective groups excluding their current values. Line 18 checks if rules exists to add all required attribute value pair or else returns false, as the values can not be added. Line 19-21 calculate negative conjuncts in rules required to add required values and returns false if the such values are present in current state. After passing through all checks, the algorithm starts creating a directed graph. Vertices (V) in the graph are attribute value pair of the values required in the query q and the required positive preconditions excluding the values in the current state. Edges E will be drawn in the direction defined in the for loop (line 23-30). If the attribute value pair (att_1, val_1) is in the negative conjunct in administrative rule for (att_2, val_2) or (att_2, val_2) is a positive conjunct in a rule to add (att_1, val_1), the edge is created from (att_2, val_2) to (att_1, val_1). Since our query type is $\mathrm{RP}_=$, it requires an additional check so that no extra values are added to the user.

Algorithm 3. Plan Generation for $\mathrm{RP}_=$ in $[\mathrm{rGURA}_{\mathrm{G}_1} - \overline{\mathrm{D}}, \mathrm{SR}_\mathrm{d}]$

```
1: Input: problem instance I = $\langle \gamma_0, q \rangle$ Output: $plan$ or false
2: if $\gamma_0 \vdash_{\mathrm{RP}_=} q$ then return $plan := \langle \rangle$;    ▷Check initial state
3: $toadd := \{(att, val) \mid att \in \mathrm{UA}, \langle u, e\_att, vset \rangle \in q, val \in vset\}$    ▷Values required in query
4: $cur_u := \{(att, val) \mid att \in \mathrm{UA}, val \in att(u)\}$    ▷Current values of user
5: for each $ug \in \mathrm{effUg(u)}$ do $cur_{ug} := \{(att, val) \mid att \in \mathrm{UA}, val \in att(ug)\}$ end for    ▷Current values of user's effective groups
6: if $(cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug})) - toadd \neq \emptyset$ then return false;    ▷Check if state $\gamma_0$ has more values than query
7: $ppre_u := \emptyset$; for each $ug \in \mathrm{effUg}(u)$ do $ppre_{ug} := \emptyset$; end for
8: for (each $(att, val) \in toadd \cup ppre_u$) do    ▷Positive precondition values for user
9:     $ppre'_u := \{(att_1, val_1) \mid \exists \langle ar, c, val \rangle \in \mathsf{canAddU}_{att}$. "$val_1 \in att_1(u)$" is a conjunct in $c\}$;
10:    $ppre_u := (ppre_u \cup (ppre'_u \setminus ppre_u)) \setminus cur_u$;
11: end for
12: for (each $ug \in \mathrm{effUg}(u)$) do    ▷Positive precondition values for effective groups
13:     for (each $(att, val) \in toadd \cup ppre_{ug}$) do
14:         $ppre'_{ug} := \{(att_1, val_1) \mid \exists \langle ar, c, val \rangle \in \mathsf{canAddUG}_{att}$. "$val_1 \in att_1(ug)$" is a conjunct in $c\}$;
15:         $ppre_{ug} := (ppre_{ug} \cup (ppre'_{ug} \setminus ppre_{ug})) \setminus cur_{ug}$;
16:     end for
17: end for
    ▷ Check if rules exists for values required
18: if $((\exists (att, val) \in toadd \cup ppre_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} ppre_{ug}) \setminus (cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug}))). \nexists \langle ar, c, val \rangle \in \mathsf{canAddU}_{att} \cup \mathsf{canAddUG}_{att}$)then return false;
    ▷ Find negation values in rules required to add values for the user and its effective groups
19: $npre_u := \{(att_1, val_1) \mid \exists (att, val) \in (toadd \cup ppre_u) \setminus cur_u. \exists \langle ar, c, val \rangle \in \mathsf{canAddU}_{att}$. "$\neg(val_1 \in att_1(u))$" is a conjunct in $c\}$
20:  for each $ug \in \mathrm{effUg}(u)$ do $npre_{ug} := \{(att_1, val_1) \mid \exists (att, val) \in (toadd \cup ppre_{ug}) \setminus cur_{ug} . \exists \langle ar, c, val \rangle \in \mathsf{canAddUG}_{att}$. "$\neg(val_1 \in att_1(ug))$" is a conjunct in $c\}$ end for
21: if $((npre_u \cap cur_u \neq \emptyset) \vee (\forall ug \in \mathrm{effUg}(u). npre_{ug} \cap cur_{ug} \neq \emptyset))$ then return false;    ▷Negation in current values
22: $G := \langle V, E \rangle$; $V := toadd \cup ppre_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} ppre_{ug}) \setminus (cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug}))$; $E := \emptyset$;    ▷Construct a directed graph
23: for each pair of nodes $((att_1, val_1), (att_2, val_2)) \in V$ do
24:     if $(((\exists \langle ar, c, val_2 \rangle \in \mathsf{canAddU}_{att_2}$. "$(val_1 \in att_1(u))$" is a conjunct in $c) \vee$
25:     $(\exists \langle ar, c, val_1 \rangle \in \mathsf{canAddU}_{att_1}$. "$\neg(val_2 \in att_2(u))$" is a conjunct in $c))$
26:     $\vee$
27:     $((\exists\, ug \in \mathrm{effUg}(u)). ((\exists \langle ar, c, val_2 \rangle \in \mathsf{canAddUG}_{att_2}$. "$(val_1 \in att_1(ug))$" is a conjunct in $c) \vee$
28:     $(\exists \langle ar, c, val_1 \rangle \in \mathsf{canAddUG}_{att_1}$. "$\neg(val_2 \in att_2(ug))$" is a conjunct in $c))))$
29:     then $E := E \cup \{\langle (att_1, val_1), (att_2, val_2) \rangle\}$;
        end if    ▷Add edges to the graph
30: end for
31: $valset := toadd - (cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug}))$;    ▷Values in query not in state $\gamma$
32: if $\exists (att_1, val_1) \in valset\ \exists \langle (att, val), (att_1, val_1) \rangle \in E$. $(att, val) \notin valset$ then return false
33: else $V := vset\ E := E - \{\langle (att, val), (att_1, val_1) \rangle \mid (att, val) \notin vset, (att_1, val_1) \notin valset\}$ end if
34: if graph G has a cycle then return false else return $plan :=$ sequence of administrative requests corresponding to the topological sort of G;
```

Therefore, once the graph is created, we create a set $valset$, which includes values required in query and not present in the current state. If the created graph has vertex in $valset$ having incoming edge not from vertex in $valset$, algorithm returns false (line 32). Otherwise it removes all the edges from vertices not in $valset$. If cycles exists in the remaining graph then algorithm returns false, else the set of administrative request corresponding to the topological sort will return the $plan$.

TABLE 7
Example Problem Instance for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}\,]$

Input: problem instance I = $\langle \gamma_0, q\rangle$ **Output:** $plan$ or false
$\psi \in \Psi$:
canAddU$_{\mathrm{roomAcc}}$ = $\{\langle \mathrm{BuildAdmin},\ \mathrm{c{+}{+}} \in \mathrm{e\_skills}(u) \wedge 2.04 \in \mathrm{roomAcc}(u),\ 1.2\rangle\ \}$,
canAddU$_{\mathrm{college}}$ $= \{\langle \mathrm{BuildAdmin},\ \mathrm{python} \in \mathrm{e\_skills}(u) \wedge 3.05 \in \mathrm{roomAcc}(u),\ \mathrm{COE}\rangle\}$,
canAddU$_{\mathrm{skills}}$ $= \{\langle \mathrm{DeptAdmin},\ \mathrm{c} \in \mathrm{e\_skills}(u),\ \mathrm{python}\rangle\}$,
canAddUG$_{\mathrm{roomAcc}}$ $= \{\langle \mathrm{BuildAdmin},\ 3.02 \in \mathrm{roomAcc}(ug),\ 1.2\rangle\ \}$,
canAssign $= \{\langle \mathrm{DeptAdmin},\ \mathrm{G}_1 \in \mathrm{directUg}(u),\ \mathrm{G}_3\rangle\ \}$
Queries:
$q_1 \in$ Q = $\{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02, 1.2\},\ \ \mathrm{e\_skills(u)} = \{\mathrm{c, c{+}{+}, python}\},\ \ \mathrm{e\_college(u)} = \{\mathrm{COS}\}\}$
$q_2 \in$ Q = $\{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02, 1.2\},\ \ \mathrm{e\_skills(u)} = \{\mathrm{c, c{+}{+}}\},\ \ \mathrm{e\_college(u)} = \{\mathrm{COS, COE}\}\}$

U = {u},
UA = {college, roomAcc, skills},
AR = {DeptAdmin, BuildAdmin},
UG = {G_1, G_2, G_3, G_5},
UGH = {(G_1, G_1), (G_2, G_2), (G_1, G_2), (G_3, G_3), (G_5, G_5)}
SCOPE$_{college}$ = {COS, COE, BUS},
SCOPE$_{roomAcc}$ = {1.2, 2.03, 2.04, 3.02, 3.05},
SCOPE$_{skills}$ = {c, c++, python, matlab}
δ (Transition Function)

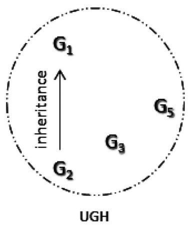


Fig. 4. Input starting state ($\gamma_0 \in \Gamma$).

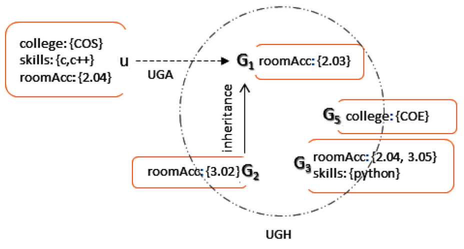


Fig. 5. Initial state for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}\,]$.

Therefore, the overall reachability plan returned will be $plan_{ug}$ from Algorithm 2 and $plan$ from Algorithm 3.

Theorem 2. *Reachability query type* $\mathrm{RP}_{=}$ *for scheme* $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ *is P.*

Proof. Algorithms 2 and 3 describe the polynomial algorithms.

Complexity. The algorithm takes polynomial time to create directed graphs and then to compute topological sort. Its complexity is $\mathcal{O}(|\mathrm{UG}| \times |\mathsf{canAssign}| + ((\sum_{att \in \mathrm{UA}} |\mathrm{SCOPE}_{att}|) \times (|\mathsf{canAddU}_{att}| + |\mathsf{canAddUG}_{att}| \times |\mathrm{UG}|)))$. □

In case of $\mathrm{RP}_{\supseteq}$ query type for $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$, we remove the extra checks to verify if no extra values are present in current state (line 6). Further line 31-33 is not required as extra values are allowed to be added to user. With these minor changes, the complexity of $\mathrm{RP}_{\supseteq}$ for scheme $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ is P.

It should be noted that $\mathrm{RP}_{=}$ for $[\mathrm{rGURA}_{\mathrm{G}_1} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ do not allow changes in group memberships of user. Therefore, computation for this scheme will start directly from Algorithm 3, obviating the execution of Algorithm 2. The $\mathrm{RP}_{\supseteq}$ query for $[\mathrm{rGURA}_{\mathrm{G}_1} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ will remove all extra conditions applied in Algorithm 3 for $\mathrm{RP}_{\supseteq}$ scheme for $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ as discussed above. Also, since $\mathrm{rGURA}_{\mathrm{G}_0}$ is a sub-problem of $\mathrm{rGURA}_{\mathrm{G}_1}$ we can conjecture that $\mathrm{RP}_{=}$ and $\mathrm{RP}_{\supseteq}$ for scheme $[\mathrm{rGURA}_{\mathrm{G}_0} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ can be solved in polynomial time.

8 Example Problem Instance

We will now illustrate the plan generation in two schemes discussed earlier with a sample input state and a set of reachability queries. Fig. 4 defines the common input for both the schemes.

Plan Generation for $\mathrm{RP}_{=}$ *in* $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}\,]$. Fig. 5 shows attributes of user and groups along with user to group direct membership. Table 7 defines set of administrative rules allowed in scheme along with two reachability queries. We will first try to find a reachability plan (if exists) for query q_1 using Algorithm 1.

Initially, $plan$ is set to empty $\langle\rangle$. The initial state is checked to find if it has more attribute values than required in query q_1. In state γ_0, the effective values of user are e_roomAcc(u) = {2.04, 2.03, 3.02}, e_skills(u) = {c, c++}, e_college(u) = {COS}, which are all required in query. The while loop checks if query q_1 is satisfied in state γ_0, which is not true as some values are missing. Now algorithm starts adding new values to the user or its effective groups and also assign new groups to user based on administrative rules defined in Table 7. The first rule requires effective skills of user having value c++ and roomAcc attribute with value 2.04. to add 1.2 value to roomAcc by administrative role BuildAdmin. Since user satisfy these conditions and 1.2 value is not directly assigned in roomAcc(u) and the value is required in q_1, it adds 1.2 value to roomAcc(u). The administrative request add(BuildAdmin, u, roomAcc, 1.2) is also appended to the $plan$. The algorithm again goes through the while loop and checks if q_1 is satisfied. The user is still missing skills attribute value python. The algorithm now tries to add group G_3 to user u. The precondition of canAssign rule is satisfied by user, but the effective values for roomAcc attribute for group G_3 are {3.05, 2.04}, which is not the subset of values required in query. Hence, G_3 cannot be assigned to user u. Next, the algorithm checks rule for skills attribute to add value python and finds that preconditions to add value python are satisfied by user u. It appends the corresponding request add(DeptAdmin, u, skills, python) to $plan$ which results in total of two requests in the plan. The algorithm again checks the new state against q_1 and finds the query is "strict" satisfied. It breaks the while loop and returns plan = add(BuildAdmin, u, roomAcc, 1.2), add(DeptAdmin, u, skills, python).

TABLE 8
Example Problem Instance for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}}-\overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$

Input: problem instance I = $\langle \gamma_0, q \rangle$ **Output:** *plan* or false
$\psi \in \Psi$:
canAddU$_{\mathrm{roomAcc}} = \{\langle \mathrm{BuildAdmin},\ \mathrm{c{+}{+}} \in \mathrm{skills}(u) \wedge \neg(2.04 \in \mathrm{roomAcc}(u)),\ 1.2\rangle\}$,
canAddU$_{\mathrm{skills}} = \{\langle \mathrm{DeptAdmin},\ \mathrm{c} \in \mathrm{skills}(u),\ \mathrm{python}\rangle\}$,
canAddU$_{\mathrm{college}} = \{\langle \mathrm{BuildAdmin},\ \mathrm{matlab} \in \mathrm{skills}(u),\ \mathrm{BUS}\rangle\}$,
canAddU$_{\mathrm{skills}} = \{\langle \mathrm{DeptAdmin},\ \mathrm{c} \in \mathrm{skills}(u) \wedge \mathrm{COS} \in \mathrm{college}(u),\ \mathrm{matlab}\rangle\}$
canAddUG$_{\mathrm{college}} = \{\langle \mathrm{BuildAdmin},\ \mathrm{python} \in \mathrm{skills}(ug) \wedge \neg(2.04 \in \mathrm{roomAcc}(ug)),\ \mathrm{COE}\rangle\}$,
canAssign $= \{\langle \mathrm{DeptAdmin},\ \mathrm{G}_1 \in \mathrm{directUg}(u),\ \mathrm{G}_3\rangle, \langle \mathrm{DeptAdmin},\ \neg(\mathrm{G}_3 \in \mathrm{directUg}(u)),\ \mathrm{G}_5\rangle\}$
Queries:
$q_1 \in$ Q = $\{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02\},\ \ \mathrm{e\_skills(u)} = \{\mathrm{c}, \mathrm{c{+}{+}}, \mathrm{python}\},\ \ \mathrm{e\_college(u)} = \{\mathrm{COS}, \mathrm{COE}\}\}$
$q_2 \in$ Q = $\{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02, 1.2\},\ \ \mathrm{e\_skills(u)} = \{\mathrm{c}, \mathrm{c{+}{+}}, \mathrm{python}\},\ \ \mathrm{e\_college(u)} = \{\mathrm{COS}, \mathrm{COE}\}\}$
$q_3 \in$ Q = $\{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02\},\ \ \mathrm{e\_skills(u)} = \{\mathrm{c}, \mathrm{c{+}{+}}, \mathrm{python}, \mathrm{matlab}\},\ \ \mathrm{e\_college(u)} = \{\mathrm{COS}, \mathrm{COE}, \mathrm{BUS}\}\}$

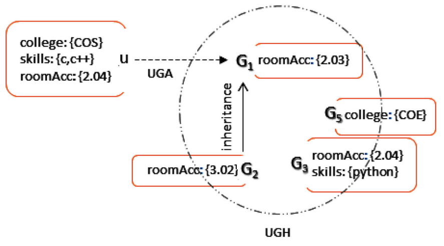


Fig. 6. Initial state for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}}-\overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$.

We now check the satisfiability of query q_2 with the same initial state. Similar to q_1, query is checked against initial state to check extra values and value 1.2 for attribute roomAcc is added to the user and requests is appended to the plan. Second rule allows to add COE for attribute college, but the preconditions are not satisfied by user. We try to add group G_3 but it also adds extra values which are not required in query. It can be noticed that after all the administrative rules are checked, the query cannot be satisfied and hence the algorithm returns false.

Plan Generation for $\mathrm{RP}_{=}$ *in* $[\mathrm{rGURA}_{\mathrm{G}_{1+}}-\overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$*.* Fig. 6 shows user and group attributes along with user to group direct membership. Table 8 defines the set of administrative rules allowed in the scheme and three reachability queries. It should be noted that the rules in Ψ have negation conjuncts and single precondition with direct attributes or group memberships for each attribute value pair or user group. We will start with Algorithm 2 to assign new groups to the user. Once groups are assigned, attributes will be added to user or its newly computed effective groups. If Algorithm 2 doesn't add new groups, the computation will still be done by Algorithm 3 with old effective groups.

Algorithm 2 creates group assignment plan (defined as $plan_{ug}$) to assign new groups to user. Two administrative rules exists in canAssign relation. Since G_3 is not directly assigned to user u, precondition is satisfied and G_3 has value python for skill attribute, which is required in query q_1, algorithm adds G_3 to the set of vertices V_{ug}. Similarly group G_5 is also added to V_{ug}. There are no more canAssign rules, hence the algorithm starts adding edges to the graph. For $(\mathrm{G}_3, \mathrm{G}_5) \in V_{ug}$, since G_3 is a negation conjunct in precondition to add G_5, therefore, directed edge is drawn from G_5 to G_3. As there are no other relevant canAssign rules and vertices pair, it breaks the loop and creates a topological sort of the graph. Sort will have $\{\mathrm{G}_5, \mathrm{G}_3\}$ order and the corresponding plan $plan_{ug}$:= assign$(\mathrm{DeptAdmin},\ \mathrm{u},\ \mathrm{G}_5)$, assign$(\mathrm{DeptAdmin},\ \mathrm{u},\ \mathrm{G}_3)$ is returned. Before proceeding to Algorithm 3, the request in $plan_{ug}$ must be executed to get new effective groups of the user. Algorithm 3 is used to assign attributes to user and newly computed effective groups (which will now have group G_5 and G_3 along with G_1 and G_2). It first checks if the query (q_1) is satisfied in the current state (line 2) which has new direct groups assigned using algorithm 2. Clearly query q_1 is satisfied with new group assignments only, hence the reachability plan for group assignments $plan_{ug}$ is returned.

For queries q_2 and q_3, group assignment plan $plan_{ug}$ is created similarly as above. Therefore, we will follow Algorithm 3 with user's effective groups as G_1, G_2, G_3 and G_5. For q_2, current state do not have value 1.2 for roomAcc attribute. The algorithm first computes *toadd*, which is the set of attribute value pair in q_2

$$\begin{aligned} toadd = \{&\langle \mathrm{roomAcc},\ 2.04\rangle, \langle \mathrm{roomAcc},\ 2.03\rangle, \\ &\langle \mathrm{roomAcc},\ 3.02\rangle, \langle \mathrm{roomAcc},\ 1.2\rangle, \\ &\langle \mathrm{skills},\ \mathrm{c}\rangle, \langle \mathrm{skills},\ \mathrm{c{+}{+}}\rangle, \langle \mathrm{skills},\ \mathrm{python}\rangle, \\ &\langle \mathrm{college},\ \mathrm{COS}\rangle, \langle \mathrm{college},\ \mathrm{COE}\rangle\}. \end{aligned}$$

It then calculates the current attribute value pair for user and its effective groups (Line 4-5)

$$\begin{aligned} cur_{\mathrm{u}} &= \{\langle \mathrm{roomAcc},\ 2.04\rangle, \langle \mathrm{skills},\ \mathrm{c}\rangle, \langle \mathrm{skills},\ \mathrm{c{+}{+}}\rangle, \\ &\quad \langle \mathrm{college},\ \mathrm{COS}\rangle\} \\ cur_{\mathrm{G}_1} &= \{\langle \mathrm{roomAcc},\ 2.03\rangle\} \quad cur_{\mathrm{G}_2} = \{\langle \mathrm{roomAcc},\ 3.02\rangle\} \\ cur_{\mathrm{G}_3} &= \{\langle \mathrm{roomAcc},\ 2.04\rangle, \langle \mathrm{skills},\ \mathrm{python}\rangle\} \\ cur_{\mathrm{G}_5} &= \{\langle \mathrm{college},\ \mathrm{COE}\rangle\}. \end{aligned}$$

The algorithm checks if the current attributes of user and its effective groups are not extra than the values required in the query. Clearly, for query q_2, no extra values are present in current state. The algorithm next computes the positive conjuncts in the preconditions required to add the values in *toadd*. It first calculates for each attribute value pair in *toadd* and then recalculates for each positive preconditions attribute value pair also. For example, positive conjunct for user to add $\langle \mathrm{roomAcc},\ 1.2\rangle$ in *toadd* is $\langle \mathrm{skills},\ \mathrm{c{+}{+}}\rangle$ and for $\langle \mathrm{skills},\ \mathrm{python}\rangle$ in *toadd* is $\langle \mathrm{skills},\ \mathrm{c}\rangle$. Therefore (using line

9), $ppre'_u := \{\langle \text{skills, c++}\rangle, \langle \text{skills, c}\rangle\}$. It then recomputes $ppre_u$ by combining its values with newly computed $ppre'_u$ after removing values already present in $ppre_u$ or cur_u. In this case, no new value is added in $ppre_u$, as both the values in $ppre'_u$ are already present in cur_u. Similarly, the positive preconditions are calculated for each effective groups

$$ppre_{\mathrm{u}} = \{\}, \quad ppre_{\mathrm{G_1}} = ppre_{\mathrm{G_2}} = ppre_{\mathrm{G_5}} = \{\langle \text{skills, python}\rangle\}$$
$$ppre_{\mathrm{G_3}} = \{\}.$$

Next, in line 18, the algorithm checks if rules exists for values required in *toadd* and positive preconditions excluding the current values. Clearly, rule exists for $\langle \text{roomAcc, 1.2}\rangle$ pair and all other values are already present in user or its effective groups. It then calculates negative conjuncts for user and its effective groups in preconditions to add values in *toadd* and positive preconditions excluding current state. For user, $\langle \text{roomAcc, 1.2}\rangle$ has negation conjunct $\langle \text{roomAcc, 2.04}\rangle$ in canAddU$_{\text{roomAcc}}$. Remaining negation conjuncts are as follows:

$$npre_{\mathrm{u}} = \{\langle \text{roomAcc, 2.04}\rangle\}$$
$$npre_{\mathrm{G_1}} = npre_{\mathrm{G_2}} = npre_{\mathrm{G_3}} = \{\langle \text{roomAcc, 2.04}\rangle\}$$
$$npre_{\mathrm{G_5}} = \{\}.$$

Line 21 checks if the negation conjuncts exists in current values of either user or its effective groups. User has $\{\langle \text{roomAcc, 2.04}\rangle\}$ pair in cur_u, therefore, roomAcc attribute cannot get value 1.2 required in q_2 since only single rule exists for user or groups. Hence, the algorithm returns false for query q_2. For query q_3, *toadd* values are

$$\begin{aligned} toadd = \{&\langle \text{roomAcc, 2.04}\rangle, \langle \text{roomAcc, 2.03}\rangle,\\ &\langle \text{roomAcc, 3.02}\rangle, \langle \text{skills, c}\rangle, \langle \text{skills, c++}\rangle,\\ &\langle \text{skills, python}\rangle, \langle \text{skills, matlab}\rangle\\ &\langle \text{college, COS}\rangle, \langle \text{college, COE}\rangle, \langle \text{college, BUS}\rangle\}.\end{aligned}$$

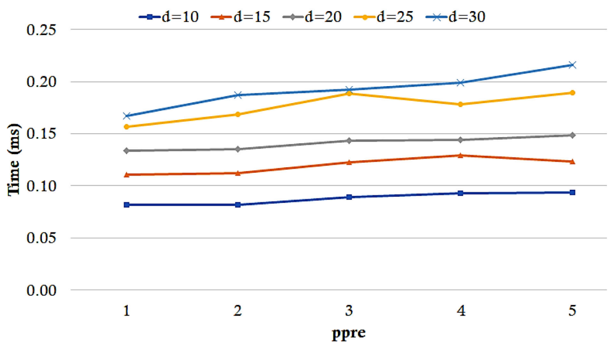
The current values are still the same as defined in query q_2. The algorithm calculates the positive conjuncts in preconditions as

$$ppre_{\mathrm{u}} = \{\langle \text{skills, matlab}\rangle\}$$
$$ppre_{\mathrm{G_1}} = ppre_{\mathrm{G_2}} = ppre_{\mathrm{G_5}} = \{\langle \text{skills, python}\rangle\}$$
$$ppre_{\mathrm{G_3}} = \{\}.$$

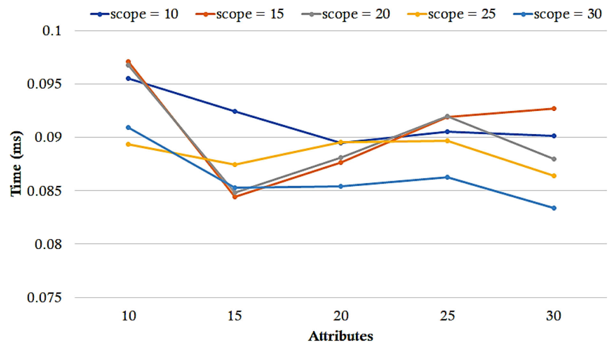
The negation conjuncts are calculated as

$$npre_{\mathrm{u}} = \{\}, npre_{\mathrm{G_1}} = npre_{\mathrm{G_2}} = npre_{\mathrm{G_3}} = \{\langle \text{roomAcc, 2.04}\rangle\}$$
$$npre_{\mathrm{G_5}} = \{\}.$$

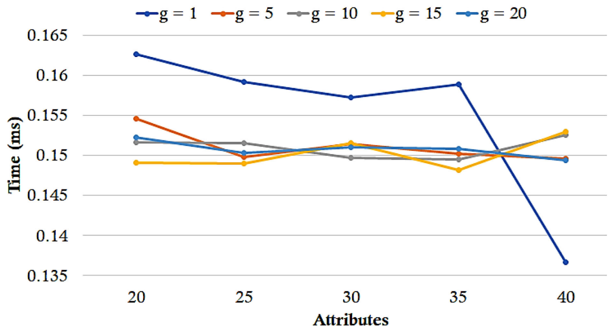
These negation values are not present in user or all of its effective groups. Therefore, the algorithm creates directed graph (line 22-30) with vertices $V := \{\langle \text{skills, matlab}\rangle\}, \{\langle \text{college, BUS}\rangle\}$. Edge in E is drawn from $\{\langle \text{skills, matlab}\rangle\}$ to $\{\langle \text{college, BUS}\rangle\}$ as $\{\langle \text{skills, matlab}\rangle\}$ is a precondition conjunct in rule to add $\{\langle \text{college, BUS}\rangle\}$. Line 31 calculates *valset* which in this case is same as V. Since no cycle exists in the graph, topological sort is created. The final reachability plan to satisfy the query q_3 is $plan :=$ assign(DeptAdmin, u, $\mathrm{G_5}$), assign(DeptAdmin, u, $\mathrm{G_3}$), add (DeptAdmin, u, skills, matlab), add(BuildAdmin, u, college, BUS). The administrative requests must be executed as ordered in the reachability plan.



(a) Varying *ppre* and *d*



(b) Varying *attr* and *scope*



(c) Varying *attr* and *groups*

Fig. 7. Performance of Algorithm 1 with different parameters.

9 Experimental Results

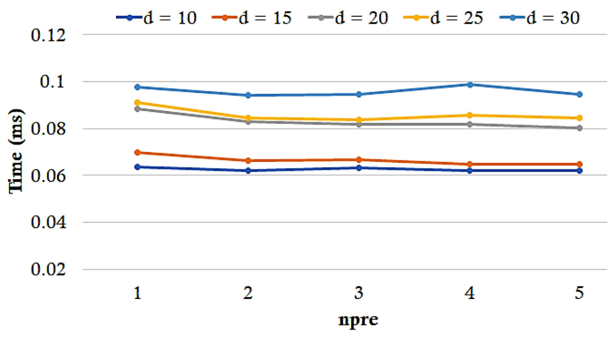
This section describes the performance evaluation on Algorithms 1, 2 and 3 (discussed in Section 7) after varying number of parameters and shows the analysis of the results drawn from these experiments. We measure the time it takes for a successful reachability plan to generate under different rGURA$_{\mathrm{G}}$ instances once a query is requested. We implemented the algorithms in Python using PyCharm.[1] We generated a random pool of attributes and their values for user and groups in the system. The initial state of the system is assigned a user, set of groups, administrative rules allowed in different schemes, group hierarchy, scope of attributes etc. where all are generated randomly. Reachability query and administrative rules are also randomly generated based on the attributes in

1. https://www.jetbrains.com/pycharm/

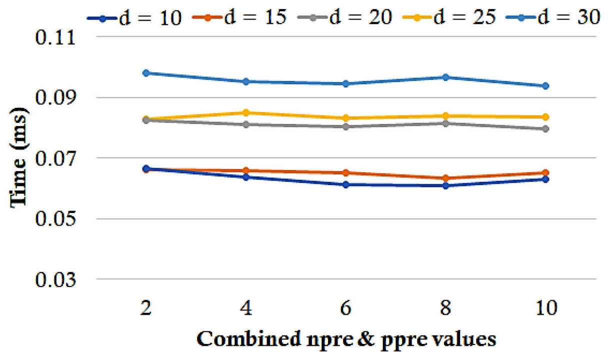
initial state (we do not want to have the query be satisfied in the initial state itself). There are several dynamic parameters which we changed to scale the algorithms: *attr* denotes the number of attributes, *scope* represents the size of the range for values of attributes. *ppre* and *npre* represent the number of positive and negative conjuncts in a rule precondition respectively. *d* represents the total number of desired attribute and value pairs specified in the reachability query which are not already available in the initial state. For example, assume in initial state, ATTR = {College, Skill} and U = {Bob}. In the initial state, College(Bob) = {COS} and Skill(Bob) = {c, Python}. If the query requires, College(Bob) = {COS, COR} and Skill(Bob) = {c, Python, c++, matlab}, then the value of *d* would be 3 as the number of attribute and value pairs that are absent in the initial state is 3. *g* denotes the number of user groups in the system. We generated administrative rules based on the values in *d*. Roughly half of the number of items in the query (for example, with *d* = 30 for Algorithm 1 results in Fig. 7a) would be for canAddUA and the other half canAddUGA and a couple for canAssign. These numbers are approximate since the input and query statements are randomly generated. Therefore, the rules which are based off of those may vary as well. We vary all these parameters in Algorithms performance evaluation under different combinations as explained below. Each data reported is an average over 500 instances generated using the same parameter values. Plan generation execution time was measured on Apple MacBook Pro with a M1 processor with 16 GB of RAM.

Results for Algorithm 1. The evaluation results are in Figs. 7a, 7b and 7c. We vary different parameters except *npre* in Algorithm 1 which denotes the number of negations in preconditions. These negation preconditions are not permitted in the rule conditions of Algorithm 1. Fig. 7a shows the impact of *d* and *ppre* on execution time. We plot the number of *ppre* on x-axis and the execution time (in ms) on y-axis. We plot each curve for a different value of *d*. For these graphs we use, *attr* = 10, *scope* = 40, *g* = 4. As expected, the execution time increases with the increase of number of *ppre* as it more takes time to check the required values for the attributes included in precondition conjuncts of administrative rules with the increasing attribute value pair required in query (denoted by *d*). For instance, the execution time for *attr* = 10, *scope* = 40 with *ppre* = 5 and *d* = 30 is nearly .22 ms, which is much higher than for same parameters but *d* = 10. The major reason is as the reachability query has more attribute values pair to be satisfied, it will take more time to generate a feasible plan.

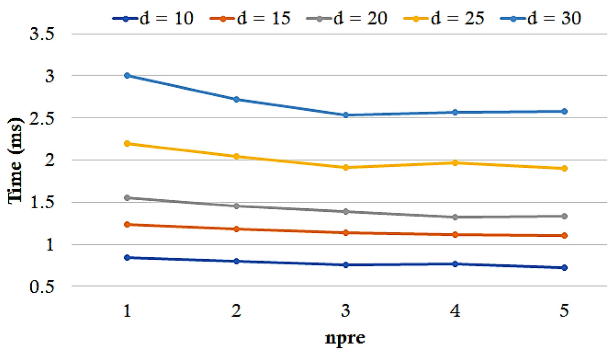
Fig. 7b shows the impact on execution time when we vary the *scope* and number of attributes *attr* (shown as Attributes). We plot the number of attributes on the x-axis and the time consumed for plan on the y-axis. In all problem instances, we ran the experiment with *d* =10, *g* = 4 and *ppre* = 5. Our results show that there is no trend of time increase as the number of attributes increase. However, the total time to execute a query in general is less with larger scope. As we are generating the query randomly with the fixed *d*, the time for solving the query shall overall be small as we have larger initial state due to more attributes or scope. Similarly, as shown in Fig. 7c, we vary the number of attributes and number of groups in the initial state. For all instances, we use *d* = 10, *scope* = 40, *ppre* = 5.



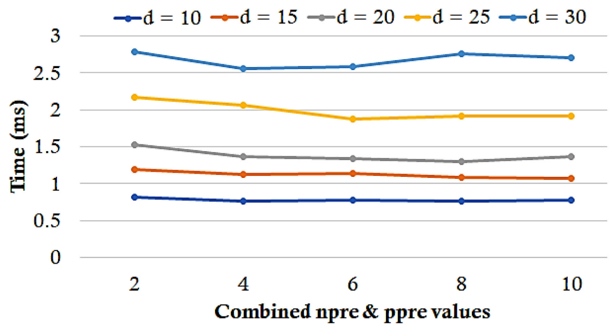
(a) Varying *npre* and *d* for Algorithm 2



(b) Varying *npre* & *ppre* with *d* for Algorithm 2



(c) Varying *npre* and *d* for Algorithm 3



(d) Varying *npre* & *ppre* with *d* for Algorithm 3

Fig. 8. Performance of Algorithms 2 & 3 with Diff. parameters.

We observe, for highest number of groups, *g* = 20, the execution time constantly remains high except for *attr* = 40. It is possible that with the large number of attributes in the initial state, the query is satisfied in early states yielding a fast plan execution. However, the dynamic generator with different queries, initial states and administrative rules, the graphs will not be able to provide a set pattern but it is fair to conjecture that overall time is still less than .30 ms with large number of attributes, values and groups.

*Results for Algorithms 2 & 3.*The results for Algorithm 2 are shown in Figs. 8a and 8b and for Algorithm 3 are illustrated in Figs. 8c and 8d. We calculate the plan generation execution time based on the number of positive *ppre* and negation conjuncts (*npre*) prescribed in a rule precondition. We perform calculations for increasing numbers of *npre* followed by combined *ppre* and *npre* in equal numbers. In practical scenarios, we assume that an administrative rule will not have more than 10 conjuncts to be satisfied for a given attribute value pair or group. Fig. 8a shows the impact of negative precondition conjuncts *npre* (on x-axis) and *d* on the execution time for Algorithm 2. The curve is nearly flat for the different values of *npre* with same *d*. We observe that checking attribute values that are not in the current state (due to *npre* requirement) for user does not take much time while the execution time increases with increasing number of required attribute-value pair in query that are not in the current state since more values are to be satisfied. Fig. 8c captures the behaviour for Algorithm 3 with the same parameter values as for Algorithm 2. It shows similar behavior except the increase in the execution time to run Algorithm 3. This is due to the fact that Algorithm 3 is more complex and requires checking all the different pre-conditions before generating the plan.

Figs. 8b and 8d evaluate the impact of changing both *npre* and *ppre* respectively for Algorithms 2 and 3. We plot the number of *ppre* and *npre* conjuncts on x-axis against execution time and observe the curves for d = 10 to 30. The plan generation time increases in both graphs as the required attribute values in query increase with *d*. It happens as the expected values for satisfying query in current state are higher when the value of *d* is large. The calculation time for the same parameters is higher in the graphs for Algorithm 3. For all the graphs in Fig. 8, we see the increase in consumed time when we increase the value of *d*.

The parameters assumed in our set of experiments are realistic in practical situations, for example, we do not expect users to carry 1000s of attributes or an organization having 100s of groups or administrative rules. However, it does not limit the scalability of our algorithms and the reachability problems can be solved in very reasonable amount of time.

10 Conclusion and Future Work

Attributes based access control defines permissions of entities based on their attributes. In this work, we presented reachability analysis for effective attributes of the user based on the direct attributes assignment to the user or its member user groups. We first stated the HGABAC model and $\mathrm{GURA_G}$ administrative model to provide some background. We defined a restricted form of $\mathrm{GURA_G}$, referred as $\mathrm{rGURA_G}$ and classified three schemes $\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_{1+}}}$ to discuss different reachability solutions. In general, we proved the reachability problem for $\mathrm{rGURA_G}$ scheme is intractable as PSPACE-complete but with certain restrictions, polynomial time algorithms can also be achieved. We empirically evaluated the algorithms under different varying parameters to understand its practicality and use in real world scenarios. In future, we can develop more polynomial algorithms for some restricted forms and perform reachability analysis on other types of queries like effective user groups or minimum number of administrative requests to satisfy query.

Acknowledgments

We are also grateful to Dr. Mahmoud Abdelsalam for his critical comments during the early phase of this work.

References

[1] R. S. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, Sep. 1994.
[2] R. S. Sandhu, "Lattice-based access control models," *IEEE Comput.*, vol. 26, no. 11, pp. 9–19, Nov. 1993.
[3] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 1996.
[4] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *Proc. IFIP Annu. Conf. Data Appl. Secur. Privacy*, 2012, pp. 41–55.
[5] L. Wang *et al.*, "A logic-based framework for attribute based access control," in *Proc. ACM Workshop Formal Methods Secur. Eng.*, 2004, pp. 45–55.
[6] H.-B. Shen and F. Hong, "An attribute-based access control model for web services," in *Proc. IEEE 7th Int. Conf. Parallel Distrib. Comput. Appl. Technol.*, 2006, pp. 74–79.
[7] V. C. Hu *et al.*, "Guide to attribute based access control (ABAC) definition and considerations," NIST Special Publication, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 800–162, 2014.
[8] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *IEEE Comput.*, vol. 48, no. 2, pp. 85–88, Feb. 2015.
[9] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.
[10] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A flexible attribute based access control method for grid computing," *J. Grid Comput.*, vol. 7, no. 2, 2009, Art. no. 169.
[11] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services," in *Proc. IEEE Int. Conf. Web Serv.*, 2005, Art. no. 569.
[12] K. Frikken, M. Atallah, and J. Li, "Attribute-based access control with hidden policies and hidden credentials," *IEEE Trans. Comput.*, vol. 55, no. 10, pp. 1259–1270, Oct. 2006.
[13] M. V. Tripunitara and N. Li, "The foundational work of Harrison-Ruzzo-Ullman revisited," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 1, pp. 28–39, Jan./Feb. 2013.
[14] L. Cirio, I. F. Cruz, and R. Tamassia, "A role and attribute based access control system using Semantic Web technologies," in *Proc. OTM Confederated Int. Conf. Move Meaningful Internet Syst.*, 2007, pp. 1256–1266.
[15] M. J. Covington and M. R. Sastry, "A contextual attribute-based access control model," in *Proc. OTM Confederated Int. Conf. Move Meaningful Internet Syst.*, 2006, pp. 1996–2006.
[16] M. Gupta and R. Sandhu, "Authorization framework for secure cloud assisted connected cars and vehicular Internet of Things," in *Proc. 23rd ACM Symp. Access Control Models Technol.*, 2018, pp. 193–204.
[17] M. Gupta *et al.*, "An attribute-based access control model for secure big data processing in hadoop ecosystem," in *Proc. 3rd ACM Workshop Attribute-Based Access Control*, 2018, pp. 13–24.
[18] M. Gupta *et al.*, "Dynamic groups and attribute-based access control for next-generation smart cars," in *Proc. 9th ACM Conf. Data Appl. Secur. Privacy*, 2019, pp. 61–72.
[19] M. Gupta, F. M. Awaysheh, J. Benson, M. Alazab, F. Patwa, and R. Sandhu, "An attribute-based access control for cloud-enabled industrial smart vehicles," *IEEE Trans. Ind. Informat.*, vol. 17, no. 6, pp. 4288–4297, Jun. 2021.
[20] M. Gupta, J. Benson, F. Patwa, and R. Sandhu, "Secure V2V and V2I communication in intelligent transportation using cloudlets," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2020.3025993.
[21] S. Bhatt, T. K. Pham, M. Gupta, J. Benson, J. Park, and R. Sandhu, "Attribute-based access control for AWS Internet of Things and secure industries of the future," *IEEE Access*, vol. 9, pp. 107 200–107 223, 2021.

[22] D. Servos and S. L. Osborn, "HGABAC: Towards a formal model of hierarchical attribute-based access control," in *Proc. Int. Symp. Found. Pract. Secur.*, 2014, pp. 187–204.

[23] M. Gupta and R. Sandhu, "The GURA_G administrative model for user and group attribute assignment," in *Proc. Int. Conf. Netw. Syst. Secur.*, 2016, pp. 318–332.

[24] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 1, pp. 105–135, 1999.

[25] X. Jin, R. Krishnan, and R. Sandhu, "A role-based administration model for attributes," in *Proc. 1st Int. Workshop Secure Resilient Archit. Syst.*, 2012, pp. 7–12.

[26] X. Jin, R. Krishnan, and R. Sandhu, "Reachability analysis for role-based administration of attributes," in *Proc. ACM Workshop Digit. Identity Manage.*, 2013, pp. 73–84.

[27] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Commun. ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[28] M. V. Tripunitara and N. Li, "A theory for comparing the expressive power of access control models," *J. Comput. Secur.*, vol. 15, no. 2, pp. 231–272, 2007.

[29] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 4, pp. 391–420, 2006.

[30] N. Li, J. C. Mitchell, and W. H. Winsborough, "Beyond proof-of-compliance: Security analysis in trust management," *J. ACM*, vol. 52, no. 3, pp. 474–514, 2005.

[31] R. S. Sandhu, "The schematic protection model: Its definition and analysis for acyclic attenuating schemes," *J. ACM*, vol. 35, no. 2, pp. 404–432, 1988.

[32] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan, "Policy analysis for administrative role based access control," in *Proc. IEEE Comput. Secur. Found. Workshop*, 2006, pp. 13 pp.–138.

[33] R. S. Sandhu, "The typed access matrix model," in *Proc. IEEE Comput. Soc. Symp. Res. Secur. Privacy*, 1992, pp. 122–136.

[34] R. J. Lipton and L. Snyder, "A linear time algorithm for deciding subject security," *J. ACM*, vol. 24, no. 3, pp. 455–464, 1977.

[35] A. Schaad and J. D. Moffett, "A lightweight approach to specification and analysis of role-based access control extensions," in *Proc. 7th ACM Symp. Access Control Models Technol.*, 2002, pp. 13–22.

[36] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough, "Towards formal verification of role-based access control policies," *IEEE Trans. Dependable Secure Comput.*, vol. 5, no. 4, pp. 242–255, Fourth Quarter 2008.

[37] P. V. Rajkumar and R. Sandhu, "Safety decidability for pre-authorization usage control with finite attribute domains," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 3, pp. 465–478, May/Jun. 2020.

[38] S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A logical language for expressing authorizations," in *Proc. IEEE Symp. Secur. Privacy*, 1997, pp. 31–42.

[39] L. Cholvy and F. Cuppens, "Analyzing consistency of security policies," in *Proc. IEEE Symp. Secur. Privacy*, 1997, pp. 103–112.

[40] A. K. Bandara, E. C. Lupu, and A. Russo, "Using event calculus to formalise policy specification and analysis," in *Proc. IEEE 4th Int. Workshop Policies Distrib. Syst. Netw.*, 2003, pp. 26–39.

[41] T. Jaeger, X. Zhang, and A. Edwards, "Policy management using access control spaces," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 3, pp. 327–364, 2003.

[42] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proc. IEEE 27th Int. Conf. Softw. Eng.*, 2005, pp. 196–205.

[43] S. D. Stoller, P. Yang, C. R. Ramakrishnan, and M. I. Gofman, "Efficient policy analysis for administrative role based access control," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 445–455.

[44] S. D. Stoller, P. Yang, M. I. Gofman, and C. Ramakrishnan, "Symbolic reachability analysis for parameterized administrative role-based access control," *Comput. Secur.*, vol. 30, no. 2, pp. 148–164, 2011.

[45] P. Gupta, S. D. Stoller, and Z. Xu, "Abductive analysis of administrative policies in rule-based access control," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 5, pp. 412–424, Sep./Oct. 2014.

[46] J. B. D. Joshi, E. Bertino, and A. Ghafoor, "An analysis of expressiveness and design issues for the generalized temporal role-based access control model," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 157–175, Second Quarter 2005.

[47] J. Crampton and G. Loizou, "Administrative scope: A foundation for role-based administrative models," *ACM Trans. Inf. Syst. Secur.*, vol. 6, pp. 201–231, 2003.

[48] W. J. Savitch, "Relationships between nondeterministic and deterministic tape complexities," *J. Comput. Syst. Sci.*, vol. 4, no. 2, pp. 177–192, 1970.

[49] C. Bäckström and B. Nebel, "Complexity results for SAS+ planning," *Comput. Intell.*, vol. 11, no. 4, pp. 625–655, 1995.

Maanak Gupta (Member, IEEE) received the BTech degree in computer science and engineering from Kuruskhetra University, Kurukshetra, Haryana, India, the MS degree in information systems from Northeastern University, Boston, Massachusetts, and the MS and PhD degrees in computer science from the University of Texas at San Antonio (UTSA), San Antonio, Texas, and has also worked as a postdoctoral fellow with the Institute for Cyber Security (ICS), UTSA. He is currently an assistant professor in computer science with Tennessee Technological University, Cookeville, USA. His research interests include security and privacy in cyber space focused in studying foundational aspects of access control and their application in technologies including cyber physical systems, cloud computing, IoT, and Big Data. He has worked in developing novel security mechanisms, models and architectures for next generation smart cars, smart cities, intelligent transportation systems, and smart farming.

Ravi Sandhu (Fellow, IEEE) is the founding executive director and chief scientist with the Institute for Cyber Security, University of Texas at San Antonio, Texas, where he holds the Lutcher Brown endowed chair in cyber security. He is a fellow of the ACM and AAAS and an inventor on 30 patents. He was the past editor-in-chief of the *IEEE Transactions on Dependable and Secure Computing*, past founding editor-in-chief of the *ACM Transactions on Information and System Security* and a past chair of ACM SIGSAC. He founded ACM CCS, SACMAT, and CODASPY, and has been a leader in numerous other security conferences. His research has focused on security models and architectures, including the seminal role-based access control model. His papers have accumulated more than 46,000 Google Scholar citations, including more than 9,500 citations for the RBAC96 paper.

Tanjila Mawla is currently working toward the doctoral degree with the Department of Computer Science, Tennessee Technological University, Cookeville, Tennessee. Her research interests include access control, formal security models, and enforcement mechanisms for smart ecosystems.

James Benson received the BSc and MSc degrees in physics from Clarkson University, Potsdam, New York, in 2007 and 2009, respectively, and the MSc degree in electrical engineering from the University of Texas at San Antonio (UTSA), San Antonio, Texas, in 2016. He has worked with the Texas Renewable Energy Institute (TSERI) and the Open Cloud Institute (OCI), UTSA, where he was assisting with data analytics and various research projects. He is currently working as a technology research analyst II with the Institute for Cyber Security (ICS) and the Center for Security and Privacy Enhanced Cloud Computing (C-SPECC), UTSA. His research interests include cyber physical systems, cloud computing, and automation.