Formal Analysis of ReBAC Policy Mining Feasibility

Shuvra Chakraborty and Ravi Sandhu
Institute for Cyber Security (ICS) and NSF Center for Security and Privacy Enhanced Cloud Computing (C-SPECC)
Department of Computer Science, The University of Texas at San Antonio
San Antonio, Texas, USA
{shuvra.chakraborty,ravi.sandhu}@utsa.edu

ABSTRACT

Relationship-Based Access Control (ReBAC) expresses authorization in terms of various direct and indirect relationships amongst entities, most commonly between users. The need for ReBAC policy mining arises when an existing access control system is reformulated in ReBAC. This paper considers the feasibility of ReBAC policy mining in context of user to user authorization, such as arises in various social and business contexts. In accordance with the policy mining literature, we assume that complete data is provided regarding user to user authorizations for a given user set, along with complete relationship data amongst these users comprising a labeled relationship graph. A ReBAC policy language is also specified. ReBAC policy mining seeks to formulate a ReBAC policy with the given policy language and relationship graph, which is exactly equivalent to the given authorizations. ReBAC policy mining feasibility problem asks whether such a policy exists and if so to provide the policy. We investigate this problem in context of different ReBAC policy languages which differ in the relationships, inverse relationships and non-relationships that can be used to build the policy. We develop a feasibility detection algorithm and analyze its complexity. We show that our policy languages are progressively more expressive as we introduce additional capability. In case of infeasibility, various solution approaches are discussed.

CCS CONCEPTS

• **Security and privacy → Access control**.

KEYWORDS

Access control; Policy mining; Relationship-Based Access Control

ACM Reference Format:
Shuvra Chakraborty and Ravi Sandhu. 2021. Formal Analysis of ReBAC Policy Mining Feasibility. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy (CODASPY '21), April 26–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3422337.3447828

CODASPY '21, April 26–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8143-7/21/04…$15.00
https://doi.org/10.1145/3422337.3447828

1 INTRODUCTION

In accordance with the emerging popularity of various Online Social Network (OSN) applications such as Facebook, Twitter, Instagram, LinkedIn, effective access control is required to protect the huge amount of personal data shared online from unauthorized access. Popular OSN systems have employed various policies to mitigate user's privacy concern. For instance, Facebook allows the user to set the privacy to only himself, all friends, a particular list of friends, or public while creating a photo album; while LinkedIn allows the user to keep their job seeking status to be seen by recruiters only. In addition to individual user policies, there are some policies applied system-wide as well, for example, the profile picture is public applies to all users on Facebook. Subsequently, academic research has been conducted to study and develop access control mechanisms for OSNs. Relationship-Based Access Control (ReBAC) is a prominent one in that regard. While ReBAC is relatively new, there is significant literature on defining the structure and features of ReBAC models. While its primary motivation originated from its applications in OSNs, it offers promising capabilities in more general applications [2].

Recently, the ReBAC policy mining problem has begun to attract attention. This problem seeks to automate the process of obtaining an equivalent ReBAC policy when a complete access control system along with supporting relationship data is given. ReBAC policy mining algorithms offer promising advancement in automating policy generation, whereas manual effort requires more time, and could be error-prone. In this paper, we investigate the ReBAC policy mining approach from a novel perspective. We study the feasibility of the ReBAC policy mining process, in context of various ReBAC rule structures.

ReBAC policy mining approaches such as [4–8] permit use of the unique identity (id) of entities (e.g., users and resources) in the generated ReBAC policies. Hence, ReBAC policy mining is always feasible. We believe that use of such ids is contrary to the core ReBAC spirit. Thereby, determining feasibility becomes a significant question in mining ReBAC policies. In case of infeasibility, we propose various solutions as an alternative to using ids in ReBAC policy generation. Throughout this study, we use the terms ReBAC policy and ReBAC rule set interchangeably.

Our central contributions in this paper are as follows.

(1) The first formal notion of ReBAC RuleSet Existence Problem (RREP) is developed.
(2) A novel algorithm for ReBAC policy mining feasibility detection is presented along with examples.
(3) RREP variations based on increasingly powerful ReBAC policy languages are identified and shown to be solved by essentially the same algorithm mentioned above.

(4) Resolution of infeasibility is addressed, and a formal solution is proposed.
(5) Significant directions of future work are discussed.

The rest of the paper is organized as follows. Section 2 discusses relevant background for this paper. Section 3 formalizes the core ReBAC RuleSet Existence Problem (RREP-0) and provides a feasibility detection algorithm along with associated proofs and complexity analysis. Section 4 extends the RREP with different ReBAC policy languages along with examples which demonstrate the increasing power of these languages. Section 5 develops an infeasibility solution for RREP-0, and discusses other solution approaches. Section 6 discusses a case study and implementation details. Finally, Section 7 discusses the pros and cons of the proposed approach to ReBAC policy mining, as well as significant directions for enhancements.

2 RELATED WORK

The rule set existence problem has been previously defined for Attribute-Based Access Control (ABAC) from given enumerated authorization [9] and given Role-Based Access Control (RBAC) authorization [10]. This paper is the first to consider this problem in context of ReBAC. We review the literature on ReBAC models and ReBAC policy mining below.

The recent proliferation of OSNs has accelerated the research of finding an access control paradigm which is different from traditional dominant access control models like ABAC [17], RBAC [14], etc. According to the early literature, ReBAC policy is characterized by the explicit tracking of interpersonal relationships between users [15]. ReBAC is a general-purpose access control model which supports the natural expression of parameterized roles, the composition of policies, and the delegation of trust [16]. A further extended hybrid-logic based ReBAC policy is given in [3].

In general, given an OSN, users and resources are interconnected via various types of relationships. In order to specify ReBAC policies, particular relationship directions between users and resources can play significant roles. For example, [12] specifies ReBAC policies based on user to user (U-U) relationships in OSN. Similarly, [1] uses resource to resource (R-R), and [11] uses resource to user (R-U) and vice versa to express ReBAC policies. In addition, [2] presents a comparative analysis of expressive power and performance implications between ReBAC and ABAC features, [19] does an extensive analysis when the OSN is updated, and [13] proposes ReBAC to be integrated with ABAC to enhance the capability and allows finer-grained controls.

Given an access control system along with supporting data, ReBAC policy mining algorithms find the equivalent ReBAC policy. This provides partial automation to the overall migration process, reduces cost and uses some measures to find the most efficient rule set. A few works on ReBAC policy mining are discussed briefly as follows. The work in [6] presents ReBAC as an object-oriented extension of ABAC where the "class" structure is able to realize the relationship between various entities, beyond the user and resources paradigms. In [7], the work in [6] is basically extended, where heuristic-guided greedy and grammar-based evolutionary algorithms for ReBAC policy mining are presented. A further extension is proposed in [5], to the evolutionary ReBAC policy mining in [7]. The extended ReBAC policy mining in [5] follows the simplification as well as feature selection by using neural network resulting in a more scalable and efficient algorithm. Some other ReBAC policy mining algorithms use decision tree [4], incomplete and noisy input data [8], and mine ReBAC policies from graph transition [18]. In comparison with [4, 8, 18], this feasibility study is limited to static relationship graph with complete input information only.

Compared to [4–8], our work in this paper concentrates on whether ReBAC mining is feasible or not without altering the core spirit of ReBAC, i.e., relationships should be the key to express policies and use of ids is prohibited. This is a fundamental difference since ReBAC policy mining is always feasible with ids. The feasibility issue in ReBAC policy mining has been considered for the first time in this paper, to the best of our knowledge.

Two relatively similar research works on ABAC ruleset existence problem can be found in [9] and [10]. In [9], where an authorization state and supporting attribute data are given as input, ABAC ruleset existence problem has been introduced for the first time. Based on the definition of conflict-free partition, [9] introduces a feasibility detection algorithm and an infeasibility correction approach. In [10], the core solution provided in [9] has been extended to accomplish the solution of ABAC ruleset existence problem when an RBAC system along with supporting attribute data are given as input. Similar to our work in this paper, [9, 10] also prohibit the use of ids in their respective rulesets.

3 ReBAC RULESET EXISTENCE PROBLEM

This section develops the formal definition of the ReBAC RuleSet Existence Problem (RREP). As we are going to investigate variations of RREP later in this paper, we call the the core RREP problem defined in this section as RREP-0.

3.1 Preliminaries

A user is an entity who performs operations (also called actions). An operation is an act performed by a user on another user. A user can be an initiator or a target of an operation. The finite (but unbounded) set of current users is denoted as U. The finite set of operations is denoted by OP, where each operation in OP is independently authorized.

Given that a user requests to perform an operation on another user, every access control system must define a checkAccess function to decide whether or not this operation is permitted or denied. The specification of checkAccess, typically as a logical formula, depends upon the details of the underlying access control model.

Definition 3.1. **checkAccess**
$checkAccess: U \times U \times OP \rightarrow \{True, False\}$ where U and OP are finite sets of users and operations respectively. A user $u \in U$ is allowed to perform operation $op \in OP$ on a user $v \in U$ iff $checkAccess(u, v, op)$ is True.

Without loss of generality, we assume OP is the singleton set $\{op\}$, since each operation is independently authorized. For simplicity, OP is thereby omitted from further definitions. For a specific access control model M we write $checkAccess_M(u, v)$.

An access request is a tuple $\langle u, v \rangle$, where $u, v \in U$ and $u \neq v$, which specifies user u has requested to perform operation op on

user v. A simple authorization system, where user to user tuples are used directly to control access authorization is as follows.

Definition 3.2. **Enumerated Authorization System (EAS)**
An EAS is a tuple $\langle U, AUTH, checkAccess_{EAS}\rangle$ where, U is the finite set of users, $AUTH \subseteq U \times U$, is the authorization relation where $\forall(u,v) \in AUTH.u \neq v$, and $checkAccess_{EAS}(u,v) \equiv (u,v) \in AUTH$.

For example, given the set of users $U = \{Alice, Bob, Cathy\}$ and $AUTH = \{(Alice, Bob), (Bob, Cathy)\}$, an access request $\langle Alice, Bob\rangle$ is granted whereas $\langle Alice, Cathy\rangle$ is denied. $AUTH$ is essentially an access matrix.

Relationships are represented as a directed labeled graph.

Definition 3.3. **Relationship Graph (RG)**
The Relationship Graph RG = (V, E, Σ) of a system is a directed labeled graph where,
i) V is the set of vertices in RG, representing the current set of users,
ii) $E \subseteq V \times V \times \Sigma$ is a finite set of labeled directed edges where Σ is a finite set of relation type specifiers.
An edge $(u, v, \sigma) \in E$, $u \neq v$, represents the relation $\sigma \in \Sigma$ from user $u \in V$ to $v \in V$ in RG where σ is the edge label.

For example, in Fig. 1 let F represent the friend relation. Then Alice is a friend of Bob, but not vice versa, whereas Cathy is a completely isolated user.

Direct relationships are represented as edges in RG, while indirect relationships are represented as paths. For our purpose, it is convenient to define path in two steps as follows.

Definition 3.4. **Linked Sequence of Vertices**
Given RG = (V, E, Σ) and a vertex pair $(u, v) \in V \times V$ where $u \neq v$, a (simple) linked sequence of vertices is a set of triples where the terminating (i.e., second) vertex of each triple is same as the starting (i.e., first) vertex of the next triple given by $\langle (u, v_i, \sigma_w), (v_i, v_j, \sigma_x), ..., (v_k, v_l, \sigma_y), (v_l, v, \sigma_z)\rangle$, where $u, v_i, v_j, ..., v_k, v_l, v \in V$, and $\sigma_w, \sigma_x, ..., \sigma_y, \sigma_z \in \Sigma$, such that once a vertex v_i occurs as a start vertex it cannot be the terminating vertex in subsequent triples.

Definition 3.5. **Path in Relationship Graph**
A (simple) linked sequence of vertices is a (simple) path from u to v if each triple belongs to E in RG, i.e., it is an edge. The path label of a path is $\sigma_w\sigma_x.....\sigma_y\sigma_z$. Its length is the number of triples, or equivalently the number of symbols in the path label.

Since we only consider simple paths in this paper we will often drop the simple qualifier. It should be noted that Def. 3.4 and 3.5 would traditionally be merged to define a path, but separating them makes it convenient to define path variations later in Def. 4.1. For convenience, given a path p in RG we understand pathLabel(p) to denote the path label of path p.

A crucial component of ReBAC is a set of rules called the ReBAC policy, formally defined as follows:

Definition 3.6. **ReBAC Policy**
A ReBAC policy, POL_{ReBAC} is a tuple, given by $\langle \Sigma, RuleSet\rangle$ where:

- Σ denotes the finite set of relation type specifiers in the system.
- RuleSet is a set of rules where, for each operation $op \in OP$, RuleSet contains the single rule $Rule_{op}$. Each $Rule_{op}$ is specified using the grammar below.

$Rule_{op}$ ::= $Rule_{op} \vee Rule_{op} \mid pathRuleExpr$
$pathRuleExpr$ ::= $pathRuleExpr \wedge pathRuleExpr \mid pathLabelExpr$
$pathLabelExpr$ ::= $pathLabelExpr.pathLabelExpr \mid edgeLabel$
$edgeLabel$ ::= $\sigma, \sigma \in \Sigma$

Here "." is the concatenation operator. As stated earlier, it suffices to consider OP to be a singleton, so RuleSet consists of a single rule. The $Rule_{op}$ expression consists of disjunction of pathRuleExpr, where each pathRuleExpr consists of conjunction of pathLabelExpr. The pathLabelExpr is a concatenated string of relationship type specifiers. The $Rule_{op}$ evaluation procedure is described in Def. 3.7.

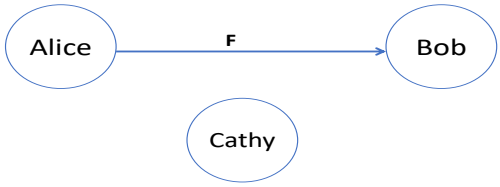

Figure 1: RG with V = {Alice, Bob, Cathy}, E = {(Alice, Bob, F)}, and Σ = {F}.

This leads to the following definition of a ReBAC system.

Definition 3.7. **ReBAC System**
A ReBAC system is a tuple, $\langle RG, POL_{ReBAC}, checkAccess_{ReBAC}\rangle$ where $checkAccess_{ReBAC}$*(a:V, b:V)* is evaluated as follows: (i) for each pathLabelExpr in $Rule_{op}$ substitute True if there exists a simple path p from a to b in RG with path label pathLabelExpr, otherwise substitute False, (ii) evaluate the resulting boolean expression.

For example, consider Fig. 1 with $Rule_{op} = F$. Given an access request ⟨Alice, Bob⟩, there is a simple path from Alice to Bob with path label F so True is substituted for F and $Rule_{op}$ evaluates to True whereby the access request is granted.

The definitions provided above bring us to definition of the central problem addressed in this paper.

Definition 3.8. **ReBAC RuleSet Existence Problem (RREP-0)**
Given an EAS = ⟨U,AUTH,$checkAccess_{EAS}$⟩ and RG = (V,E,Σ) with V=U, does there exist a RuleSet as in Def. 3.6 so that the resulting ReBAC system satisfies:

$$(\forall u, v \in U)[checkAccess_{ReBAC}(u,v) \Leftrightarrow checkAccess_{EAS}(u,v)]$$

Such a RuleSet, if it exists, is said to be a suitable RuleSet, otherwise the problem is said to be infeasible.

For example, for the RG in Fig. 1, a suitable RuleSet exists only if the given AUTH = $\{(Alice, Bob)\}$, with $Rule_{op} = F$. Any other AUTH relation will not have a suitable ReBAC RuleSet.

RuleSet Generality

A natural question to investigate at this point is the generality of our ReBAC RuleSet structure. We consider three criteria in this regard: variety of entities available, expressiveness of policy language and relationship depth. In this paper, we limit our scope to user to user relationships only which is a common case in OSNs. More generally, ReBAC policy mining may incorporate multiple entity types. For example, [6] uses the familiar class-object concept in their ReBAC rules, where each class represents a a particular entity type and an object is an instance of a class. In comparison with [6], we deal with a single class User. We discuss expressiveness in Section 4 and will show by examples that our rule structure is not the most general one. Given a vertex pair (a,b) in RG and simple path p from a to b, the relationship depth is the length of the path. With finite RG, the relationship depth is inherently limited by the maximum simple path length between any vertex pair in RG. In [6], relationship depth is provided as algorithm input. While in our paper relationship depth is not provided as input, a slight modification would accommodate this. Specifically, adding a constraint in line 5 of Algorithm 1 to limit the maximum simple path length to a provided value. Other constraints could be similarly enforced. For example, a constraint that limits the number of path labels used in the conjunctive term generation in line 16-17 of Algorithm 1 to a given numeric value.

3.2 Feasibility Detection Algorithm

In this subsection, the feasibility detection Algorithm 1 for RREP-0 is presented along with proofs and complexity analysis. The algorithm iterates through each tuple $(a, b) \in AUTH$, and either finds a rule that is correct for (a, b) or deems the tuple to be infeasible and records it in failedAuthList. In each iteration it computes all possible simple paths from a to b to find whether the resulting collection of pathLabels is collectively satisfied by any unauthorized tuple (i.e., a tuple not in $AUTH$). The function FindAllSimplePath, which is able to find all simple paths between any vertex pair in RG, is described briefly in appendix A. If (a,b) is disconnected in RG then it is infeasible. Otherwise, all possible pathLabels for (a,b) are generated in line 9 as in Def. 3.5. If there exists any unauthorized vertex pair which satisfies all possible pathLabels from a to b, (a,b) is infeasible as in line 14. The $Rule_{op}$ is updated otherwise and (a,b) is removed from further consideration, as shown in line 16-17.

At the end, if rule generation is not feasible for any particular tuple in AUTH, i.e., failedAuthList is not empty, the algorithm returns an infeasible result along with all infeasible tuples in failedAuthList. Another alternative is to abort at the point where first infeasible tuple is encountered if failedAuthList is not available. If rule generation is feasible for every tuple $(a, b) \in AUTH$, $Rule_{op}$ is generated and feasible status is returned.

Theorem 3.9. *The overall complexity of RREP-0 feasibility detection Algorithm 1 is* $O(|V|^4 \times (|E|!))$.

Proof:

In order to compute Algorithm 1 complexity, Algorithm 2 and 3 in appendix A are needed to be considered first. Algorithm 2 finds the set all possible simple paths between a vertex pair in RG using a variant of DFS in Algorithm 3. Since it considers only simple path, the overall complexity of Algorithm 2 is $O(|E|!)$, considering $|V| \leq |E|$. Therefore, the complexity of line 5 and 9 in Algorithm 1 is $O(|E|!)$. In line 10-11, the SAT_{ab} function computation takes overall $O(|V|^2 \times (|E|!))$. The computation complexity of finding set intersections in line 10 takes $O(|E|!)$. Line 13-17 produces trivial complexity compared to the others. The while loop in line 4-17 runs $|AUTH| < |V|^2$ times. Hence, the overall complexity of Algorithm 1 is $O(|V|^4 \times (|E|!))$.

The asymptotic complexity of the current approach is high, especially because computation of all possible simple paths between any pair of vertices in RG gives the ultimate lower bound. However, RG can be a sparse one. Also, it can be easily noticed that pre-computing and storing all possible simple paths between any pairs in RG regardless of the AUTH can effectively reduce the computation time inside the loop. Moreover, for many such practical problems heuristic solutions are often effective. Later in this study, it has been discussed that our ReBAC rule structure is not the most general one. Feasibility algorithm can certainly change based on the variety of ReBAC rule structures. Therefore, overall complexity of determining ReBAC policy mining feasibility can vary based on such factors. A detailed study of these is out of scope of this paper.

The correctness proof of Algorithm 1 is as follows.

Algorithm 1 ReBAC RuleSet Existence Problem-0 Algorithm

Input: An EAS $\langle$ U,AUTH,$checkAccess_{EAS}\rangle$ and a RG = (V,E,Σ) where V=U

Output: Feasible/infeasible status. If feasible $\rightarrow$ generate ReBAC rule, return "infeasible" and set of infeasible authorization tuples otherwise.

1: $Rule_{op} := NULL$
2: failedAuthList := $\emptyset$
3: AUTHset := AUTH //copying AUTH
4: **while** $\exists(a, b) \in AUTHset$ **do**
5: $SP(a, b)$:= FindAllSimplePath(a,b, RG) //see appendix A
6: **if** $SP(a, b) = \emptyset$ **then**
7: $failedAuthList := failedAuthList \cup \{(a, b)\}$ //Not Feasible for (a,b) tuple
8: $AUTHset \setminus := \{(a, b)\}$ and **Continue**
9: $PATHLABEL(a.b) := \{pathLabel(p) | p \in SP(a, b)\}$
10: **for** each $pl \in PATHLABEL(a.b)$ **do**
11: $SAT_{ab}(pl) = \{(c, d) \in V \times V |$ there exists a simple path s from c to d in RG, c$\neq$d, (c,d)$\notin$AUTH, pl=pathLabel(s)$\}$
12: $Q_{ab} := \bigcap_{pl \in PATHLABEL(a.b)} SAT_{ab}(pl)$
13: **if** $Q_{ab} \neq \emptyset$ **then**
14: $failedAuthList := failedAuthList \cup \{(a, b)\}$ //Not Feasible for (a,b) tuple
15: $AUTHset \setminus := \{(a, b)\}$ and **Continue**
16: **if** $Rule_{op}$ is NULL **then** $Rule_{op} := \bigwedge_{pl \in PATHLABEL(a.b)} pl$
else $Rule_{op} := Rule_{op} \vee \bigwedge_{pl \in PATHLABEL(a.b)} pl$
17: $AUTHset \setminus := \{(a, b)\}$
18: **if** failedAuthList is $\emptyset$ **then return** ("feasible", $Rule_{op}$) **else return** ("infeasible", $Rule_{op}$, failedAuthList)

Theorem 3.10. *Given a RREP-0 instance as in Def. 3.8, a suitable RuleSet exists iff Algorithm 1 generates the* $Rule_{op}$.

Proof:
Assume, Algorithm 1 generates the $Rule_{op}$. According to Algorithm 1, for each $(a, b) \in AUTH$, all possible paths from a to b in RG are searched over to find the collection of pathLabel(p) where p is a simple path from a to b, such that there exists no unauthorized tuple $(c, d) \in V \times V \setminus AUTH, c \neq d$ where the collection of pathLabel(q) is a superset of the collection of pathLabel(p), where q is a simple path from c to d in RG. In Algorithm 1, $Rule_{op}$ consists of disjunctions of such conjunction of the collection of pathLabel(p), generated for each $(a, b) \in AUTH$. By the definition of checkAccess in Def. 3.7, the generated $Rule_{op}$ evaluates to true for each $(a, b) \in AUTH$ while denying all $(c, d) \in V \times V \setminus AUTH, c \neq d$. Hence, $Rule_{op}$ constitutes a suitable RuleSet.

To prove the opposite direction, assume a suitable RuleSet $Rule'_{op}$ constituted by Def. 3.6 exists. Therefore, by the definition of RREP-0, $Rule'_{op}$ evaluates to true for each $(a, b) \in AUTH$ while denying all unauthorized tuple $(c, d) \in V \times V \setminus AUTH, c \neq d$. By the procedure of $Rule'_{op}$ evaluation provided in Def. 3.7, there exists at least a conjunctive term in $Rule'_{op}$ which is true for a $(a, b) \in AUTH$ where for all pathLabelExprs in the corresponding conjunctive term, there exists a simple path p from a to b in RG such that pathLabel(p) = pathLabelExpr. According to Algorithm 1, for each $(a, b) \in AUTH$, all possible paths from a to b in RG are searched over to find such conjunction of the collection of pathLabel(p) and $Rule_{op}$ consists of disjunction of such conjunctions, generated for each $(a, b) \in AUTH$. Thereby, Algorithm 1 generates the feasible status and $Rule_{op}$, where each conjunctive term denoted by t' in $Rule'_{op}$ must have at least a conjunctive term t in $Rule_{op}$ where the pathLabels in t' are a subset of the pathLabels in t. Hence, the claim holds in both directions and Theorem 3.10 is proved.

RREP-0 is the core of our ReBAC feasibility analysis. An example of ReBAC rule generation is discussed in Section 6.

4 VARIATIONS OF ReBAC RULESET EXISTENCE PROBLEM

By definition of RREP-0 in Def. 3.8, there are three key factors which affect the feasibility detection process: i) the authorization relation AUTH, ii) $Rule_{op}$ structure, and iii) RG. For example, an AUTH relation can be symmetric or asymmetric, RG can be directed or undirected, and the $Rule_{op}$ specification grammar can be modified to add more or less expressive power. In this section, we consider some RREP variations focusing on ReBAC rule structure.

4.1 Proposed RREP Variations

The following discussion proposes four variations of RREP. According to Def. 3.3, the given RG is a directed labeled graph. Therefore, Algorithm 1 can work with directed RG only. Given an undirected relationship graph $RG^Y = (V, E^Y, \Sigma)$, an equivalent directed labeled relationship graph RG = (V, E, Σ) can be generated by enhancing the set of edges. For each edge $(a, b, \sigma) \in E^Y$, symmetric edges (a, b, σ) and (b, a, σ) are added to E. For each $(u, v) \in AUTH$, symmetric authorization tuples (u, v) and (v, u) are added to updated AUTH relation as well. It is evident that the undirected RG along with undirected AUTH can be reduced to core RREP-0 and Algorithm 1 can be deployed to solve the feasibility detection. Thus it suffices to consider directed RG.

Table 1: Path variations in RG

Characteristics	SCP	SPP	SCPP
$(a, b, \sigma) \rightarrow (a, b, \sigma) \in E, \sigma \in \Sigma$	✓	✓	✓
$(a, b, \overline{\sigma}) \rightarrow (a, b, \sigma) \notin E, \overline{\sigma} \in \overline{\Sigma}$	✓		✓
$(a, b, \sigma^{-1}) \rightarrow (b, a, \sigma) \in E, \sigma^{-1} \in \Sigma^{-1}$		✓	✓
$(a, b, \overline{\sigma}^{-1}) \rightarrow (b, a, \sigma) \notin E, \overline{\sigma}^{-1} \in \overline{\Sigma}^{-1}$			✓

Before proceeding to the other variations of RREP, three extended sets of relationships are defined as follows for a given Σ.

- $\overline{\Sigma} = \{\overline{\sigma} | \sigma \in \Sigma\}$. For each relation type specifier $\sigma \in \Sigma$, $\overline{\sigma}$ denotes "no σ relation". Therefore, $\overline{\Sigma}$ is the set of non-relationship type specifiers in RG.
- $\Sigma^{-1} = \{\sigma^{-1} | \sigma \in \Sigma\}$. For each relation type specifier $\sigma \in \Sigma$, σ^{-1} denotes "inverse σ relation". Therefore, Σ^{-1} is the set of inverse relation type specifiers in RG.
- $\overline{\Sigma}^{-1} = \{\overline{\sigma}^{-1} | \overline{\sigma} \in \overline{\Sigma}\}$. Here, $\overline{\Sigma}^{-1}$ denotes the set of non-relationship inverse relation type specifiers in RG.

The inverse non-relationship specifier $\overline{\sigma^{-1}}$ is not considered, since as shown in Appendix B it is equivalent to $\overline{\sigma}^{-1}$ and hence redundant. There is no redundancy amongst $\overline{\sigma}, \sigma^{-1}$ and $\overline{\sigma}^{-1}$, as we will see in Section 6.

RREP-0 uses simple path definition in RG. In order to specify extensions to RREP-0, three path variations in RG are defined as follows utilizing the extended relation types defined above.

Definition 4.1. **Path Variations in RG**
The definition of (simple) linked sequence of vertices in Def. 3.4 is extended to include the extended symbols in $\overline{\Sigma}$, Σ^{-1} and $\overline{\Sigma}^{-1}$, in addition to Σ. The definition of (simple) path in Def. 3.5 is extended as summarized in Table 1 to give the following three extended notions of path.

i) Simple Complementary Path (SCP) allows symbols from Σ and $\overline{\Sigma}$ respectively requiring the triple to be an edge or not an edge as indicated in the top two rows of Table 1.
ii) Simple Permissive Path (SPP) allows symbols from Σ and Σ^{-1} respectively requiring the triple to be an edge or the inverse of an edge as in the first and third rows of Table 1.
iii) Simple Complementary Permissive Path (SCPP) allows symbols from Σ, $\overline{\Sigma}$, Σ^{-1} and $\overline{\Sigma}^{-1}$ respectively requiring the triple to be an edge, not an edge, the inverse of an edge or the inverse of a "not an edge" as in the four rows of Table 1.

Based on the three path definitions introduced above, three variations of RREP problem, named as RREP-1, RREP-2, and RREP-3 are defined as follows.

Definition 4.2. **RREP-1, RREP-2, and RREP-3**
Given the definition of RREP-0 as in Def. 3.8, the definitions of RREP-1, RREP-2, and RREP-3 are similar, except for distinctions noted in Table 2.

Table 2: RREP variations

(a) RREP-0	(b) RREP-1	(c) RREP-2	(d) RREP-3
RuleSet as in Def. 3.6	*edgeLabel* ::= $\sigma \mid \overline{\sigma}$	*edgeLabel* ::= $\sigma \mid \sigma^{-1}$	*edgeLabel* ::= $\sigma \mid \overline{\sigma} \mid \sigma^{-1} \mid \overline{\sigma}^{-1}$
check Access as in Def. 3.7	simple path is replaced by SCP	simple path is replaced by SPP	simple path is replaced by SCPP

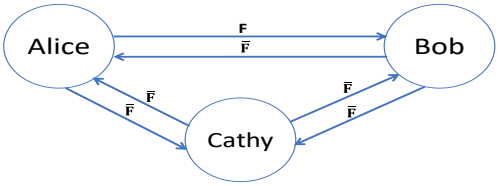

Figure 2: RG of Fig. 1 enhanced with non-relationship edges.

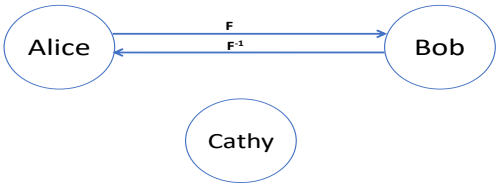

Figure 3: RG of Fig. 1 enhanced with inverse edges.

Table 2 describes the distinctions of RREP-1 to 3 in terms of comparison with RREP-0 features. Table 2 shows that RREP-1 to 3 vary from RREP-0 based on two related aspects: RuleSet and checkAccess definitions. Row 1 of Table 2 shows that, the $Rule_{op}$ grammar specified for RREP-1 to 3 vary in edgeLabel definitions only, compared to RREP-0 RuleSet definition as in Def. 3.6. Row 2 of Table 2 shows that $Rule_{op}$ of RREP-1 to 3 uses the same evaluation criteria compared to RREP-0, except simple path is changed to SCP, SPP and SCPP respectively.

4.2 Reduction of RREP Variations

The major difference between RREP-0 and the proposed variations RREP-1 to RREP-3 is that simple path definition in RREP-0 consists of given edges in RG, whereas SCP, SPP and SCPP bring additional "virtual edges" into consideration. We can reduce the enhanced path definitions of SCP, SPP and SCPP to the traditional path definition by enhancing the original RG with these virtual edges. Given the RG of Fig. 1, its enhancements with additional edges for SCP, SPP and SCPP are respectively shown in Figs. 2, 3 and 4. These enhancements are formally stated as follows.

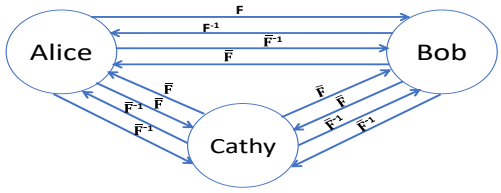

Figure 4: RG of Fig. 1 enhanced with non-relationship, inverse and non-relationship inverse edges.

Definition 4.3. **Enhancements of RG**
Given a directed labeled relationship graph RG = (V, E, Σ), let

- $\overline{E} = \{(u, v, \overline{\sigma}) \mid u \neq v \wedge (u, v, \sigma) \notin E\}$
  These are called non-relationship edges.
- $E^{-1} = \{(u, v, \sigma^{-1}) \mid u \neq v \wedge (v, u, \sigma) \in E\}$
  These are called inverse edges.
- $\overline{E}^{-1} = \{(u, v, \overline{\sigma}^{-1}) | u \neq v \wedge (v, u, \sigma) \notin E\}$
  These are called non-relationship inverse edges.

The enhanced RG, denoted RG_E, is defined as follows:

- For RREP-1: $RG_E = (V, E \cup \overline{E}, \Sigma \cup \overline{\Sigma})$
- For RREP-2: $RG_E = (V, E \cup E^{-1}, \Sigma \cup \Sigma^{-1})$
- For RREP-3: $RG_E = (V, E \cup \overline{E} \cup E^{-1} \cup \overline{E}^{-1}, \Sigma \cup \overline{\Sigma} \cup \Sigma^{-1} \cup \overline{\Sigma}^{-1})$

Note that RG_E imposes some consistency requirements such as (u, v, σ) is an edge in RG_E iff $(u, v, \overline{\sigma})$ is not an edge in RG_E.

The lemma below follows trivially from the definitions.

Lemma 4.4. *There is an SCP (respectively SPP, SCPP) p from u to v with pathLabel(p) in RG iff there is a simple path p from u to v in RG_E for RREP-1 (respectively RREP-2, RREP-3) with pathLabel(p).*

It follows that Algorithm 1 for RREP-0 with correspondingly enhanced RG can be used to solve the feasibility detection problem for RREP-1, RREP-2 and RREP-3 as well.

4.3 Limitation of RREP-0 to RREP-3

It is easy to construct examples that are beyond the scope of the variations discussed above. In the RGs of both Fig. 5 and Fig. 6, there are two simple paths from Alice to Ray with path labels "F.F" and "F.F.F". However, there is a significant difference between the two RGs. In Fig. 5 the simple paths from Alice to Ray are disjoint with respect to their edges, while this is not so for Fig. 6. Specification of disjoint paths is not possible in our rule structure variations. In the most general case any computable property of RG can be utilized in the rule structure.

A ReBAC policy language for user to user relationship is presented in [12]. Although [12] offers different rule structures for accessing user, target user as well as system administrator views, a basic comparative study between the rule set structure of our work and the ReBAC policy presented in [12] is as follows.

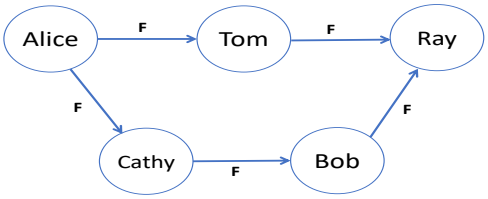

Figure 5

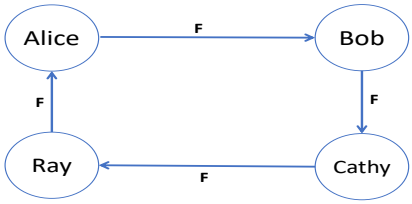

Figure 7: RREP-0 infeasibility example.

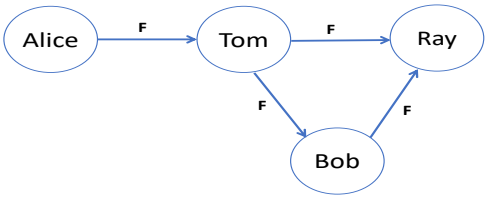

Figure 6

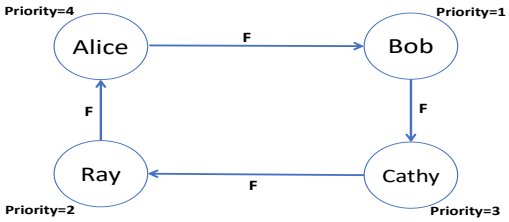

Figure 8: Adding "priority" attribute to Fig. 7.

(1) In [12], each pathLabelExpr is limited by the maximum number of edges allowed in the path, specified as hopcount. Our rule structure does not allow such numeric value on edge count in RG. Moreover, [12] offers negative pathLabelExpr, that means an entire relationship pattern that must not exist from accessing user to target user in the RG. In our work, allowing non-relationship edges accomplish the fact of traversing the graph in "not in a relationship" directions, however, its semantics is completely different.
(2) Repeating a relationship pattern unlimited (*) or 0/1 times (?) has been included in [12]. Our ReBAC policy can accomplish the similar task by repeating the rule expressions as many times as desired. Note that infinite repetitions are not possible for simple paths in a finite graph.
(3) The rule evaluation in [12] can start from a particular user as noted in the rule, but our ReBAC policy evaluation starts from any node in RG, therefore, can be referred as system policy. For both of the works, pathLabelExprs are constituted by using disjunction and conjunction operators.

From the discussions above, it can be summarized that, our rule structure lacks some features as compared to [12] such as hopcount on the pathLabelExpr, enhances a few such as allowing complementary and permissive path in RG, and has similar structure such as use of disjunction and conjunction of pathLabelExpr.

5 PROPOSED INFEASIBILITY SOLUTIONS

In this section we propose a solution to infeasibility in RREP-0 and illustrated by examples. Other possible direction of solution approaches and limitations will be discussed briefly.

5.1 Proposed Infeasibility Correction

Given a RREP-0 instance as in Def. 3.8, if no suitable RuleSet exists (i.e., Algorithm 1 returns infeasible result) we say there is an infeasibility problem. In such cases we can make a suitable RuleSet generation possible by adding new relationships as follows.

i) Select a symbol $op \notin \Sigma$.
ii) Add the path expression op as a disjunction to the generated $Rule_{op}$ by Algorithm 1 to construct $Rule_{op} \vee op$.
iii) For each $(u, v) \in failedAuthList$ add an edge (u,v,op) to E in RG.

Theorem 5.1. *The infeasibility correction solution above is correct for the modified RREP-0 problem with modified RG and* Σ.

Proof:
For each $(u, v) \in AUTH$, where Algorithm 1 fails to generate the rule, the proposed solution above adds an edge from u to v in RG with edge label op. It is trivial that a simple path of length 1 with pathLabelExpr op thereby exists in the modified RG for each such $(u, v) \in failedAuthList$, generated by Algorithm 1. Therefore, the pathLabelExpr op turns true for each such infeasible authorization tuples only.

Fig. 7 presents an RG the set of users $V = \{Alice, Bob, Cathy, Ray\}$, the set of edges E = $\{(Alice, Bob, F), (Bob, Cathy, F), (Cathy, Ray, F), (Ray, Alice, F)\}$, and the set of relation type specifiers, $\Sigma = \{F\}$. Let AUTH = $\{(Alice, Bob), (Cathy, Ray)\}$. According to the RuleSet structure given in Def. 3.6, RREP-0 fails since there exists a single simple path from Alice to Bob, where path label is F. However, "F" is also true for (Alice, Bob), (Bob, Cathy), (Cathy, Ray), and (Ray, Alice). The same scenario occurs while finding rule for (Cathy, Ray).

Therefore, the given AUTH is concluded as infeasible by Algorithm 1, and failedAuthList contains both (Alice, Bob) and (Cathy, Ray). According to the solution above, two additional edges (Alice, Bob, op) and (Cathy, Ray, op) are added to E, and Σ is updated to $\{F, op\}$. The generated $Rule_{op}$ is op.

5.2 Alternate Infeasibility Correction

An alternate approach to infeasibility correction is to add an attribute named "priority" to each vertex in the RG, illustrated by example as follows. Consider the solution provided in Fig. 8, given the prior infeasibility example: AUTH = $\{(Alice, Bob), (Cathy, Ray)\}$ and RG is as shown in Fig. 7. Each user vertex in the given RG in Fig. 7 has been assigned a positive integer priority value, and the ordered sequence of vertex priority values associated with the path p from vertex a to b in RG is the same order followed by the vertices through the path p. For example, ordered sequence of vertex priority values associated with the path from Alice to Ray in Fig. 8 is $\langle 4, 1, 3, 2\rangle$. The $Rule_{op}$ given in Def. 3.6 is modified in order to accommodate the use of priority value as follows:

$Rule_{op}$ ::= $Rule_{op} \vee Rule_{op} \mid pathRuleExpr$
$pathRuleExpr$ ::= $pathRuleExpr \wedge pathRuleExpr \mid$
$(pathLabelExpr, priorityOrder)$
$priorityOrder$ ::=> | < | ϕ
$pathLabelExpr$ ::= $pathLabelExpr.pathLabelExpr \mid edgeLabel$
$edgeLabel$ ::= σ, $\sigma \in \Sigma$
where >,<, and ϕ represent increasing, decreasing and don't care orders, respectively, and pathRuleExpr consists of conjunction of (pathLabelExpr, priorityOrder) pairs.

The evaluation procedure of $checkAccess_{ReBAC}$(a:V, b:V) in a ReBAC system with the specified $Rule_{op}$ is as follows:
(i) for each (pathLabelExpr, Order) pair in $Rule_{op}$ substitute True if there exists a simple path p from a to b in RG with path label pathLabelExpr where the ordered sequence of vertex priority values associated with path p follows the priorityOrder order, otherwise substitute False, (ii) evaluate the resulting boolean expression.

Let's recall the AUTH = $\{(Alice, Bob), (Cathy, Ray)\}$ noted earlier for Fig. 7. According to the proposed $Rule_{op}$ structure, the generated $Rule_{op} = (F, <)$ solves the infeasibility because the simple path labeled F from Alice to Bob follows the decreasing order as $4 > 2$. The same case occurs for (Cathy, Ray) since $3 > 2$, whereas (Bob, Cathy) and (Ray, Alice) do not.

5.3 Limitations of Current Infeasibility Solution

The infeasibility solution provided in Section 5 adds only a single pathLabelExpr "op" to the $Rule_{op}$, regardless of the number of infeasible tuples in the AUTH, adding $|AUTH|$ number of additional edges in RG in the worst case. To clarify more with an example, given the RG in Fig. 9 and the set of authorization tuples $\{(Alice, Ray), (Alice, Bob), (Alice, Cathy)\}$, this solution adds three edges with label op originating from Alice to Bob, Ray, and Cathy, respectively. Therefore, the $Rule_{op}$ is "op". However, a solution fewer added edges can be obtained for the given AUTH by

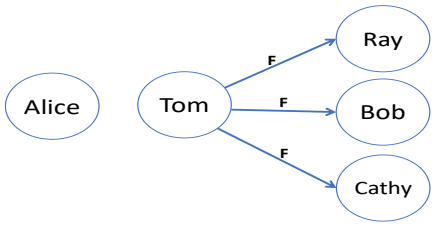

Figure 9

adding a single edge from Alice to Tom labeled as "op". It is clearly evident that the edge from Alice to Tom creates simple paths from Alice to Ray, Bob, and Cathy. Therefore, the possible $Rule_{op}$ for the given AUTH is "op.F". This demonstrates the trade-off between minimum size of rule and adding minimum number edges in RG to correct infeasibility. The solution of Subsection 5.1 keeps the given RG unchanged while adding new relationship edges to RG. An alternative approach could be to remove some edges from the given RG.

6 CASE STUDIES

In this section we present case studies to show the relative power of the rule structures of RREP variations defined in this paper. We also discuss the need for rule optimization.

Consider the RG shown in Fig. 10(a), along with its inverse, non-relationship and non-relationship inverse edges shown in Figs. 10(b), 11(a) and 11(b). For different values of AUTH we get different feasibility results as follows, where we understand that Algorithm 1 will be run with correspondingly enhanced RGs (i.e., Fig. 10(a) for RREP-0, union of Figs. 10(a) and 11(a) for RREP-1, union of Figs. 10(a) and 10(b) for RREP-2, and union of Figs. 10(a), 10(b), 11(a) and 11(b) for RREP-3).

(1) Let AUTH = {(Ray,Cathy), (Bob,Cathy)}. Then Algorithm 1 will return success for RREP-0, RREP-1, RREP-2 and RREP-3. Note that feasibility of RREP-0 always implies feasibility of RREP-1, RREP-2 and RREP-3 since the simple path of RREP-0 is included in the enhanced path definitions of the latter. The rule returned for RREP-0 and RREP-2 is F∨F which is logically equivalent to F. The rules generated for RREP-1 and RREP-3 are more complex due to the increased number of paths in the enhanced RGs.
(2) Let AUTH = {(Cathy,Ray), (Cathy,Bob)}. For RREP-0 and RREP-1 Algorithm 1 will return failure. For RREP-2 it will return $F^{-1} \vee F^{-1}$. The formula for RREP-3 is more complex.
(3) Let, AUTH = {(Alice, Bob), (Alice, Cathy), (Alice, Ray), (Bob, Alice), (Bob, Ray), (Cathy, Alice), (Cathy, Bob), (Cathy,Ray), (Ray, Alice), (Ray,Bob)}. For RREP-0 and RREP-2 Algorithm 1 will return failure. For RREP-1 and RREP-3 it will return success with complex formulae due to the multiplicity of paths in the enhanced RGs.
(4) Let's consider, AUTH = {(Ray,Cathy), (Bob,Cathy), (Cathy,Ray), (Cathy,Bob), (Alice, Cathy)}. For RREP-0, RREP-1 and RREP-2

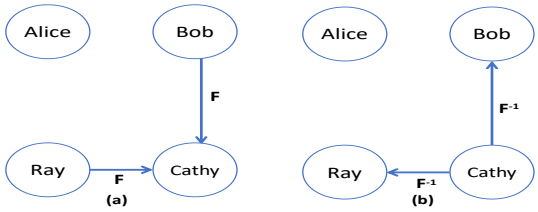

Figure 10: (a) Given RG
(b) Inverse edges for RG of Fig. 10(a)

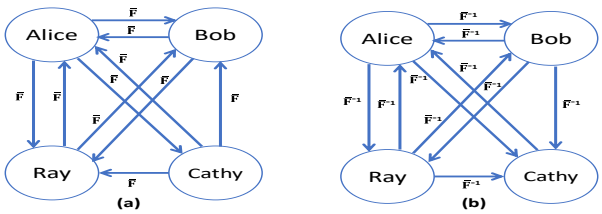

Figure 11: (a) Non-relationship edges for RG of Fig. 10(a)
(b) Non-relationship inverse edges for RG of Fig. 10(a)

Algorithm 1 will return failure. For RREP-3 it will return success with a complex formula which would logically reduce to $\overline{F}^{-1}.\overline{F}^{-1}.F \vee F^{-1}$.

These examples establish that the rule structure of RREP-3 is strictly more expressive than RREP-0, RREP-1 and RREP-2. Note that RREP-0 is the weakest as argued above. RREP-1 and RREP-2 are incomparable.

The generated rule may contain unnecessary path labels in conjunctive terms if all possible path labels are being used. Therefore, a few simple rule optimization techniques are used in the implementation. As stated in Algorithm 1, for any tuple (a,b) in AUTH, all possible path labels from a to b are AND'ed to form the conjunctive term after determining the feasibility. Instead of using all possible path labels, the smallest possible subset of those is used to form the conjunctive term such that it does not evaluate to true for any unauthorized tuple. For example, given the RG in Fig.10(a), and an EAS where V is identical and set of authorization relations AUTH = $\{(Alice, Ray), (Alice, Bob)\}$, the $Rule_{op}$ computed using RREP-3 by Algorithm 1 comprises a conjunction of 24 terms as follows:
$\overline{F}.F^{-1} \wedge \overline{F}^{-1} \wedge \overline{F}.\overline{F} \wedge \overline{F}^{-1}.\overline{F} \wedge \overline{F}.\overline{F}^{-1} \wedge \overline{F}^{-1}.\overline{F}^{-1}.F^{-1} \wedge \overline{F}^{-1}.\overline{F}.\overline{F} \wedge \overline{F}^{-1}.F^{-1} \wedge \overline{F}^{-1}.F^{-1}.\overline{F} \wedge \overline{F}.F^{-1}.\overline{F}^{-1} \wedge \overline{F}.\overline{F}^{-1}.\overline{F} \wedge \overline{F}^{-1}.F^{-1}.\overline{F}^{-1} \wedge \overline{F}.\overline{F}.\overline{F} \wedge \overline{F}.\overline{F}.\overline{F}^{-1} \wedge \overline{F}^{-1}.\overline{F}.F^{-1} \wedge \overline{F}^{-1}.\overline{F}^{-1}.\overline{F} \wedge \overline{F}.F^{-1}.\overline{F} \wedge \overline{F}.F.\overline{F} \wedge \overline{F} \wedge \overline{F}^{-1}.F.F^{-1} \wedge \overline{F}.\overline{F}^{-1}.F^{-1} \wedge \overline{F}^{-1}.F.\overline{F} \wedge \overline{F}^{-1}.\overline{F}^{-1} \wedge \overline{F}.F.F^{-1}$
Both tuples (Alice, Bob) and (Alice, Ray) in AUTH would generate this conjunction since they have the same set of path labels. After applying the specified smallest possible subset of path labels in a conjunctive term technique, the specified $Rule_{op}$ turns into significantly smaller rule, given by $\overline{F}.F^{-1}$. Another way of rule minimization is: after completion of rule generation, a conjunctive term in the generated rule, say c1, removes all conjunctive terms c2 in the rule if all path labels in c1 are included in c2. The Java implementation of feasibility detection along with the described rule minimization techniques can be found here (https://github.com/shuvrac7/Formal-Analysis-of-ReBAC-Policy-Mining-Feasibility.git).

7 FUTURE RESEARCH

The RREP has been introduced for the first time in this paper. Here, a few directions for future enhancement will be addressed briefly.

(1) The proposed feasibility detection Algorithm 1 produces overall exponential asymptotic complexity, considering the rule minimization applied in the implementation. Given a directed RG = (V, E, Σ) and an EAS as defined in Def. 3.3 and 3.2, respectively, Algorithm 1 needs to compute all possible simple paths for all $(u, v) \in V \times V, u \neq v$. In the worst case, the total possible path labels in RG can be estimated as follows:
   - The length of simple path from for any $(u, v) \in V \times V, u \neq v$ in RG ranges from 1 to $(|V| - 1)$, where V is the set of users in the system.
   - Given the finite set of relation type specifiers Σ, for all possible simple paths of length 1, the number of possible path labels in RG is $|\Sigma|$. By using the same concept, for all possible simple paths of length 2, the number of possible path labels in RG is $|\Sigma| \times |\Sigma|$. Therefore, for all possible simple paths of length $|V| - 1$, the total number of possible path labels in RG is $|\Sigma|^{|V|-1}$.
   - Therefore, given a graph RG, all possible path labels considering all possible simple paths in RG is: $\Sigma + \Sigma^2 + \Sigma^3 + \ldots + \Sigma^{|V|-1}$.

   A significant form of improvement in the current approach would be to reduce computation complexity.
(2) In this paper, the context to ReBAC rule set structure is restricted to RREP-0 to RREP-3. The feasibility detection algorithm depends on the intended rule set structure, as shown in the paper. Adding new features to the existing rule structures, how much modification does the feasibility detection algorithm need when some new features are added to rule structure, which features make the rule structures more general in practical scenario, etc., can be interesting research problems.
(3) An advanced direction could be: given any ReBAC rule structure, can the process of the feasibility detection be completely automated?
(4) Here, some solution approaches to infeasibility have been discussed. More efficient solution approaches can be proposed, which remained as open research problem.
(5) In this study, no loops are allowed in RG. What could be done in order to manage such a RG with self loops, as well as path including cycles? Our current algorithm would work with path including cycles where length limit is given if slightly modified. We have not investigated cycles without length limits.

(6) Only exact solutions have been considered so far. Is having inexact solution reduce computation complexity? In that case, what are the factors needed to be considered?
(7) Extend the feasibility problem definition as well as infeasiblity solutions beyond user to user context.

ACKNOWLEDGEMENT

This work is partially supported by NSF CREST Grant HRD-1736209.

REFERENCES

[1] Tahmina Ahmed, Farhan Patwa, and Ravi Sandhu. 2016. Object-to-Object Relationship-Based Access Control: Model and Multi-Cloud Demonstration (Invited Paper). In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*. 297–304.
[2] Tahmina Ahmed, Ravi Sandhu, and Jaehong Park. 2017. Classifying and Comparing Attribute-Based and Relationship-Based Access Control. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (Scottsdale, Arizona, USA) *(CODASPY '17)*. Association for Computing Machinery, New York, NY, USA, 59–70.
[3] Glenn Bruns, Philip W.L. Fong, Ida Siahaan, and Michael Huth. 2012. Relationship-Based Access Control: Its Expression and Enforcement through Hybrid Logic *(CODASPY '12)*. Association for Computing Machinery, New York, NY, USA, 117–124.
[4] Thang Bui and Scott D. Stoller. 2020. A Decision Tree Learning Approach for Mining Relationship-Based Access Control Policies. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies* (Barcelona, Spain) *(SACMAT '20)*. Association for Computing Machinery, New York, NY, USA, 167–178.
[5] Thang Bui, Scott D. Stoller, and Hieu Le. 2019. Efficient and Extensible Policy Mining for Relationship-Based Access Control. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies* (Toronto ON, Canada) *(SACMAT '19)*. Association for Computing Machinery, New York, NY, USA, 161–172.
[6] Thang Bui, Scott D. Stoller, and Jiajie Li. 2017. Mining Relationship-Based Access Control Policies. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies* (Indianapolis, Indiana, USA) *(SACMAT '17 Abstracts)*. Association for Computing Machinery, New York, NY, USA, 239–246.
[7] Thang Bui, Scott D. Stoller, and Jiajie Li. 2019. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers Security* 80 (2019), 317 – 333.
[8] Thang Bui, Scott D. Stoller, and Jiajie Li. 2019. Mining Relationship-Based Access Control Policies from Incomplete and Noisy Data. In *Foundations and Practice of Security*, Nur Zincir-Heywood, Guillaume Bonfante, Mourad Debbabi, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 267–284.
[9] Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. 2019. On the Feasibility of Attribute-Based Access Control Policy Mining. In *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*. 245–252.
[10] Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. 2020. On the Feasibility of RBAC to ABAC Policy Mining: A Formal Analysis. In *Secure Knowledge Management In Artificial Intelligence Era*. Springer Singapore, Singapore, 147–163.
[11] Y. Cheng, J. Park, and R. Sandhu. 2012. Relationship-Based Access Control for Online Social Networks: Beyond User-to-User Relationships. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. 646–655.
[12] Yuan Cheng, Jaehong Park, and Ravi Sandhu. 2012. A User-to-User Relationship-Based Access Control Model for Online Social Networks. In *Data and Applications Security and Privacy XXVI"*. Springer Berlin Heidelberg, Berlin, Heidelberg, 8–24.
[13] Yuan Cheng, Jaehong Park, and Ravi Sandhu. 2014. Attribute-Aware Relationship-Based Access Control for Online Social Networks. In *Data and Applications Security and Privacy XXVIII*, Vijay Atluri and Günther Pernul (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 292–306.
[14] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. 2001. Proposed NIST standard for role-based access control. *ACM TISSEC* 4, 3 (2001), 224–274.
[15] Philip W.L. Fong. 2011. Relationship-Based Access Control: Protection Model and Policy Language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy* (San Antonio, TX, USA) *(CODASPY '11)*. Association for Computing Machinery, New York, NY, USA, 191–202.
[16] Philip W.L. Fong and Ida Siahaan. 2011. Relationship-Based Access Control Policies and Their Policy Languages. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies* (Innsbruck, Austria) *(SACMAT '11)*. Association for Computing Machinery, New York, NY, USA, 51–60.
[17] Vincent C Hu et al. 2014. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication* 800 (2014), 162.
[18] Padmavathi Iyer and Amirreza Masoumzadeh. 2019. Generalized Mining of Relationship-Based Access Control Policies in Evolving Systems. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies* (Toronto ON, Canada) *(SACMAT '19)*. Association for Computing Machinery, New York, NY, USA, 135–140.
[19] Amirreza Masoumzadeh. 2018. Security Analysis of Relationship-Based Access Control Policies *(CODASPY '18)*. Association for Computing Machinery, New York, NY, USA, 186–195.

A PATH GENERATION ALGORITHM

Algorithm 2 FindAllSimplePath

Input: Vertex source, vertex dest, $RG = (V, E, \Sigma)$
Output: Set of all simple paths from source to dest in RG

1: //visitVertex is a map where visitVertex[$u \in V$] = white means "not visited", visitVertex[$u \in V$] = grey means "visited but not finished yet"
2: //visitEdge is a map where visitEdge[$e \in E$]=white means "not visited", visitEdge[$e \in E$]=grey means "visited but not finished yet"
3: **for** $u \in V$ **do**
4: visitVertex[u]:=white
5: **for** $e \in E$ **do**
6: visitEdge[e]:=white
7: PS := $\emptyset$
8: Modified-DFS-Visit(source, dest, RG, PS, $\langle\rangle$) //assuming visitVertex and visitEdge are globally defined
9: **return** PS

Algorithm 3 Modified-DFS-Visit

Input: vertex src, vertex dest, $RG(V, E, \Sigma)$, PS, tempPath
Output: Path generation from src to dest in RG

1: **if** src == dest **then**
2: $PS\cup$:= $tempPath$
3: **return**
4: visitVertex[src]:=grey
5: **for** each edge $e \in E$, where e=(x,y,σ) and x=src **do**
6: **if** visitEdge[e] == white and visitVertex[y] == white **then**
7: Modified-DFS-Vist(y,dest,RG,PS,appendSeq(tempPath,e)) //appendSeq() is a trivial function which appends edge e to the path sequence, tempPath and returns the new ordered path sequence
8: visitVertex[src]:=white
9: **for** each edge $e \in E$, where e=(x,y,σ) and x=src **do**
10: visitEdge[e] := white
11: **return**

For completeness, the algorithm used for all possible path generation in Algorithm 1, called FindAllSimplePath has been included. Given a RG and a vertex pair (source,dest), algorithm FindAllSimplePath returns the set of all possible simple paths from source to dest in RG. It is basically a modified form of core Depth-First-Search from vertex source to vertex dest in RG.

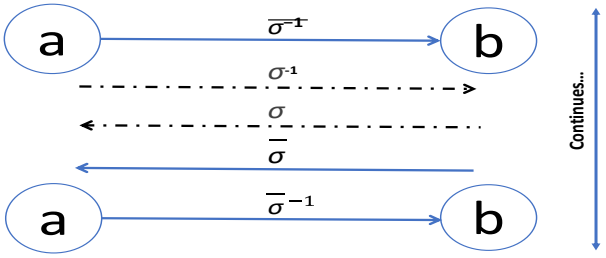

Figure 12: Given a $\sigma \in \Sigma$, $\overline{\sigma}^{-1} \equiv \overline{\sigma^{-1}}$ where solid and dotted lines represent the edges that must and must not exist, respectively.

B INVERSE NON-RELATIONSHIP EDGE

Given a $\sigma \in \Sigma$, $\overline{\sigma^{-1}}$ is called the inverse non-relationship of σ. We show that $\overline{\sigma}^{-1} \equiv \overline{\sigma^{-1}}$. Fig. 12 shows a sequence of equivalences going from top to bottom, or vice versa, which establish this. Relationships that cannot exist are shown in dotted lines while relationships that must exist are shown in solid lines. From top to bottom, a relationship $\overline{\sigma^{-1}}$ from a to b precludes a relationship of σ^{-1} from a to b, which in turn precludes a relationship of σ from b to a. Thereby there is a non-relationship $\overline{\sigma}$ from b to a, and finally its inverse $\overline{\sigma}^{-1}$ from a to b. The argument in reverse holds from bottom to top.