

# Constraint-Enhanced Role Engineering via Answer Set Programming

Jinwei Hu<sup>1</sup> Khaled M. Khan<sup>1</sup> Yun Bai<sup>2</sup> Yan Zhang<sup>2</sup>

<sup>1</sup>Department of Computer Science and Engineering, Qatar University, Doha 2713, Qatar

<sup>2</sup>School of Computing and Mathematics, University of Western Sydney, Sydney 1797, Australia  
{jinwei, k.khan}@qu.edu.qa {ybai, yan}@scm.uws.edu.au

## ABSTRACT

*Role engineering (RE)* aims to develop and maintain appropriate *role-based access control (RBAC)* configurations. However, RE with constraints in place is not well-studied. Constraints usually describe organizations' security and business requirements. An inconsistency between configurations and constraints compromises security and availability, as it may authorize otherwise forbidden access and deprive users of due privileges. In this paper, we apply *answer set programming (ASP)* to discover RBAC configurations that comply with constraints and meet various optimization objectives. We first formulate the need of supporting constraints as a problem independent of and complementary to existing RE problems. We then present a flexible framework for translating the proposed problem to ASP programs. In this way, the problem can be addressed via ASP solvers. Finally, we demonstrate the effectiveness and efficiency of our approach through experimental results.

## Categories and Subject Descriptors

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

## General Terms

Security, Management

## Keywords

RBAC, Constraint, Role Engineering, Answer Set Programming

## 1. INTRODUCTION

It proves challenging to build a secure and manageable access control system. By introducing “roles” as an intermediate level between users and permissions, *role-based access control (RBAC)* mitigates the difficulty. There, users are assigned to roles; roles in turn are associated with permissions [9, 24]. RBAC has been regarded as an effective approach to the access control problem of medium and large size organizations [23].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ASIACCS '12, May 2–4, 2012, Seoul, Korea.

Copyright 2012 ACM 978-1-4503-1303-2/12/05 ...\$10.00.

However, RBAC systems are costly to develop and maintain. Putting aside other problems, the creation of an RBAC configuration, which consists of identifying a set of appropriate roles and assigning users and permissions to roles, is not easy [14, 21]. As such, various *role engineering (RE)* approaches have been proposed to find “good” configurations, e.g., [5, 8, 21, 29]. Provided with certain information about the organization in question, an RE tool produces a configuration. Nonetheless, one main limitation of existing approaches is that, the discovered configuration does not necessarily meet constraints.

Organizations' security officers often specify constraints to enforce high-level security objectives. One typical example is a *separation-of-duty* policy, which, for example, prohibits a user from performing two mutually exclusive permissions (e.g., prepare a check and authorize a check).<sup>1</sup> Moreover, constraints could also be used to model business requirements. For example, an organization may place a constraint that a role *emergencyManager* be assigned to at least two users, so that at least two managers are available in an emergency. As an inherent part of the classic RBAC models [24] and the standards [2, 9], constraints play an important part in capturing organizations' requirements [1, 7, 17, 27].

A configuration inconsistent with constraints, if deployed, may undermine organizations' interests, by means of authorizing otherwise forbidden accesses or depriving users of due privileges. This poses continual challenges. When being deployed, the configuration ought to comply with constraints. Afterwards, this compliance should persist. However, organizations' security and business requirements are subject to constant changes. One misleading assumption is that people can establish their requirements once for all; it obstructs the integration of research efforts with practices [26]. Hence, a deployed configuration needs updating from time to time to reflect organizations' requirements [8, 21].

Unfortunately, so far little effort has been devoted to constraint support when organizations are adopting RBAC configurations or undergoing evolution. On the one hand, the state of the art of RE barely takes constraints into consideration; as a result, it is necessary to adapt discovered configurations for constraints. As pointed out in [21, 29], a post-processing like this is often unavoidable and, if done manually, is a large bottleneck. Hence, an effective and efficient support of constraints for RE is worth investigation. On the other hand, the research literature on constraints in access control has focused mainly on checking if there exists a configuration consistent with constraints (e.g., [27]) or if a change to a configuration is safe with respect to constraints (e.g., [7, 17]). They provide inadequate help. Although some approaches such as [27] generate configurations in presence of constraints, they are not tailored

<sup>1</sup>To increase the cost of a fraud, it is often required that no user can both prepare and authorize a check.

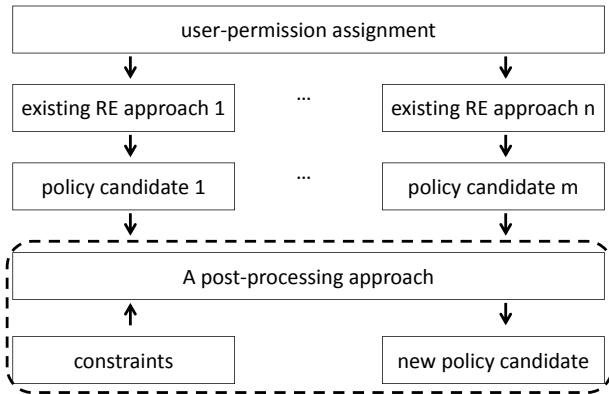


Figure 1: The idea of our approach.

for RE. For example, RE aims to discover “good” configurations; however, the quality of configurations is not a concern there.

We treat constraint support as a modular post-processing of configurations produced by RE tools, as shown in Figure 1. The reasons are many-fold. First, different situations call for different types of RE tools, depending on available information. If integrating constraint support with existing functions, one may have to augment each type of RE tools in order to deal with different situations. In contrast, modular constraint support as a post-processing provides a uniform approach. Second, this design also enables us to combine advantages of existing RE algorithms. In view of the diversity of RE techniques, different tools may discover multiple configurations for a given input; each of them bears properties an RE tool specializes in. It is likely to (partially) retain the properties if we take these configurations into consideration when seeking a configuration consistent with constraints. Finally, as mentioned above, constraints are dynamic; a separate module for constraints is more flexible, manageable, and extensible.

With this choice of design, we are provided with a set of constraints, denoted as  $C$ , which is specified by organizations’ security officers, and a set of configurations, denoted as  $\Gamma$ , which contains the running configuration and/or configurations discovered by RE tools. The problem of supporting constraints for RE, denoted as *Role Constraint Problem (RCP)*, boils down to finding a configuration  $\gamma_0$  that is consistent with  $C$  and close to  $\Gamma$  as a whole. The requirement of closeness attempts to limit the deviation of  $\gamma_0$  from  $\Gamma$ , as configurations in  $\Gamma$  may fit the organization in question in one way or another. Also, in this way,  $\gamma_0$  is expected to be of high quality among those consistent with  $C$ .

To formulate *RCP*, we first define a simple language for specifying constraints. We then take a declarative approach based on *Answer Set Programming (ASP)* [3]. In brief, we encode each *RCP* instance as an ASP program, compute an answer set of the resulting program, and extract a solution. Although the applications of ASP follow the general “encode-compute-extract” way [3], they differ in problem representations in ASP. In our case, there are two principal factors which render the proposed problem and approach non-trivial. First, the constraint language should be amendable to ASP interpretation, but also expressive enough to capture common idioms of security and business requirements. There is a trade-off to make. Second, an ASP encoding should capture requirements on solutions’ quality, including both the closeness to  $\Gamma$  and other practical needs.

This ASP-based approach provides a declarative representation of the problem and allows us to adopt mature ASP solvers that have been proved to work well in practice. Moreover, its rich modeling

language eases the understanding and explanation of the problem and the approach. The main contributions of this work are:

- We propose and formally define the problem of supporting constraints in role engineering. *RCP* is formulated as a problem independent of and complementary to existing role engineering problems. Among others, this enables us to add the support of constraints modularly. We also study the computational complexity of *RCP*.
- We present a framework for encoding *RCP* in ASP. *RCP* can thus be addressed via existing ASP solvers. The framework is flexible in that various *RCP* variants can be encoded in a straightforward manner.
- We undertake experiments to validate the practicality of the framework.

The rest of the paper is organized as follows. In Section 2, we review the notions of RBAC. In Section 3, we propose the constraint language, define *RCP*, and study its computational complexity; two examples of *RCP* are also given. In Section 4, we present a transformation of *RCP* into ASP. We show the experiment results in Section 5. In Section 6, we discuss the assumptions and *RCP* variants. Finally, related work is discussed in Section 7, followed by conclusions in Section 8.

## 2. ROLE-BASED ACCESS CONTROL

An RBAC configuration is a tuple  $\gamma = \langle \mathcal{U}, \mathcal{R}, \mathcal{P}, UA, PA \rangle$ , where  $\mathcal{U}$  is a set of users,  $\mathcal{R}$  is a set of roles, and  $\mathcal{P}$  is a set of permissions,  $UA \subseteq \mathcal{R} \times \mathcal{U}$  is the user-role relation, assigning users to roles, and  $PA \subseteq \mathcal{R} \times \mathcal{P}$  is the role-permission relation, associating permissions with roles. Based on  $UA$  and  $PA$ , one derives  $\gamma$ ’s user-permission relation  $UPA = \{(u, p) \mid \exists r \in \mathcal{R} : (r, u) \in UA \wedge (r, p) \in PA\}$ . We usually denote an RBAC configuration as  $\gamma$ , possibly with subscripts; unless otherwise stated, a configuration  $\gamma_i$  denotes a tuple  $\langle \mathcal{U}_i, \mathcal{R}_i, \mathcal{P}_i, UA_i, PA_i \rangle$ .

**Example 1.** Consider access control of grades in a university: four representative users *alice*, *bob*, *carl*, and *dave*,<sup>2</sup> may be assigned to roles *stu* (student), *ta* (teaching assistant), *fac* (faculty), and *dean*; roles could be assigned to four permissions: *assign*, *view*, *receive*, and *change* grades (denoted as *asg*, *view*, *rec*, and *chg*, respectively). Figure 2 shows an example configuration of this scenario, denoted as  $\gamma_{uni}$ . For example, *alice* is assigned to the roles *stu* and *ta*, and the role *stu* is assigned to *rec*. Therefore, *alice* is able to receive grades with the role *stu*.

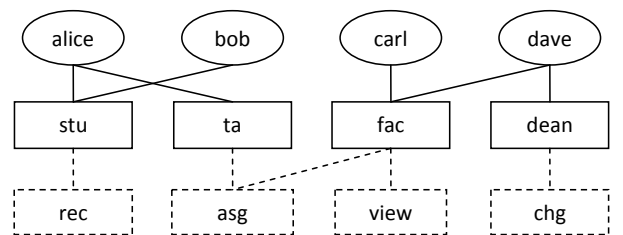


Figure 2: An example RBAC configuration  $\gamma_{uni}$ . Solid lines represent the user-role relation, and dashed lines represent the role-permission relation.

<sup>2</sup>Since ASP solvers usually take capitalized words as variables and others as constants, we write names in lowercase.

Table 1: Functions of RBAC configurations.

$\text{user}_\gamma[x] = \begin{cases} \{u \in \mathcal{U} \mid (x, u) \in UA\} & \text{if } x \in \mathcal{R} \\ \bigcup_{r \in \text{role}_\gamma[x]} \text{user}_\gamma[r] & \text{if } x \in \mathcal{P} \end{cases}$
$\text{role}_\gamma[x] = \begin{cases} \{r \in \mathcal{R} \mid (r, x) \in UA\} & \text{if } x \in \mathcal{U} \\ \{r \in \mathcal{P} \mid (r, x) \in PA\} & \text{if } x \in \mathcal{P} \end{cases}$
$\text{perm}_\gamma[x] = \begin{cases} \bigcup_{r \in \text{role}_\gamma[x]} \text{perm}_\gamma[r] & \text{if } x \in \mathcal{U} \\ \{p \in \mathcal{P} \mid (x, p) \in PA\} & \text{if } x \in \mathcal{R} \end{cases}$

Given a configuration  $\gamma$ , we define three functions as shown in Table 1. Function  $\text{user}_\gamma[\cdot] : \mathcal{R} \cup \mathcal{P} \mapsto 2^{\mathcal{U}}$  gives the set of users that a role or a permission is associated with in  $\gamma$ . Take  $\gamma_{\text{uni}}$  for instance; we have  $\text{user}_{\gamma_{\text{uni}}}[\text{fac}] = \{\text{carl}, \text{dave}\}$  and  $\text{user}_{\gamma_{\text{uni}}}[\text{chg}] = \{\text{dave}\}$ . Function  $\text{role}_\gamma[\cdot] : \mathcal{U} \cup \mathcal{P} \mapsto 2^{\mathcal{R}}$  returns the set of roles that are assigned to a user or a permission in  $\gamma$ . For instance  $\text{role}_{\gamma_{\text{uni}}}[\text{bob}] = \{\text{stu}\}$  and  $\text{role}_{\gamma_{\text{uni}}}[\text{asg}] = \{\text{ta}, \text{fac}\}$ . Finally, function  $\text{perm}_\gamma[\cdot] : \mathcal{U} \cup \mathcal{R} \mapsto 2^{\mathcal{P}}$  maps the users and roles to their permissions in  $\gamma$ . For example,  $\text{perm}_{\gamma_{\text{uni}}}[\text{dave}] = \{\text{asg}, \text{view}, \text{chg}\}$  and  $\text{perm}_{\gamma_{\text{uni}}}[\text{dean}] = \{\text{chg}\}$ .

### 3. PROBLEM DEFINITION

#### 3.1 Constraint Specification

A configuration  $\gamma$  decides on which permissions a user may perform. The set  $\text{perm}_\gamma[u]$  contains all the permissions available to  $u$ , whereas  $\text{user}_\gamma[p]$  returns the set of users capable of executing  $p$ . Also, in a sense, a role is identified by the set of users and/or the set of permissions that are associated with it. Hence, a configuration is mostly characterized by the set relationships among users, roles, and permissions. Accordingly, constraints should concern these set relationships.

**Definition 2.** A *set expression* is obtained by applying the two rules finitely many times: (1) Each of  $\text{user}[x]$ ,  $\text{role}[x]$ , and  $\text{perm}[x]$  is a set expression, where  $x \in \mathcal{U} \cup \mathcal{R} \cup \mathcal{P}$ . (2) If  $s_1$  and  $s_2$  are set expressions, then so are  $s_1 \cap s_2$  and  $s_1 \cup s_2$ .

Set expressions are evaluated against configurations. The evaluation of  $s$  against a configuration  $\gamma$ , denoted as  $s/\gamma$ , is obtained by replacing every appearance of  $\text{user}$  with  $\text{user}_\gamma$ ,  $\text{role}$  with  $\text{role}_\gamma$ , and  $\text{perm}$  with  $\text{perm}_\gamma$ . Formally, it is defined inductively:  $(s_1 \cap s_2)/\gamma = (s_1/\gamma) \cap (s_2/\gamma)$ ,  $(s_1 \cup s_2)/\gamma = (s_1/\gamma) \cup (s_2/\gamma)$ ,  $\text{user}[x]/\gamma = \text{user}_\gamma[x]$ ,  $\text{role}[x]/\gamma = \text{role}_\gamma[x]$ , and  $\text{perm}[x]/\gamma = \text{perm}_\gamma[x]$ . For example,  $\text{user}[p_1] \cap \text{user}[p_2]$  is evaluated as  $\text{user}_\gamma[p_1] \cap \text{user}_\gamma[p_2]$ . Notice that  $\text{user}[u]$  evaluates to  $\{u\}$  if  $u \in \mathcal{U}$ , regardless of  $\gamma$ . For simplicity we write set  $\{u_1, \dots, u_n\} \subseteq \mathcal{U}$  as an abbreviation for  $\bigcup_{i=1}^n \text{user}[u_i]$ ; similar abbreviations apply to  $\text{role}[x]$  and  $\text{perm}[x]$ .

**Definition 3.** A *structural constraint* has the form  $s_1 \subseteq s_2$ , where  $s_1$  and  $s_2$  are set expressions. A configuration  $\gamma$  *satisfies* a constraint  $s_1 \subseteq s_2$  if  $s_1/\gamma \subseteq s_2/\gamma$ .

Structural constraints are inspired by the property language in [18]. They state subset-relationships between sets. Since the sets are derived from a configuration's relations, constraints actually put requirements on the structure of those relations. For example,  $\text{user}[r_1] \subseteq \text{user}[p_1] \cap \text{user}[p_2]$  stipulates that role  $r_1$ 's user set be a subset of the intersection of permission  $p_1$ 's user set and  $p_2$ 's user set; in terms of access control, it says that any user in role  $r_1$  should possess the permissions  $p_1$  and  $p_2$ .

**Definition 4.** A *quantity constraint* has the form  $|s|\theta n$ , where  $\theta$  is an operator from the set  $\{=, \neq, \leq, \geq\}$  and  $n$  is an integer. A

configuration  $\gamma$  *satisfies* a constraint  $|s|\theta n$  if  $|s/\gamma|\theta n$ , where where  $|\cdot|$  denotes the size of a set or a relation.

Quantity constraints concern the size of a set. For example,  $|\text{user}[p_1] \cap \text{user}[p_2]| \geq 1$  says that at least one user is able to exercise both  $p_1$  and  $p_2$ . Suppose, for instance, that  $p_1$  and  $p_2$  are the two steps of an emergency procedure; this constraint ensures that proper measures could be taken in an emergency by at least one user.

Given a set  $C$  of constraints, we say  $\gamma$  *satisfies*  $C$  if it satisfies all constraints in  $C$ .

**Example 5.** We give more examples to show how this language can express a wide range of constraint idioms in the context of Example 1.

*con1* Students can receive grades:  $\{\text{rec}\} \subseteq \text{perm}[\text{stu}]$ . This exemplifies a constraint on roles' permission range.

*con2* If allowed to change grades, a role can also view grades:  $\text{role}[\text{chg}] \subseteq \text{role}[\text{view}]$ . Anyone with such a role can view grades before s/he changes them. Constraints of this type are often meant to guarantee roles' job functions.

*con3* The capabilities to change grades are confined to carl and dave:  $\text{user}[\text{chg}] \subseteq \{\text{carl}, \text{dave}\}$ . This is an example of the *assignment range policy* [19]. Its general form,  $\text{user}[p_1] \cup \dots \cup \text{user}[p_m] \subseteq \{u_1, \dots, u_n\}$ , requires that the permissions  $\{p_1, \dots, p_m\}$  be possessed only by users in  $\{u_1, \dots, u_n\}$ . A similar constraint  $\text{user}[p_1] \cap \dots \cap \text{user}[p_m] \subseteq \{u_1, \dots, u_n\}$  enforces *bounded safety* [18], i.e., the users that can have all permissions  $\{p_1, \dots, p_m\}$  are limited to users  $\{u_1, \dots, u_n\}$ .

*con4* No students can assign grades:  $|\text{user}[\text{stu}] \cap \text{user}[\text{asg}]| = 0$ . Note that the constraint  $|\text{perm}[\text{stu}] \cap \{\text{asg}\}| = 0$  does not capture this requirement.

*con4* is an instance of *safety policy* [18], as it denies the students the permission  $\text{asg}$ . More generally, a safety policy prohibits a set of users from certain roles or permissions. Correspondingly, the constraint  $|\text{user}[p] \cap \text{set}| = 0$  (respectively,  $|\text{user}[r] \cap \text{set}| = 0$ ) specifies that users in  $\text{set}$  ought not to acquire the permission  $p$  (respectively, to take the role  $r$ ).

*con5* All faculty members can both assign and view grades:  $\text{user}[\text{fac}] \subseteq \text{user}[\text{asg}] \cap \text{user}[\text{view}]$ . Note that this constraint is different from  $\{\text{view}, \text{asg}\} \subseteq \text{perm}[\text{fac}]$ , which is a sufficient condition for *con5* though.

*con5* is an example of *availability containment policy* [18], for it assures users of permissions. A general form of this policy,  $\text{user}[r] \subseteq \text{user}[p_1] \cap \dots \cap \text{user}[p_n]$ , says that a set of permissions  $\{p_1, \dots, p_n\}$  is available to members of a role  $r$ . On the other hand, a *safety containment policy*,  $\text{user}[p] \subseteq \text{user}[r_1] \cap \dots \cap \text{user}[r_m]$ , states that every user having a permission  $p$  is assigned to a set of roles  $\{r_1, \dots, r_m\}$ . For example, the constraint  $\text{user}[\text{chg}] \subseteq \text{user}[\text{dean}]$  requires that the permission to change grades be available to the dean.

*con6* No user can both receive and assign grades. This is an instance of a simple but widely used form of *separation-of-duty (sod)* policy. It can be interpreted either statically or dynamically, corresponding to *static sod (ssod)* and *dynamic sod (dsod)*. In line with the *ssod* policy, a user can not have both permissions (via any combination of roles). By contrast,

the dsod policy allows users to obtain both permissions; but it forbids users from activating them both [9].

Consider a constraint  $con6_d : |\text{role}[\text{rec}] \cap \text{role}[\text{asg}]| = 0$ ; it demands that any role be assigned to at most one of  $\text{rec}$  and  $\text{asg}$ . This constraint sets the basis for the dsod policy enforcement. With  $con6_d$  in effect, what remains is to prevent a user from activating both a role assigned to  $\text{rec}$  and a role assigned to  $\text{asg}$ ; this is beyond the scope of this paper.

Another constraint  $con6_s : |\text{user}[\text{rec}] \cap \text{user}[\text{asg}]| = 0$  is needed to enforce the ssod policy. It states that no user has both  $\text{rec}$  and  $\text{asg}$ , as required by the ssod policy exactly.

It can be verified that  $\gamma_{uni}$  satisfies  $con1$ ,  $con3$ ,  $con5$ , and  $con6_d$ , but not  $con2$ ,  $con4$ , or  $con6_s$ .

### 3.2 The Problem

$RCP$  is to, given a set  $C$  of constraints and a set  $\Gamma$  of configurations, find a configuration that satisfies  $C$  and is also close to  $\Gamma$ . First of all, we list and explain the assumptions.

**AS1** We assume that  $\mathcal{U}_1 = \mathcal{U}_2$  and  $\mathcal{P}_1 = \mathcal{P}_2$  for any  $\gamma_1, \gamma_2 \in \Gamma$ . As configurations in  $\Gamma$  are usually made for the same system, it is safe to assume that the system has fixed sets of users and of permissions during problem solving stage. Hence, this assumption is commonly met.

**AS2** We assume that  $\mathcal{R}_1 = \mathcal{R}_2$  for any  $\gamma_1, \gamma_2 \in \Gamma$ . This assumption simplifies the presentation of our approach. We lift this assumption in Section 6.2.

Therefore, we may simply assume that configurations in  $\Gamma$  build atop the fixed sets  $\mathcal{U}$ ,  $\mathcal{R}$ , and  $\mathcal{P}$ . Let  $\text{space}(\mathcal{U}, \mathcal{R}, \mathcal{P})$  be the set of all configurations that may be formed based on them; formally,  $\text{space}(\mathcal{U}, \mathcal{R}, \mathcal{P}) = \{(\mathcal{U}, \mathcal{R}, \mathcal{P}, UA, PA) \mid UA \subseteq \mathcal{R} \times \mathcal{U} \wedge PA \subseteq \mathcal{R} \times \mathcal{P}\}$ .

**AS3** We assume that if a configuration  $\gamma_0$  is a solution then it belongs to  $\text{space}(\mathcal{U}, \mathcal{R}, \mathcal{P})$ .

With **AS3**, we only search in  $\text{space}(\mathcal{U}, \mathcal{R}, \mathcal{P})$  for a solution  $\gamma_0$ . Note that, despite **AS3**,  $\gamma_0$  may contain *dangling* roles, those not associated with any user or permission; one can remove these roles from  $\gamma_0$ . In other words, **AS3** only requires  $\mathcal{R}_0 \subseteq \mathcal{R}$  but not  $\mathcal{R}_0 = \mathcal{R}$ . **AS3** will also be discussed in Section 6.2.

Second, given the set  $C$  of constraints, it is likely that many configurations satisfy  $C$ . There has to be a measurement of configurations' quality in order to select among them. Suppose that  $\gamma_0$  is one of the configurations satisfying  $C$ . The evaluation of  $\gamma_0$ 's quality ought to take  $\Gamma$  into account, as the configurations in  $\Gamma$  go through changes to make  $\gamma_0$ . Intuitively, the fewer changes are made, the less perturbation is caused and the more cost-effective the whole process is.

We use a measurement of *closeness* between  $\Gamma$  and  $\gamma_0$  to assess its quality. Observe that configurations differ in their user-role relations, role-permission, and user-permission relations. For  $\gamma_1 \in \Gamma$ , define the *difference* between  $\gamma_0$  and  $\gamma_1$  as  $\text{diff}(\gamma_1, \gamma_0) = (\gamma_0 - \gamma_1) \cup (\gamma_1 - \gamma_0)$ , where<sup>3</sup>

$$\begin{aligned} \gamma_0 - \gamma_1 &= (UA_0 \setminus UA_1) \cup (PA_0 \setminus PA_1) \cup (UPA_0 \setminus UPA_1) \\ \gamma_1 - \gamma_0 &= (UA_1 \setminus UA_0) \cup (PA_1 \setminus PA_0) \cup (UPA_1 \setminus UPA_0). \end{aligned}$$

Hence,  $\gamma_0$  differs from  $\gamma_1$  in  $|\text{diff}(\gamma_1, \gamma_0)|$  many places. We define the *distance* between  $\Gamma$  and  $\gamma_0$  as

$$\text{dist}(\Gamma, \gamma_0) = \sum_{\gamma_i \in \Gamma} |\text{diff}(\gamma_i, \gamma_0)|$$

<sup>3</sup>Given two sets  $A$  and  $B$ ,  $A \setminus B = \{a \in A \mid a \notin B\}$ .

Finally, we are able to define the problem.

**Definition 6.** Given a constraint set  $C$  and a configuration set  $\Gamma$ , find a configuration  $\gamma_0$  such that  $\gamma_0$  satisfies  $C$  and  $\text{dist}(\Gamma, \gamma_0)$  is minimized. Denote this problem as  $RCP\langle C, \Gamma \rangle$ .

**Example 7.** Suppose that an RE tool discovers the configuration  $\gamma_{uni}$  in Example 1 for the university. Suppose further that security officers of the university require the constraints in  $C_{uni} = \{con1, con2, con3, con4, con5, con6_s\}$ , among which  $\gamma_{uni}$  satisfies only  $con1$ ,  $con3$ , and  $con5$ . In this case, one may resolve  $RCP\langle C_{uni}, \{\gamma_{uni}\} \rangle$  to reach a qualified configuration, if any.

**Example 8.** For another example, suppose that  $\gamma_{uni}$  is the running configuration in the university. But an RE tool suggests that the university migrate to another configuration  $\gamma'_{uni}$ , whose user-role and role-permission relations are shown below.

$$\begin{aligned} UA'_{uni} &= \{(ta, alice), (stu, bob), (fac, carl), (dean, dave)\} \\ PA'_{uni} &= \{(stu, rec), (ta, asg), (fac, asg), \\ &\quad (dean, asg), (dean, view), (dean, chg)\} \end{aligned}$$

Suppose again that security officers need to enforce all constraints in  $C_{uni}$  (see Example 7). Neither  $\gamma_{uni}$  nor  $\gamma'_{uni}$  meets the requirements (as  $\gamma'_{uni}$  does not satisfy  $con5$ ). However, they may somehow capture the access control policy of the university. Rather than working out a configuration from scratch, we could try to resolve  $RCP\langle C_{uni}, \{\gamma_{uni}, \gamma'_{uni}\} \rangle$ .

### 3.3 Computational Complexity

$RCP$  is NP-hard. To show this, we will prove the NP-completeness of its corresponding decision problem. Given a positive integer  $K$ , the *decision problem of  $RCP\langle C, \Gamma \rangle$* , denoted as  $D-RCP\langle C, \Gamma, K \rangle$ , decides whether there is a configuration  $\gamma_0$  such that  $\gamma_0$  satisfies  $C$  and  $\text{dist}(\Gamma, \gamma_0) \leq K$ . We first consider a subclass where constraints in  $C$  are all quantity ones, then a subclass with constraints being structural, and finally a special subclass in light of [27].

**Theorem 9.**  $D-RCP\langle C, \Gamma, K \rangle$  is NP-complete, where constraints in  $C$  are *quantity* constraints.

See Appendix A for the proof. To show its NP-harness, we reduce the NP-complete problem *satisfiability (SAT)* to  $D-RCP\langle C, \Gamma, K \rangle$ . The idea is to use user-role assignments to model truth assignments of Boolean variables, and quantity constraints to model clauses of SAT.

**Theorem 10.**  $D-RCP\langle C, \Gamma, K \rangle$  is NP-complete, where constraints in  $C$  are *structural* constraints.

The proof is omitted due to space limits. When constraints only take the form  $\text{perm}[u] = \{p_1, \dots, p_n\}$ , the complexity result remains. This implies that the problem is hard, even when set expressions are limited to simple ones, which do not use  $\cap$  or  $\cup$  and are *semi-static*.<sup>4</sup> These constraints actually model the user-permission relation of the role mining problem [28].

Sun et al. [27] prove NP-hardness results of several subclasses of the *Assignment Feasibility Problem (AFP)*; the most of them have a counterpart of  $RCP$  with the same complexity. However, while a subclass of AFP ([27, Lemma 11]) is in P, its counterpart subclass of  $RCP$  is NP-hard.

**Theorem 11.**  $D-RCP\langle C, \Gamma, K \rangle$  is NP-complete, where constraints in  $C$  have one of the forms: (1)  $|\text{role}[u] \cap \{r\}| = 0$ , (2)

<sup>4</sup>A structural constraint  $s_1 \subseteq s_2$  is semi-static if either  $s_1$  or  $s_2$  evaluates to a set of constants.

$|\text{user}[r']| > 0$ , (3)  $\text{user}[r] \subseteq \text{user}[r']$ , and (4)  $\text{user}[r] = U$ , where  $\{r, r'\} \subseteq \mathcal{R}$ ,  $u \in \mathcal{U}$ , and  $U \subseteq \mathcal{U}$ .

The theorem can be proved by adapting the proof of [27, Lemma 19]. This implies that *RCP* is a different problem than AFP. Further comparison will be discussed in Section 7.

Faced with such a problem as  $RCP\langle C, \Gamma \rangle$ , we choose to encode it in answer set programming (ASP). The main reasons are two-fold. First, modern ASP solvers provide us with highly efficient inference engines, which work well in practice. Second, ASP has a rich modeling language, including programming constructs such as variables and aggregate operations. Consequently, the encoding is easy to understand and to explain to security officers how a configuration is selected. Moreover, it is straightforward to extend the encoding of *RCP* to handle its variants, as will be discussed in Section 6.1. We present such an encoding in Section 4.

## 4. REPRESENTATION IN ASP

### 4.1 ASP Preliminaries

ASP is a recent form of declarative programming approach to search problems. The idea is to first represent a search problem in a logic program, then employ ASP solvers to compute stable models (i.e., answer sets) of the program, and finally extract solutions of the search problem from the answer sets. Compared with other declarative approaches like SAT (SATisfiability problem), ASP features in its expressive modeling language. This advantage often leads to concise representation of problems. We review the main concepts of ASP. Readers are referred to [3, 13] for details.

An answer set program (or program for short) is a finite set of rules of the form

$$a \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n. \quad (1)$$

where  $a$  is either an atom or  $\perp$ ,  $b_i$  and  $c_j$  are atoms, and  $\text{not}$  denotes (default) negation. Besides,  $a$  is called the *head* of the rule and  $\{b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n\}$  the *body* of the rule. We say that a rule is a *fact* if  $m = 0$  and  $n = 0$ . For simplicity, we omit  $\leftarrow$  when writing facts.

In programs, variables are used to abbreviate rules of the same pattern. The ground instantiation of a program  $\Pi$ , denoted as  $G(\Pi)$ , is a program obtained by replacing the variables with terms from the underlying Herbrand universe of the program. An interpretation  $A$  is a subset of the Herbrand base of  $\Pi$ . We say that  $A$  *satisfies* a rule of the form (1) if  $A$  satisfies its head (i.e.,  $a \in A$ ) whenever  $A$  satisfies its body (i.e.,  $\{b_1, \dots, b_m\} \subseteq A$  and  $\{c_1, \dots, c_n\} \cap A = \emptyset$ ), and that  $A$  is an *answer set* of  $\Pi$  if  $A$  is the minimal set (in the sense of set inclusion) that satisfies all rules in  $G(\Pi)^A$ , which is obtained from  $G(\Pi)$  by

- first deleting all rules of form (1) where  $c_j \in A$  for some  $j \in [1, n]$ , and
- then deleting  $\text{not } c_j$  in the bodies of the remaining rules.

We use two statements of the ASP solver Clingo [13]. One is an aggregate of the form  $l \#count\{l_1, \dots, l_n\} u$ . This aggregate is true if the number of true literals in the set  $\{l_1, \dots, l_n\}$  is between  $l$  and  $u$ , inclusively. The other is an optimization statement of the form  $\#minimize [l_1, \dots, l_n]$ . It requires that the number of true literals in the multiset  $[l_1, \dots, l_n]$  be minimal.

### 4.2 Overview

$RCP\langle C, \Gamma \rangle$  can be encoded in at least two ways. With the assumptions in Section 3.2, we only search in  $\text{space}(\mathcal{U}, \mathcal{R}, \mathcal{P})$  for a

$$\begin{aligned} \Pi(\Gamma) &= \{config(\gamma) \mid \gamma \in \Gamma\} \cup \{config(\gamma_0)\} \\ &\cup \{hold(ask(r, u, ua), \gamma) \mid (r, u) \in UA \wedge \gamma \in \Gamma\} \\ &\cup \{hold(ask(r, p, pa), \gamma) \mid (r, p) \in PA \wedge \gamma \in \Gamma\} \\ &\cup \{urp(x) \mid x \in \mathcal{U} \cup \mathcal{R} \cup \mathcal{P}\} \cup \{role(r) \mid r \in \mathcal{R}\} \\ &\cup \{type(ua), type(pa)\} \end{aligned}$$

Figure 3: Definition of  $\Pi(\Gamma)$ .

solution. Since each configuration therein can be seen as a subset of  $\mathcal{R} \times \mathcal{U} \cup \mathcal{R} \times \mathcal{P}$ , we can use ASP rules to guess such a subset and thus obtain a configuration denoted as  $\gamma_0$ . Next, we define ASP rules to check if  $\gamma_0$  satisfies  $C$ . Finally we calculate the distance between  $\gamma_0$  and  $\Gamma$ . We use an alternative definition of distance: it is now calculated on the basis of assignments. Denote the set of all possible assignments  $\mathcal{R} \times \mathcal{U} \cup \mathcal{R} \times \mathcal{P} \cup \mathcal{U} \times \mathcal{P}$  as  $\mathcal{A}$ . For each  $a \in \mathcal{A}$ , if  $a$  holds in  $\gamma_0$  (i.e.,  $a \in UA_0 \cup PA_0 \cup UPA_0$ ), then the distance with respect to  $a$  is the number of configurations in  $\Gamma$  in which  $a$  does not hold; otherwise, it is the number of configurations in which  $a$  holds. For instance, suppose that  $\Gamma = \{\gamma_1, \gamma_2, \gamma_3\}$  and that  $a$  holds in  $\gamma_1$  and  $\gamma_2$ , but not  $\gamma_3$ . If  $a$  holds in  $\gamma_0$ , then the distance with respect to  $a$  is 1 as only  $\gamma_3$  does not contain  $a$ ; otherwise it is 2, as both  $\gamma_1$  and  $\gamma_2$  contain  $a$ . Then,  $\text{dist}(\Gamma, \gamma_0)$  is the sum of this number for all  $a \in \mathcal{A}$ . In other words, the set  $\Gamma$  actually puts a price at  $\gamma_0$  including an assignment and at its lacking the assignment. The distance is then the sum of  $\gamma_0$ 's cost for each  $a \in \mathcal{A}$ . This encoding is concise; it uses two ASP rules for guessing and minimizing distance. Also, it is irrelevant to  $|\Gamma|$ . However, it may contain  $|\mathcal{A}|$  many facts (before grounding), which is overwhelming for large deployments.

Another encoding is done by modeling changes to one configuration of  $\Gamma$ . We first select a configuration  $\gamma_1$  from  $\Gamma$  and guess a set of changes to it; the changes would take  $\gamma_1$  to another configuration, say  $\gamma_0$ . We then test if  $\gamma_0$  satisfies  $C$ . Recall that the distance between  $\Gamma$  and  $\gamma_0$  is the sum of the distance between  $\gamma$  and  $\gamma_0$  for  $\gamma \in \Gamma$ . Here, the latter distance is calculated by counting the assignments that hold in exactly one of  $\gamma_0$  and  $\gamma$ . We minimize the sum of this counting. In comparison with the first encoding, it does not represent  $\mathcal{A}$ , but only the assignments held in configurations in  $\Gamma$ . We observe from the configurations in Section 5 that the number of held assignments is much less than  $|\mathcal{A}|$ . Hence, we prefer and present this encoding.

### 4.3 Transformation

We now transform  $RCP\langle C, \Gamma \rangle$  to an ASP program, denoted as  $\Pi(RCP\langle C, \Gamma \rangle)$ . It consists of four parts: the first part  $\Pi(\Gamma)$  lays the basis, the second one  $\Pi(\gamma_0)$  guesses a potential solution  $\gamma_0$ , the third one  $\Pi(\text{quality})$  ensures closeness, and the fourth one  $\Pi(C)$  enforces constraints in  $C$ .

As shown in Figure 3,  $\Pi(\Gamma)$  contains facts for further reasoning. First, a fact  $config(\gamma)$  declares a configuration  $\gamma$ . Second, we use terms  $ask(r, u, ua)$  and  $ask(r, p, pa)$  to denote possible user-role and role-permission assignments, respectively; a fact  $hold(ask(r, u, ua), \gamma)$  means that an assignment  $(r, u)$  holds in  $\gamma$ . For each role  $r$ ,  $\Pi(\Gamma)$  contains a fact  $role(r)$ . A fact  $urp(x)$  means  $x$  is either a user, a role, or a permission. Finally, facts  $type(ua)$  and  $type(pa)$  denote the user-role relation and the role-permission relation, respectively.

Next we present  $\Pi(\gamma_0)$  in Figure 4. Suppose that  $\gamma_1 \in \Gamma$  is selected; a solution  $\gamma_0$  is regarded as reached by making changes to  $\gamma_1$ . Rules (2) and (3) define applicable changes to  $\gamma_1$ , which are either adding non-existing assignments or deleting existing ones.

$$\begin{aligned}
\text{change}(\text{add}(\text{asg}(R, X, T))) &\leftarrow \text{not hold}(\text{asg}(R, X, T), \gamma_1), \text{type}(T), \text{role}(R), \text{wrp}(X), \text{not role}(X). & (2) \\
\text{change}(\text{del}(\text{asg}(R, X, T))) &\leftarrow \text{hold}(\text{asg}(R, X, T), \gamma_1). & (3) \\
\text{applied}(G) &\leftarrow \text{change}(G), \text{not not\_applied}(G). & (4) \\
\text{not\_applied}(G) &\leftarrow \text{change}(G), \text{not applied}(G). & (5) \\
\text{hold}(\text{Asg}, \gamma_0) &\leftarrow \text{applied}(\text{add}(\text{Asg})). & (6) \\
\text{hold}(\text{Asg}, \gamma_0) &\leftarrow \text{not applied}(\text{del}(\text{Asg})), \text{hold}(\text{Asg}, \gamma_1). & (7) \\
\text{hold}(\text{asg}(U, P, \text{upa}), Y) &\leftarrow \text{hold}(\text{asg}(R, U, \text{ua}), Y), \text{hold}(\text{asg}(R, P, \text{pa}), Y). & (8) \\
\text{dif}(\text{Asg}, Y) &\leftarrow \text{hold}(\text{Asg}, \gamma_0), \text{not hold}(\text{Asg}, Y), \text{config}(Y). & (9) \\
\text{dif}(\text{Asg}, Y) &\leftarrow \text{not hold}(\text{Asg}, \gamma_0), \text{hold}(\text{Asg}, Y). & (10)
\end{aligned}$$

Figure 4: Rules in  $\Pi(\gamma_0)$ .

In rule (2), when  $\text{wrp}(x)$  holds but  $\text{role}(x)$  does not,  $x$  is either a user or a permission. Rules (4) and (5) guess a subset of changes that are applied to  $\gamma_1$ . Rules (6) and (7) further construct  $\gamma_0$  from the changes and  $\gamma_1$ : an assignment holds in  $\gamma_0$  if and only if the corresponding addition is applied (when the assignment does not hold in  $\gamma_1$ ) or the corresponding deletion is not applied (when the assignment holds in  $\gamma_1$ ). For each configuration in  $\Gamma$  and  $\gamma_0$ , rule (8) derives the user-permission relation from the user-role and role-permission relations; a term  $\text{asg}(u, p, \text{upa})$  denotes a possible user-permission assignment. Finally, for each  $\gamma \in \Gamma$ , rules (9) and (10) mark the assignments appearing in exactly one of  $\gamma_0$  and  $\gamma$ .

$\Pi(\text{quality})$  contains statement (11), which minimizes the marked assignments. As will be discussed in Section 6.1,  $\Pi(\text{quality})$  may include other statements to optimize solutions in ways other than closeness.

$$\# \text{minimize} [\text{dif}(\text{Asg}, Y) : \text{config}(Y)]. \quad (11)$$

Finally, we encode constraints. Each constraint  $c$  has an ASP program  $\Pi(c)$ ; we let  $\Pi(C) = \bigcup_{c \in C} \Pi(c)$ . Consider a structural constraint  $s \subseteq s'$ . Let  $s(x)$  denotes the fact that  $x$  is a member of  $s$  (evaluated against  $\gamma_0$ ). Rule (12) captures this constraint; it says a conflict arises if an answer set contains  $s(x)$  but not  $s'(x)$ . Therefore, an answer set of  $\Pi(\text{RCP}\langle C, \Gamma \rangle)$  includes  $s'(x)$  whenever it includes  $s(x)$ . That means, in configurations extracted from the answer sets, a member of  $s$  also belongs to  $s'$ .

$$\perp \leftarrow s(X), \text{not } s'(X). \quad (12)$$

Now the question is how to define  $s(X)$  (and  $s'(X)$ ). Observe that a set expression  $s$  can be rewritten (in polynomial time) as below:

$$s = \bigcap_{i=1}^m s_i \quad s_i = \bigcup_{j=1}^{k_i} s_{i,j} \quad (13)$$

where  $m, k_1, \dots, k_m$  are positive integers, and  $s_{i,j}$  is of the form  $\text{user}[x]$ ,  $\text{role}[x]$ , or  $\text{perm}[x]$ .

To define such a set expression, rule (14) models the intersection; it says that a ground instance of  $s(X)$  holds if so do the corresponding ground instances of  $\{s_{i1}(X), \dots, s_{im}(X)\}$ .

$$s(X) \leftarrow s_{i1}(X), \dots, s_{im}(X). \quad (14)$$

In turn, rule (15) models the union; it states that a ground instance of  $s_{ij}(X)$  holds whenever so does at least one of the corresponding instances of  $\{s_{i1}(X), \dots, s_{ik_i}(X)\}$

$$s_{ij}(X) \leftarrow 1 \{s_{i1}(X), \dots, s_{ik_i}(X)\}, \text{wrp}(X). \quad (15)$$

Note that  $s$  may evaluate to be a mixed set of users, roles, and/or permissions; so rule (15) contains  $\text{wrp}(X)$  in its body.

Since we want to know if  $\gamma_0$  satisfies  $C$ , set expressions are evaluated against  $\gamma_0$ . Accordingly, in the rules for  $s$  we replace  $s_{ij}(X)$  with  $\text{hold}(\text{Asg}, \gamma_0)$ , depending on  $s_{i,j}$ 's type. For example,  $s_{i,j} = \text{user}[\text{stu}]$  evaluates to the set of users who take the role  $\text{stu}$ ; in this case,  $\text{hold}(\text{asg}(\text{stu}, X, \text{ua}), \gamma_0)$  takes place of  $s_{ij}(X)$ .

For a quantity constraint  $|s|\theta n$ , the translation is similar. We first encode  $s$ , and then count the number of members in  $s$ . Rules (16)-(19) represent the constraint when  $\theta$  takes each one of  $\{=, \neq, \leq, \geq\}$ .

$$\perp \leftarrow \text{not } n \# \text{count}\{s(X)\} n. \quad (16)$$

$$\perp \leftarrow n \# \text{count}\{s(X)\} n. \quad (17)$$

$$\perp \leftarrow n + 1 \# \text{count}\{s(X)\}. \quad (18)$$

$$\perp \leftarrow \# \text{count}\{s(X)\} n - 1. \quad (19)$$

Rule (16) says an exact number  $n$  of ground instances of  $s(X)$  hold; otherwise there arise a conflict. On the contrary, rule (17) forbids the case where exactly  $n$  many such instances hold. The remaining two rules work likewise.

### Optimization.

The above translation is the most general approach; every  $\text{RCP}$  instance can be handled in this way. However, the resulting program is not necessarily optimal. With *domain knowledge* of the problem, further optimization to the encoding is possible. For example, domain knowledge enables us to characterize those constraints most likely to be used in practice, and hence to identify a number of strategies for optimizing the transformation. See Appendix C for discussions.

**Example 12.** Take constraint  $\text{con5}$ :  $\text{user}[\text{fac}] \subseteq \text{user}[\text{asg}] \cap \text{user}[\text{view}]$  for example; the following program encodes it.

$$\left\{ \begin{array}{l} \text{con5\_s}(X) \leftarrow \text{hold}(\text{asg}(\text{fac}, X, \text{ua}), \gamma_0). \\ \text{con5\_s'}(X) \leftarrow \text{hold}(\text{asg}(X, \text{asg}, \text{upa}), \gamma_0), \\ \quad \text{hold}(\text{asg}(X, \text{view}, \text{upa}), \gamma_0). \\ \perp \leftarrow \text{con5\_s}(X), \text{not } \text{con5\_s'}(X). \end{array} \right.$$

The first two rules corresponds to the set expressions  $\text{user}[\text{fac}]$  and  $\text{user}[\text{asg}] \cap \text{user}[\text{view}]$ , respectively. The last rule relates the two sets, in light of rule (12). Note that this program already uses the optimization in Appendix C.

**Theorem 13.**  $\Pi(\text{RCP}\langle C, \Gamma \rangle)$  has an answer set if and only if  $\text{RCP}\langle C, \Gamma \rangle$  has a solution.

See Appendix B for a sketch of the proof.

Table 2: The configurations used in experiments.  $V_{UA}$  is the average number of roles that users are assigned to,  $V_{PA}$  the average number of permissions roles are assigned to, and  $V_{UPA}$  the average number of permissions users obtain.

Datasets	$ \mathcal{U} $	$ \mathcal{R} $	$ \mathcal{P} $	$V_{UA}$	$V_{PA}$	$V_{UPA}$
Dom	79	20	231	1.39	30.15	9.24
EMEA	35	34	3046	1.00	212.09	206.29
FW1	365	66	709	2.39	16.35	87.53
FW2	325	10	590	1.34	67.60	112.08
HCare	46	15	46	2.31	6.87	32.30
MelbS	51	48	185	1.41	9.71	10.59
USA	10021	276	277	4.44	1.88	4.53
APJ	2044	454	1164	1.19	3.70	3.35
AmSm	3477	192	1587	1.37	26.29	30.26
AmLa	3485	404	10127	1.14	211.65	53.17

## 5. EXPERIMENTS

Cases where  $|\Gamma| = 1$ .

Table 2 shows a collection of configurations [8].<sup>5</sup> These configurations have been widely tested in role engineering community. For each such configuration  $\gamma$ , we construct an experimental instance  $RCP\langle C, \{\gamma\} \rangle$ . The set  $C$  comprises  $C_{ran}$  and  $C_{man}$ . Constraints in  $C_{ran}$  are generated based on  $\gamma$  as follows.

- For each  $u \in \mathcal{U}$ , create a pair of constraints  $\text{role}_\gamma[u] \setminus R_1 \subseteq \text{role}[u] \subseteq \text{role}_\gamma[u] \cup R_2$ ;  $R_1$  and  $R_2$  are randomly chosen from  $\text{role}_\gamma[u]$  and  $\mathcal{R} \setminus \text{role}_\gamma[u]$ , respectively, such that  $|R_1| = \alpha^- \times |\text{role}_\gamma[u]|$  and  $|R_2| = \alpha^+ \times |\text{role}_\gamma[u]|$ , where  $\alpha^-$  and  $\alpha^+$  are positive numbers.
- For each  $r \in \mathcal{R}$ , create a pair of constraints  $\text{perm}_\gamma[r] \setminus P_1 \subseteq \text{perm}[r] \subseteq \text{perm}_\gamma[r] \cup P_2$ ;  $P_1$  and  $P_2$  are randomly chosen from  $\text{perm}_\gamma[r]$  and  $\mathcal{P} \setminus \text{perm}_\gamma[r]$ , respectively, such that  $|P_1| = \alpha^- \times |\text{perm}_\gamma[r]|$  and  $|P_2| = \alpha^+ \times |\text{perm}_\gamma[r]|$ .
- For each  $u \in \mathcal{U}$ , create a pair of constraints  $\text{perm}_\gamma[u] \setminus P_1 \subseteq \text{perm}[u] \subseteq \text{perm}_\gamma[u] \cup P_2$ ;  $P_1$  and  $P_2$  are randomly chosen from  $\text{perm}_\gamma[u]$  and  $\mathcal{P} \setminus \text{perm}_\gamma[u]$ , respectively, such that  $|P_1| = \alpha^- \times |\text{perm}_\gamma[u]|$  and  $|P_2| = \alpha^+ \times |\text{perm}_\gamma[u]|$ .

We now explain the constraints in  $C_{ran}$ . First, it is easy to see that  $\gamma$  satisfies  $C_{ran}$ . Second,  $C_{ran}$  is parameterized by  $(\alpha^+, \alpha^-)$ . Consider  $\text{perm}_\gamma[r] \setminus P_1 \subseteq \text{perm}[r] \subseteq \text{perm}_\gamma[r] \cup P_2$  for example and suppose  $|\text{perm}_\gamma[r]| = 10$ ,  $\alpha^+ = 1.5$ , and  $\alpha^- = 0.9$ . This pair of constraints states that  $r$  must have one permission (which is randomly chosen from  $\text{perm}_\gamma[r]$ ), and are allowed to have another 15 permissions (which are randomly chosen from  $\mathcal{P} \setminus \text{perm}_\gamma[r]$ ) in addition to those in  $\text{perm}_\gamma[r]$ . In this sense, each pair of constraints in  $C_{ran}$  models a *range* of, for example, permissions that a role could obtain. When  $\alpha^- \geq 1$ , the pair of constraints is reduced to  $\text{perm}[r] \subseteq \text{perm}_\gamma[r] \cup P_2$ . Finally, we believe these range constraints arise in practice; see Appendix C for discussions.

As for  $C_{man}$ , we manually create a number  $\beta$  of constraints of the following patterns:

- $\text{role}[p_1] \cup \text{role}[p_2] \subseteq \text{role}[p_3] \cap \text{role}[p_4]$  to let any role assigned to  $p_1$  or  $p_2$  have both  $p_3$  and  $p_4$  as well,
- $|\text{user}[p_1] \cap \text{user}[p_2]| = 0$  to support sod policy regarding  $p_1$  and  $p_2$ , and

<sup>5</sup>Ene et al. [8] applied their RE algorithms to real-world access control rules and obtained these RBAC configurations, which are available at: [http://www.hpl.hp.com/personal/Robert\\_Schreiber/](http://www.hpl.hp.com/personal/Robert_Schreiber/).

- $\text{user}[r_1] \subseteq \text{user}[r_2]$  and  $\text{perm}[r_2] \subseteq \text{perm}[r_1]$  so that any user of  $r_1$  is a member of  $r_2$  and any permission of  $r_2$  is also assigned to  $r_1$ .

It is guaranteed that at least two thirds of the constraints in  $C_{man}$  are not satisfied by  $\gamma$ .

Experiments were performed on a Windows 7 laptop with Intel Core 2.66GHz i5-560M processor and 4GB RAM. ASP programs were executed with the grounder *gringo* 3.0.3 and the solver *clasp* 2.0.0.<sup>6</sup> Our concerns lie in the efficiency and the scalability of the approach. Each test was limited to 600 seconds and forced to terminate otherwise. Table 3 shows the experiment results, with varying  $(\alpha^+, \alpha^-)$  but fixed  $\beta = 40$ . For each setting of  $(\alpha^+, \alpha^-)$  in each data-set, the result was averaged over 3 runs with constraints generated separately. Each run includes both the transformation process and ASP solving process.

Table 3: The computing time in seconds when  $\gamma_0$  does not exist and when it does. *t/o* denotes “timeout”.

$\alpha^-$	0.9				1.0	
$\alpha^+$	1.0	1.5	1.8	2.0	1.0	1.5
$\gamma_0$ does not exist						
Dom	0.32	0.29	0.28	0.32	1.71	1.80
EMEA	1.73	1.99	2.21	2.47	2.58	2.61
FW1	2.01	2.79	3.23	3.32	5.23	5.18
FW2	2.54	2.87	3.11	3.51	4.51	5.60
HCare	0.12	0.12	0.14	0.13	1.27	1.30
MelbS	0.20	0.18	0.22	0.22	0.36	0.42
USA	12.33	14.91	15.11	16.92	16.15	18.31
APJ	1.51	1.99	2.26	2.22	2.02	2.34
AmSm	15.20	21.28	25.05	27.31	16.28	24.32
AmLa	42.57	132.65	401.22	<i>t/o</i>	79.88	<i>t/o</i>
$\gamma_0$ exists						
Dom	0.82	2.62	6.17	6.29	3.53	7.83
EMEA	3.13	4.18	5.93	5.88	6.12	7.21
FW1	5.10	6.53	8.13	8.58	12.17	22.19
FW2	12.92	34.17	53.29	87.15	56.31	95.28
HCare	1.03	3.77	5.29	5.83	4.60	8.18
MelbS	1.32	2.92	4.03	6.29	2.01	6.89
USA	25.82	36.18	51.55	52.17	32.01	58.09
APJ	4.74	4.12	5.64	6.13	3.54	5.11
AmSm	18.23	30.03	38.85	56.93	35.17	43.25
AmLa	219.16	426.83	<i>t/o</i>	<i>t/o</i>	193.27	<i>t/o</i>

When no solution existed for tested  $RCP$  instances, all tests terminated within the time limit, except for AmLa. The results were encouraging; even for large-size data-set AmSm, the tests returned within 30 seconds. When a solution  $\gamma_0$  was found, the computing time was more demanding, but still acceptable; for example, solutions were returned within 60 seconds for AmSm. The tests for AmLa timed out as  $(\alpha^+, \alpha^-)$  increases.

It is interesting that the tests for AmSm were about 4 to 10 times less efficient than those for APJ, while they are well-matched in terms of  $|\mathcal{U}|$  and  $|\mathcal{P}|$ . However, we notice that APJ has twice as many role as AmSm; as a result,  $V_{PA}$  and  $V_{UPA}$  of AmSm are much larger than those of APJ. It seems plausible to say that the time is sensitive to  $V_{PA}$  and  $V_{UPA}$ . This is further confirmed by FW2. Medium-size as FW2 is,  $V_{PA}$  is 67.6 and  $V_{UPA}$  is 112.08; the tests of FW2 were much slower than those of FW1. However, one exception is EMEA. Despite the large  $V_{PA}$  and  $V_{UPA}$ , the tests were efficient, perhaps because of the small  $|\mathcal{U}|$ . The number of users also played a dominate role in the USA cases.

In Figure 5, we compare the performance when  $\beta$  takes 20, 40, and 60, respectively, with  $\alpha^+ = 1.5$  and  $\alpha^- = 0.9$ . It can be seen that the value of  $\beta$  has an impact, especially on large data-sets. When  $\beta = 60$ , the tests of AmLa timed out. Except for AmLa, the approach scaled well with respect to  $\beta$ . We are interested in the

<sup>6</sup><http://sourceforge.net/projects/potassco/>.

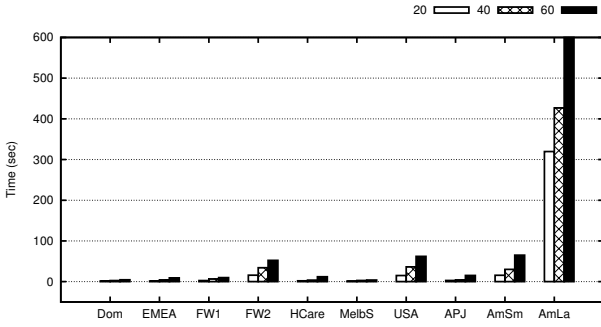


Figure 5: The comparison of computing time with a solution returned when varying  $\beta = 20$ ,  $\beta = 40$ , and  $\beta = 60$ .

number of constraints in  $C_{man}$  rather than  $C_{ran}$  for two reasons: (1) the encoding for  $C_{man}$  is more complex, which may result in considerable overhead; (2)  $\gamma$  satisfies  $C_{ran}$  but not  $C_{man}$ .

### Cases where $|\Gamma| > 1$ .

For each data-set, we already have one configuration. To acquire more configurations, we ran the role mining function of IBM Role Modeling Assistant [15] with the user-permission assignments of each data-set.

However,  $\Gamma$  did not necessarily meet the assumption that  $\mathcal{R}_1 = \mathcal{R}_2$  for any  $\gamma_1, \gamma_2 \in \Gamma$ . As will be discussed in Section 6.2, we pre-processed  $\Gamma$  to meet the assumption. First, we composed a tuple of roles, each of which is from a configuration of  $\Gamma$ , compared the similarity between roles in the tuple, and regarded them as counterpart roles if they were similar enough (by some similarity threshold). We used the similarity measure of [29, Definition 3]. In this way, we actually named the roles of each configuration and picked the same name for counterpart roles. Second, if there exists  $\gamma_1, \gamma_2 \in \Gamma$  such that  $|\mathcal{R}_1| \neq |\mathcal{R}_2|$ , we assume, without loss of generality, that  $\gamma_1$  has the maximum number of roles in  $\Gamma$ . Then we create dummy roles to fill the vacancies of missing roles in each configuration except  $\gamma_1$ . In this way, we made the configuration set of the tested instances meet the assumption. We only tested the instances where  $|\Gamma| = 2$  and  $|\Gamma| = 3$ .

Here,  $C$  also consists of two parts. One is similar to  $C_{ran}$  of previous tests. Take the constraints on user-role relation for example. We first obtain two sets:  $\text{leastR}[u] = \bigcap_{\gamma_i \in \Gamma} \text{role}_{\gamma_i}[u]$  and  $\text{mostR}[u] = \bigcup_{\gamma_i \in \Gamma} \text{role}_{\gamma_i}[u]$ ; and then create a pair of constraints  $\text{leastR}[u] \setminus R_1 \subseteq \text{role}[u] \subseteq \text{mostR}[u] \cup R_2$ ;  $R_1$  and  $R_2$  are randomly chosen from  $\text{leastR}[u]$  and  $\mathcal{R} \setminus \text{mostR}[u]$ , respectively, such that  $|R_1| = \alpha^- \times |\text{leastR}[u]|$  and  $|R_2| = \alpha^+ \times |\text{mostR}[u]|$ . The other part is exactly  $C_{man}$ .

When transforming  $RCP\langle C, \Gamma \rangle$ , the selected configuration  $\gamma_1$  is chosen randomly from  $\Gamma$ . Table 4 reports the results, when  $\alpha^+ = 1.5$ ,  $\alpha^- = 0.9$ , and  $\beta = 40$ . Here we set the timeout limit as 1800 seconds. The reported time was averaged over 3 runs. First, the time was much more demanding than the instances where  $|\Gamma| = 1$ . Second, the performance seemed sensitive to  $|\Gamma|$ ; the tests for  $|\Gamma| = 3$  were almost 3 times slower than those for  $|\Gamma| = 2$ . When  $|\Gamma| = 3$ , the programs for AmLa ran out of memory within time limit.

## 6. DISCUSSIONS

### 6.1 The Quality of Solutions

#### Optimization Objectives.

In order to discover “good” configurations, RE usually has certain optimization objectives, such as minimizing the number of roles or the number of edges (i.e.,  $|UA| + |PA|$ ). ASP provides strong support in this regard. We first define the roles that are associated with any user or permission by rule (20); statement (21) minimizes the role set. Statement (22) minimizes the edge set.

$$\text{not\_dangling}(R) \leftarrow \text{hold}(\text{asg}(R, -, -), \gamma_0), \text{role}(R). \quad (20)$$

$$\# \text{minimize} \{ \text{not\_dangling}(R) \}. \quad (21)$$

$$\# \text{minimize} \{ \text{hold}(\text{asg}(X_1, X_2, T), \gamma_0) : \text{type}(T) \}. \quad (22)$$

#### Roles’ Semantic Meanings.

There is a possibility that a solution fails to retain roles’ semantic meanings. Structural constraints can be put to exclude such solutions. A role’s meaning is essentially captured by a real-world concept, which can be expressed as a set of attributes; a role is said to match its meaning if the set of its users is exactly the set of users who have all the attributes in the corresponding set [21]. Therefore, for a semantically meaningful role  $r$ , a constraint of the form  $\text{user}[r] = \{u_1, \dots, u_n\}$  protects its meaning.

#### Bounded Changes.

Previously, an arbitrary number of changes could be applied to the selected configuration  $\gamma_1$ . However, since  $\gamma_1$  is either the running one or discovered by an RE tool, it makes more sense to seek a solution  $\gamma_0$  in proximity to  $\gamma_1$ . In this case, the number of changes applied to  $\gamma_1$  should be bounded. Suppose  $B$  is the bound.

We encode the requirement of bounded changes in an ASP program  $\Pi(B)$ . No doubt that  $\Pi(B)$  contains the following statement

$$0 \# \text{count} \{ \text{applied}(G) \} B.$$

which is a direct encoding of this requirement. We could have one more restriction. Let  $N$  be the number of assignments in  $\gamma_1$  (i.e.  $N = |UA_1| + |PA_1|$ ).  $\Pi(B)$  contains the following statement

$$(N - B) \# \text{count} \{ \text{hold}(\text{asg}(X_1, X_2, T), \gamma_0) : \text{type}(T) \} (N + B).$$

It says that one can make  $\gamma_0$  from  $\gamma_1$  by deleting at most  $B$  many assignments or by adding at most  $B$  many assignments.

#### Weights.

One may associate weights with configurations in  $\Gamma$ . For example, consider a migration process where an organization is running under a configuration  $\gamma_{cur}$  and may migrate into  $\gamma_{mig}$  suggested by RE tools. In this case,  $\Gamma = \{ \gamma_{cur}, \gamma_{mig} \}$ . Considering that  $\gamma_{mig}$  is optimized in various ways, one may associate  $\gamma_{mig}$  with a weight of 3 and  $\gamma_{cur}$  with a weight of 1 so that the closeness to  $\gamma_{mig}$  takes priority. Let  $\text{weight}(\gamma, w)$  denotes a fact that  $\gamma$  has a weight of  $w$ . To support weights, statement (23) is substituted for (11) in  $\Pi(\text{quality})$

$$\# \text{minimize} [\text{dif}(\text{Asg}, Y) : \text{weight}(Y, W) = W]. \quad (23)$$

#### Multiple Solutions.

It is likely that more than one solution exists for an  $RCP$  instance. Although these solutions are deemed equally good with respect to closeness to  $\Gamma$ , they may differ in other prospects, such as the above-mentioned optimization objectives. Since these metrics can vary from case to case and may be subjective, it is debatable to hard-code them in programs. ASP solvers are good enough to return all or a given number of solutions. Thus one may choose an appropriate configuration from them, either manually or by some evaluation algorithms. When closeness is not a major concern, one



Table 4: The computing time in seconds when a solution was found for  $RCP\langle C, \Gamma \rangle$ .  $o/m$  denotes “out-of-memory”.

	Dom	EMEA	FW1	FW2	HCare	MelbS	USA	APJ	AmSm	AmLa
$ \Gamma  = 1$	2.62	4.18	6.53	34.17	3.77	2.92	36.18	4.12	30.03	426.83
$ \Gamma  = 2$	3.17	5.84	9.44	41.18	5.37	6.75	43.94	8.61	50.62	812.75
$ \Gamma  = 3$	10.39	14.53	30.10	132.97	10.13	15.62	87.63	28.22	213.65	$o/m$

replaces  $\Pi(\text{closeness})$  with the following statement to search for solutions in specified proximity to  $\Gamma$ .

$$K \#count \{ dif(Asg, Y) : config(Y) \} K + \Delta .$$

Solutions thereof may exhibit certain properties that one is after.

## 6.2 Assumptions

Recall that we made the assumption that  $\mathcal{R}_1 = \mathcal{R}_2$  for any  $\gamma_1, \gamma_2 \in \Gamma$  (i.e., **AS2**) in Section 3.2. That is, all configurations in  $\Gamma$  have the same role set. This sometimes appears unreasonable. We now lift it by pre-processing configurations in  $\Gamma$ . **AS2** entails two requirements: (1) for each  $\gamma_i \in \Gamma$  there is a 1-1 mapping  $M_i$  from  $\mathcal{R}_i$  to a set  $RNames$  of role names, and (2)  $|\mathcal{R}_1| = |\mathcal{R}_2|$  for any  $\gamma_1, \gamma_2 \in \Gamma$ . For configurations that are designed in guide of business information analysis or mined in view of role semantics [5, 11, 21], these mappings could be established along with the setup of configurations. For configurations lacking this information, we may apply some similarity measurement to roles, such as the one used in [29, Definition 3], so that counterpart roles are mapped to the same name.

Assume that mappings are established for the requirement (1). For the requirement (2), we assume, without loss of generality, that  $\gamma_1$  has the largest number of roles. We create dummy roles for the other configurations to equalize the sizes of their role sets. Take  $\gamma_2$  for example. We need to add a number  $|\mathcal{R}_1| - |\mathcal{R}_2|$  of roles to  $\mathcal{R}_2$  and extend  $M_2$  by mapping dummy roles to names in  $RNames \setminus \{M_2(r) \mid r \in \mathcal{R}_2\}$ . We may further assume that the dummy roles are not assigned to any users or permissions.

One drawback of this pre-processing is that a solution  $\gamma_0$  always has the same number of roles as  $\gamma_1$ . From  $\gamma_2$ 's viewpoint, it has to add all its dummy roles. However, the addition of such roles is not counted in  $\text{dist}(\Gamma, \gamma_0)$ , thus having no effect on the selection of solutions.

To mitigate this side-effect, we define additional rules. Suppose that a dummy role  $r$  of  $\gamma_2$  remains not assigned to any users or permissions in  $\gamma_0$ ; then  $r$  could be neglected. From  $\gamma_2$ 's perspective,  $r$  can be removed as if it were never added. For each configuration  $\gamma \in \Gamma$ , denote the set of  $\gamma$ 's dummy roles as  $D_\gamma$ ; let  $\mathcal{D} = \bigcup_{\gamma \in \Gamma} D_\gamma$ . Define a mapping  $d : \mathcal{D} \mapsto \{1, \dots, N - 1\}$  such that for each  $r \in \mathcal{D}$ ,  $d(r) = |\{\gamma \in \Gamma \mid r \in D_\gamma\}|$ , where  $N = |\Gamma|$ ; namely,  $d(r)$  is the number of configurations where  $r$  is a dummy role, whereas  $N - d(r)$  is the number of configurations where it is not. If  $r$  remains not assigned to any user or permission in  $\gamma_0$ , it goes against  $N - d(r)$  many configurations; otherwise, it is not dummy in  $\gamma_0$ , and goes against  $d(r)$  many configurations. We minimize the number of configurations that dummy roles go against in total by the following rules, where  $dummy(r, v)$  are facts for  $r \in \mathcal{D}$  and  $v = d(r)$ .

$$\begin{aligned} assigned(R) &\leftarrow hold(asg(R, \_, \_), \gamma_0, dummy(R, \_)). \\ \#minimize[assigned(R) : dummy(R, V) = V \\ \text{not } assigned(R) : dummy(R, V) = N - V]. \end{aligned}$$

Another assumption (**AS3**) makes  $\mathcal{R}_0 \subseteq \mathcal{R}$ . The number of roles is an important metric of configurations' quality. Briefly, the fewer roles, the better a configuration is [28, 21]. Since configurations in  $\Gamma$  are either the running one or recommended by RE tools, they should contain a reasonable number of roles. Therefore, it

is worthwhile to investigate whether a solution is reachable under this assumption. In contrast, if a solution is allowed to contain new roles, various other implications remain to be clarified; for example, a natural question is how many new roles are enough. Finally, if no solution exists for an  $RCP$  instance, security officers may settle for a configuration that violates constraints to a limited extend, or allow a certain number of new roles. However, this deserves detailed study and is left for future work.

### Role hierarchy.

For simplicity, we omitted role hierarchy in RBAC configurations. It is not hard to extend the approach to handle it. We need to augment  $\Pi(\Gamma)$  with facts about role hierarchy, and  $\Pi(\gamma_0)$  with rules defining applicable changes to role hierarchy and rules modeling its inheritance and transitivity. However, this encoding of role hierarchy may result in blowup of problem size. A two-stage approach is of interest: first flatten configurations and solve  $RCP$ , and then construct role hierarchy. We leave this for future work.

## 6.3 Expressiveness of Constraints

We concern ourselves mainly with static constraints, instead of dynamic or historical constraints [1, 7]. We believe this suffices for  $RCP$ . First,  $RCP$  instances arise prior to the deployment of configurations or when the running configuration fails to meet new constraints; constraints should capture properties of configurations' relations, rather than their run-time behaviors. Second, it has been argued that certain security policies may be modeled as static constraints but not as dynamic ones [17]. This shows the more fundamental position of static constraints.

A general sod policy  $\langle P, k \rangle$  states that no  $k - 1$  or fewer users together have all permissions in  $P$ , where  $P \subseteq \mathcal{P}$  and  $1 < k \leq |P|$  is an integer [17]. When  $k = 2$ , the sod policy can be expressed by a single constraint  $|\bigcup_{p \in P} user[p]| = 0$ . When  $k > 2$ , we need to put  $\binom{|U|}{k-1}$  many constraints of the form  $|\bigcup_{j=1}^{k-1} perm[u_j] \cap P| < n$  to model the sod policy. In this case,  $C$  has an exponential blowup in size. Instead, one may consider encoding the sod policy in ASP directly; we leave this for future work. However, we believe sod policies with  $k = 2$  are more common.

## 7. RELATED WORK

A close related work is [16], which guarantees that no role contains more than a given number of permissions in the discovered configurations. For one thing, [16] deals with a special class of the constraints considered in this work. For another, we support constraints in a modular way. In [16], the proposed algorithm accepts  $\langle \mathcal{U}, \mathcal{P}, UPA \rangle$  and constraints as input, and produces a configuration satisfying the constraints. Rather, our approach is fed with configuration candidates and constraints.

In general, our approach can be considered as a complement for existing RE methodologies. It fine-tunes the given set of configurations in the hope of reaching a configuration in compliance with constraints. There exists a wealth of literature on RE. The problem has been approached from different perspectives, e.g., business information analysis [5, 25], data mining [21, 30], and machine learning [11]. While these tools work out a configuration optimized in various ways, there is no guarantee of its consistency with constraints. Though Coyne [6] and Shin et al. [25] mention constraints

in passing; their approaches lack technical details of handling constraints.

Vaidya et al. [29] study the problem of discovering a optimal configuration that is similar to a deployed one. First, the similarity measurement only evaluates role-permission assignments. This measurement is not suitable for *RCP*, because differences in user-role and user-permission relations have equal, if not more, impacts on the acceptability of configurations. Second, *RCP* may taken into account configurations besides the deployed one. Finally, constraint support is not the focus of [29].

Lu et al. [20] propose an approach to constraint-aware role mining problem, which is essentially the original role mining problem [28] augmented with possible negative authorizations. The negative authorizations can help identify underlying constraints. An assumption is made that the user-permission assignments (the input of the problem) imply the information of constraints. This assumption appears less reasonable in two cases. First, when the user-permission assignments contain noisy data or errors [10, 22], the implied constraints may not suit the organization in question. Second, some constraints may not be embodied by the input. More specifically, [20] works with sod policy and exceptions, overlooking constraints of other types. In contrast, our work avoids the noisy data issue by relying on other RE tools such as [22] to produce configuration candidates. Further, we deal with constraints provided by the organization in question. Since constraints are usually high-level security and business requirements, we believe that the organization's security officers are able to propose constraints, especially when they can inspect the running configuration and/or the configurations suggested by RE tools.

Molloy et al. [21] present a roadmap for RE research. Among others, they mention one valuable problem—to update a deployed configuration in some optimal and “localized” way. This work could be viewed as a response to this problem. The closeness requirement limits the deviation of discovered configurations from the deployed one. Due to ASP's richness in modeling, other optimization objectives may be encoded as well.

Sun et al. [27] investigate the problem of assigning permission-roles to users under a variety of constraints, mostly from a computational complexity perspective. Our language can express all types of constraints there, except that [27] considers a user-role qualification relation. Sun et al. further consider the problem of generating user-role assignments that are consistent with given constraints and propose an algorithm. As implied by Theorem 11, [27] differs from this work in several aspects. First of all, constraints in place are different. Second, [27] puts more emphasis on authorization, and thus generates user-role assignments. By contrast, this work aims to find an appropriate configuration and consequently considers user-role relation, role-permission relation, and user-permission relation. To ensure solutions' quality, *RCP* also requires the configuration be close to the provided ones. There is no such requirement in [27]. Finally, we employ ASP to encode *RCP*, whereas Sun et al. reduce the problem to SAT. Though ASP and SAT are closely related, ASP programs tend to be more understandable.

Constraints in RBAC context are well-studied from different perspectives. Some works propose expressive languages for specifying various constraints [1, 7], whereas others put forward effective enforcement mechanisms for constraints [4, 7]. Neither do we propose a new language for expressing constraints nor a mechanism to enforce them. Li et al. [17] study the constraint generation problem, which is to derive constraints from high-level security policies and configurations. In contrast, *RCP* assumes the existence of constraints and wants to generate configurations. An access control system may face these two problems at different stages.

## 8. CONCLUSIONS

Constraints, an inherent component of RBAC models, capture important security and business requirements. In this paper, we have formulated the problem of enhancing role engineering with constraint support (i.e., *RCP*) and provided an ASP-based solution framework. The definition of *RCP* enables to utilize the huge body of role engineering tools and to add constraint support in a modular way. The framework harnesses ASP's rich modeling language to present a concise, declarative representation of *RCP*. We have also performed experiments to validate the framework. As for future work, we plan to consider *RCP* variants with more expressive constraints such as the resiliency policies [19].

## Acknowledgment

We thank the anonymous reviewers for their helpful comments. This publication was made possible by the support of an NPRP grant (NPRP 09-079-1-013) from the Qatar National Research Fund (QNRF). The statements made herein are solely the responsibility of the authors.

## 9. REFERENCES

- [1] G.-J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, 2000.
- [2] ANSI. *American national standard for information technology - role based access control*. ANSI INCITS 359-2004, Feb. 2004.
- [3] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003. <http://www.baral.us/bookone/>.
- [4] D. A. Basin, S. J. Burri, and G. Karjoth. Dynamic enforcement of abstract separation of duty constraints. In *ESORICS*, pages 250–267, 2009.
- [5] A. Colantonio, R. D. Pietro, A. Ocello, and N. V. Verde. A formal framework to elicit roles with business meaning in rbac systems. In *ACM Symposium on Access Control Models and Technologies*, pages 85–94, 2009.
- [6] E. J. Coyne. Role engineering. In *ACM Workshop on Role-Based Access Control*, page 4, 1995.
- [7] J. Crampton. Specifying and enforcing constraints in role-based access control. In *ACM Symposium on Access Control Models and Technologies*, pages 43–50, 2003.
- [8] A. Ene, W. G. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *ACM Symposium on Access Control Models and Technologies*, pages 1–10, 2008.
- [9] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.
- [10] M. Frank, J. M. Buhmann, and D. A. Basin. On the definition of role mining. In *ACM Symposium on Access Control Models and Technologies*, pages 35–44, 2010.
- [11] M. Frank, A. P. Streich, D. A. Basin, and J. M. Buhmann. A probabilistic approach to hybrid role mining. In *ACM Conference on Computer and Communications Security*, pages 101–111, 2009.
- [12] M. R. Garey and D. J. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

- [13] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. *A User's Guide to gringo, clasp, clingo, and iclingo*.
- [14] C. Giblin, M. Graf, G. Karjoth, A. Wespi, I. Molloy, J. Lobo, and S. Calo. Towards an integrated approach to role engineering. In *2nd ACM Workshop on Assurable & Usable Security Configuration (SafeConfig)*, 2010.
- [15] IBM. Tivoli security role modeling assistant. <https://www14.software.ibm.com/iwm/web/cc/earlyprograms/tivoli/tivrole/index.shtml>, 2011.
- [16] R. Kumar, S. Sural, and A. Gupta. Mining rbac roles under cardinality constraint. In *Proceedings of the 6th international conference on Information systems security, ICISS'10*, pages 171–185, Berlin, Heidelberg, 2010. Springer-Verlag.
- [17] N. Li, Z. Bizri, and M. V. Tripunitara. On mutually-exclusive roles and separation of duty. In *ACM Conference on Computer and Communications Security*, pages 42–51, 2004.
- [18] N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM Trans. Inf. Syst. Secur.*, 9(4):391–420, 2006.
- [19] N. Li, Q. Wang, and M. Tripunitara. Resiliency policies in access control. *ACM Trans. Inf. Syst. Secur.*, 12:20:1–20:34, April 2009.
- [20] H. Lu, J. Vaidya, V. Atluri, and Y. Hong. Constraint-aware role mining via extended boolean matrix decomposition. *IEEE Transactions on Dependable and Secure Computing*, 2012.
- [21] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with multiple objectives. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):36, Dec 2010.
- [22] I. Molloy, N. Li, Y. A. Qi, J. Lobo, and L. Dickens. Mining roles with noisy data. In *SACMAT*, pages 45–54, 2010.
- [23] A. C. O'Connor and R. J. Loomis. 2010 economic analysis of role-based access control. Technical report, RTI International, 2010.
- [24] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.
- [25] D. Shin, G.-J. Ahn, S. Cho, and S. Jin. On modeling system-centric information for role engineering. In *ACM Symposium on Access Control Models and Technologies*, pages 169–178, 2003.
- [26] S. Sinclair and S. W. Smith. What's wrong with access control in the real world? *IEEE Security & Privacy*, 8(4):70–73, 2010.
- [27] Y. Sun, Q. Wang, N. Li, E. Bertino, and M. Atallah. On the complexity of authorization in rbac under qualification and security constraints. *IEEE Transactions on Dependable and Secure Computing*, 8(6):883–897, 2011.
- [28] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In *ACM Symposium on Access Control Models and Technologies*, pages 175–184, 2007.
- [29] J. Vaidya, V. Atluri, Q. Guo, and N. Adam. Migrating to optimal rbac with minimal perturbation. In *ACM Symposium on Access Control Models and Technologies*, June 2008.
- [30] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *ACM Conference on Computer and Communications Security*, pages 144–153, 2006.

## APPENDIX

### A. PROOF OF THEOREM 9

A non-deterministic Turing machine can guess a configuration in  $\text{space}(\mathcal{U}, \mathcal{R}, \mathcal{P})$ , compute its distance from  $\Gamma$ , and verify if it satisfies  $C$ ; all can be done in polynomial time. Hence, the problem is in NP.

To show its NP-hardness, we reduce the NP-complete problem *satisfiability* (SAT) to problems of this sub-class. An SAT problem asks, given a set  $X$  of Boolean variables and a collection  $H$  of clauses over  $X$ , whether there is a satisfying truth assignment for  $H$  [12].

The idea is to use user-role assignments to model truth assignments of variables, and quantity constraints to model clauses in  $H$ . First, we let  $\Gamma = \{\gamma\}$  and construct  $\gamma$  as follows: let  $\mathcal{U} = \{u\}$ ,  $\mathcal{R} = \{r_x, r_{\bar{x}} \mid x \in X\}$ , and  $\mathcal{P} = \{p\}$ ; further let  $UA = \{(r_x, u), (r_{\bar{x}}, u) \mid x \in X\}$  and  $PA = \{(r_x, p), (r_{\bar{x}}, p) \mid x \in X\}$ . The assignments  $(r_x, u)$  and  $(r_{\bar{x}}, u)$  are meant to represent the truth-assignment and false-assignment to  $x$ , respectively. Second, we construct the set  $C$  of constraints. For each  $x \in X$ , we put constraint (24). This constraint requires that  $u$  take either  $r_x$  or  $r_{\bar{x}}$  but not both, thus modeling the fact that  $x$  is either true or false. For each clause  $\{l_1, \dots, l_n\}$  where  $l_i$  is either  $x_i$  or  $\bar{x}_i$ , we put constraint (25). This constraint requires that  $u$  have at least one role which corresponds to  $l_i$ . Finally, we let  $K = |X|$ .

$$\text{role}[u] \cap \{r_x, r_{\bar{x}}\} = 1 \quad (24)$$

$$\left| \bigcup_{i=1}^n (\text{role}[u] \cap \{r_{l_i}\}) \right| \geq 1 \quad (25)$$

Suppose that  $\tau : X \mapsto \{T, F\}$  is a solution to the SAT problem. We define  $\gamma_0$  as  $(r_x, u) \in UA_0$  and  $(r_{\bar{x}}, u) \notin UA_0$  if  $\tau(x) = T$  and  $(r_x, u) \notin UA_0$  and  $(r_{\bar{x}}, u) \in UA_0$  if  $\tau(x) = F$ . It can be verified that  $\text{dist}(\Gamma, \gamma_0) \leq K$ . We now show constraints are also satisfied by  $\gamma_0$ . Since only one of  $x$  and  $\bar{x}$  is true, constraints of the form (24) are satisfied. For a clause  $\{l_1, \dots, l_n\}$ , at least one literal  $l_i$  is true. Hence the assignment  $(r_{l_i}, u)$  holds in  $\gamma_0$ ; it follows that constraints of the form (25) are satisfied.

On the other hand, suppose  $\gamma_0$  is a solution. Then we let  $\tau(x) = T$  if  $(r_x, u) \in UA_0$  and  $\tau(x) = F$  otherwise. For any clause  $\{l_1, \dots, l_n\}$ , there is a constraint of the form (25), which is satisfied by  $\gamma_0$ . Hence,  $\text{role}_{\gamma_0}[u] \cap \{r_{l_i}\} \neq \emptyset$  for at least one  $l_i$ ; that is,  $l_i$  is true under  $\tau$ . Therefore the clause is also true.

### B. PROOF SKETCH OF THEOREM 13

We first prove the soundness. Denote  $\Pi(RCP\langle C, \Gamma \rangle)$  as  $\Pi$  and suppose that  $A$  is an answer set of  $\Pi$ . Then, from the definition of answer sets,  $A$  is a  $\subseteq$ -minimal model of  $G(\Pi)^A$ . We define  $\gamma_0$  as follows.

$$\begin{aligned} UA_0 &= \{(r, u) \mid \text{hold}(\text{asg}(r, u, ua), \gamma_0) \in A\} \\ PA_0 &= \{(r, p) \mid \text{hold}(\text{asg}(r, p, pa), \gamma_0) \in A\} \end{aligned}$$

We need to show  $\gamma_0$  is a solution of  $RCP\langle C, \Gamma \rangle$ . According to Definition 6, this suffices to prove that  $\gamma_0$  satisfies  $C$  and that  $\text{dist}(\Gamma, \gamma_0)$  is minimized. The latter is guaranteed by the statement (11). We prove the former below.

Consider a structural constraint  $s \subseteq s'$  and suppose that  $x \in s/\gamma_0$ . We show that  $x \in s'/\gamma_0$  also holds. First, from (13) and rules (14) and (15), it follows that  $s(x) \in A$ . Second, by rule (12), we have  $s'(x) \in A$  as well; because otherwise there arises a conflict and thus  $A$  is not a model of  $G(\Pi)^A$ . Further, notice that facts like  $s'(x) \in A$  can only be derived via rules (14) and (15). Since  $A$

$$\begin{aligned}
A = & \{hold(ask(r, u, ua), \gamma_0) \mid (r, u) \in UA_0\} \cup \{hold(ask(r, p, pa), \gamma_0) \mid (r, p) \in PA_0\} \cup \{hold(ask(u, p, upa), \gamma_0) \mid (u, p) \in UPA_0\} \\
& \cup \{change(add(ask(r, u, ua))), change(add(ask(r, p, pa))) \mid (r, u) \in UA_1, (r, p) \notin PA_1\} \\
& \cup \{change(del(ask(r, u, ua))), change(del(ask(r, p, pa))) \mid (r, u) \in UA_1, (r, p) \in PA_1\} \\
& \cup \{dif(ask(r, u, ua), \gamma_i) \mid (r, u) \in (UA_i \setminus UA_0) \cup (UA_0 \setminus UA_i), \gamma_i \in \Gamma\} \cup \{dif(ask(r, p, pa), \gamma_i) \mid (r, p) \in (PA_i \setminus PA_0) \cup (PA_0 \setminus PA_i), \gamma_i \in \Gamma\} \\
& \cup \{dif(ask(u, p, upa), \gamma_i) \mid (u, p) \in (UPA_i \setminus UPA_0) \cup (UPA_0 \setminus UPA_i), \gamma_i \in \Gamma\} \\
& \cup \{applied(change(add(r, u, ua))), applied(change(add(r, p, pa))) \mid (r, u) \in UA_0 \setminus UA_1, (r, p) \in PA_0 \setminus PA_1\} \\
& \cup \{applied(change(del(r, u, ua))), applied(change(del(r, p, pa))) \mid (r, u) \in UA_1 \setminus UA_0, (r, p) \in PA_1 \setminus PA_0\} \\
& \cup \{not\_applied(change(add(r, u, ua))), not\_applied(change(add(r, p, pa))) \mid (r, u) \notin UA_0 \cup UA_1, (r, p) \notin PA_0 \cup PA_1\} \\
& \cup \{not\_applied(change(del(r, u, ua))), not\_applied(change(del(r, p, pa))) \mid (r, u) \in UA_1 \cap UA_0, (r, p) \in PA_1 \cap PA_0\} \\
& \cup \{s(x) \mid x \in s/\gamma_0, s \text{ is a set expression in } C\}
\end{aligned}$$

Figure 6: The set  $A$  of facts constructed from  $\gamma_0$ .

is a  $\subseteq$ -minimal model of  $G(\Pi)^A$ ,  $A$  must contain a combination of facts of the form  $hold(ask(r, u, ua), \gamma_0)$ ,  $hold(ask(r, p, pa), \gamma_0)$ , and  $hold(ask(u, p, upa), \gamma_0)$ , which derives  $s'(x)$  by rules (14) and (15). Suppose otherwise that this combination does not exist but  $s'(x) \in A$ . In this case,  $A$  is not  $\subseteq$ -minimal among the models of  $G(\Pi)^A$ , because, informally, no evidence supports  $s'(x)$ . Therefore,  $A$  contains such facts and  $\gamma_0$  contains the corresponding assignments; these assignments together lead to  $x \in s'/\gamma_0$ . Similarly, we can prove the case for a quantity constraint.

We now prove the completeness. Suppose that  $\gamma_0$  is a solution of  $RCP\langle C, \Gamma \rangle$ . Consider the set  $A$  of facts in Figure 6. To show that  $A$  is an answer set of  $\Pi$ , we need to prove that  $A$  is a  $\subseteq$ -minimal model of  $G(\Pi)^A$ . To this end, we need to (1) enumerate each rule in  $G(\Pi)^A$  and verify that  $A$  satisfies the rule, and to (2) assure that any proper subset of  $A$  is not a model of  $G(\Pi)^A$ . It is not hard to verify (1); we now show (2).

First, suppose that a proper subset of  $A$  excludes a fact of the form  $hold(ask(r, u, ua), \gamma_0)$  or  $hold(ask(r, p, pa), \gamma_0)$ ; denote the subset as  $B$ . Note that only rules (6) and (7) take  $hold(ask(r, u, ua), \gamma_0)$  or  $hold(ask(r, p, pa), \gamma_0)$  as the heads. If  $B$  contains all  $A$ 's facts of the form  $applied(G)$  and  $not\_applied(G)$ , then  $B$  does not satisfy the ground instances of one of these two rules. On the other hand, if  $B$  lacks facts of the form  $applied(G)$  and  $not\_applied(G)$ , it is likely that  $B$  satisfies the ground instances of rules (6) and (7). However, a proper subset of  $A$  lacking at least one of the ground instances of  $applied(G)$  and  $not\_applied(G)$  does not satisfy one of the ground instances of rules (4) and (5).

Second, for the rules with  $change(add(ask(r, u, ua))), change(del(ask(r, u, ua))), hold(ask(u, p, upa), \gamma_i)$ , and  $dif(ask(r, u, ua), \gamma_i)$ , their bodies contain facts from  $\Pi(\Gamma)$  and/or facts  $hold(ask(r, u, ua), \gamma_0)$  or  $hold(ask(r, p, pa), \gamma_0)$ . Therefore, a proper subset of  $A$  not containing any of these facts does not satisfy ground instances of the rules (3), (4), or (8)-(10).

Finally, for rules in  $\Pi(C)$ , consider a constraint  $c : s \subseteq s'$ ; notice that  $x \in s/\gamma_0$  (respectively,  $x \in s'/\gamma_0$ ) if and only if  $s(x) \in A$  (respectively,  $s'(x) \in A$ ). Further, since  $\gamma_0$  satisfies  $c$ ,  $A$  satisfies the rules in  $G(\Pi)^A$  which originally belong to  $\Pi(c)$ . Suppose that  $B$  is a proper subset of  $A$  excluding at least  $s(x)$ . Since the rule with  $s(x)$  as the head includes only positive atoms in the body,  $B$  does not contain  $s_i(x)$  either, if we assume the rule takes the form (14) for some  $i$ . From rule (15), we know that some  $s_{ij}(x) \notin B$ . Recall that  $s_{ij}(x)$  is a fact of the form  $hold(ask(r, u, ua), \gamma_0)$ ,  $hold(ask(r, u, pa), \gamma_0)$ , or  $hold(ask(u, p, upa), \gamma_0)$ . Lacking one of them,  $B$  fails to satisfy some rule in  $G(\Pi)^A$ , as shown above.

## C. TRANSFORMATION IN PRACTICE

Domain knowledge may be used to simplify the program  $\Pi(RCP\langle C, \Gamma \rangle)$ . For example, if alice is a student (according to her profile), she can neither be a faculty nor a dean. In this case, we may exclude these assignments from  $\gamma_0$ ; thus the facts for these assignments are not needed. For another example, it appears unlikely for the role *stu* to acquire the permission of changing grades. Put formally, configurations often comply with constraints of the following form:

- $R_1 \subseteq \text{role}[u] \subseteq R_2$  for  $u \in \mathcal{U}$ ,
- $P_1 \subseteq \text{perm}[r] \subseteq P_2$  for  $r \in \mathcal{R}$ , and
- $P_3 \subseteq \text{perm}[u] \subseteq P_4$  for  $u \in \mathcal{U}$ .

In other words, there are ranges in which proper assignments should reside. A configuration determines exactly what a user/role can and cannot do. In contrast, these ranges are bounds, corresponding to what a user/role must be able to do and what a user/role is forbidden from, respectively. It is relatively easy and practical to define ranges.

With these constraints, we can define rules which hopefully accelerate ASP solving. Take  $R_1 \subseteq \text{role}[u] \subseteq R_2$  for instance. Rules like (26) and (27) interpret this pair of constraints, where  $r_1 \in R_1$  and  $r_3 \notin R_2$ ; ASP solvers can use them to reduce search space.

$$\perp \leftarrow \text{not } hold(ask(r_1, u, ua), \gamma_0). \quad (26)$$

$$\perp \leftarrow hold(ask(r_3, u, ua), \gamma_0). \quad (27)$$

One may notice that these ranges depend on extra information beyond  $C$  and  $\Gamma$ . This information is arguably available. For example, from users' positions, titles, and jobs, one can infer their role ranges. To estimate permission ranges for users and roles, several sources can be exploited, including access logs, project specifications, and organizational information (e.g., departments and groups) [21]. For example, if access logs show that  $u$  never used or even requested a permission  $p$ ,  $p$  should be out of  $u$ 's reach; on the other hand, if  $u$  performs  $p$  regularly,  $u$  shall obtain  $p$ . Although it is difficult to make an accurate estimate, a rough one would be practical and helpful.

Moreover, domain knowledge enables us to characterize those constraints most likely to be used in practice, and hence to identify a number of strategies for optimizing the transformation. For example, among the constraints we examined, most set expressions take the following forms

$$s = s_{1,1} \cap \dots \cap s_{m,1} \quad s = s_{1,1} \cup \dots \cup s_{1,k_1}.$$

We have a more concise representation of  $s$  as follows.

$$s(X) \leftarrow s_{11}(X), \dots, s_{m1}(X).$$

$$s(X) \leftarrow 1 \{s_{11}(X), \dots, s_{1k_1}(X)\}, \text{urp}(X).$$