CERIAS Tech Report 2005-83

A THEORY BASED ON SECURITY ANALYSIS FOR COMPARING THE EXPRESSIVE POWER OF ACCESS CONTROL MODELS

by Mahesh V. Tripunitara

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

A THEORY BASED ON SECURITY ANALYSIS FOR COMPARING THE EXPRESSIVE POWER OF ACCESS CONTROL MODELS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Mahesh V. Tripunitara

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2005

To Annie, Arthur, Gypsy and Pattie.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

ABSTRACT

Tripunitara, Mahesh V. Ph.D., Purdue University, December, 2005. A Theory Based on Security Analysis for Comparing the Expressive Power of Access Control Models. Major Professor: Ninghui Li.

We present a theory for comparing the expressive power of access control models. Our theory is based on reductions that preserve the results of security analysis. Security analysis is an approach to the verification of security policies in access control systems. We demonstrate the effectiveness of the theory by applying it in several cases. Also, we present related results on safety analysis in Discretionary Access Control (DAC) and security analysis in Role-Based Access Control (RBAC).

1 INTRODUCTION

In this dissertation, we show the existence of a theory based on security analysis for comparing access control models. We justify the design of the theory and demonstrate its effectiveness.

Access control enables the controlled sharing of resources among principals. It is "the traditional center of gravity" of computer security [1] and is recognized as an important area of research in computer security. An *access control system* decides whether an entity may access another entity. The entity that initiates access is called a principal, subject or user. The entity to which access is initiated is called a resource or object. When a subject requests access to an object, it specifies also the manner in which it desires such access; for example, read, write or execute. A manner of access is called an *access right*. An access right and an object are sometimes together referred to as a permission. An access control system makes its decision based on a *protection state*; at any given time, the protection state includes all information needed by the access control system to make its decision should a subject request access.

The protection state of an access control system can sometimes be changed by subjects in the system. In such a case, the rule by which the protection state can be changed is specified along with the protection state. Such a rule is called a *state change* or an *administrative* rule. Thus, an access control system is a state-change system and is specified by a double $\langle \gamma, \psi \rangle$, where γ is the start or current state and ψ is the state-change rule.

An example of an access control system is shown in Figure 1.1. The *access matrix* M represents the state γ. The commands comprise the state-change rule ψ. In γ, two subjects exist, Alice and Bob. The rows of M are indexed by subjects and columns by objects. Every subject is also an object. Each cell of M contains the set of rights a subject possesses to an object. For example, Alice has the own right to Bob, and Bob has the $read$ right to File_1. Each command in ψ takes parameters that are instantiated by subjects or

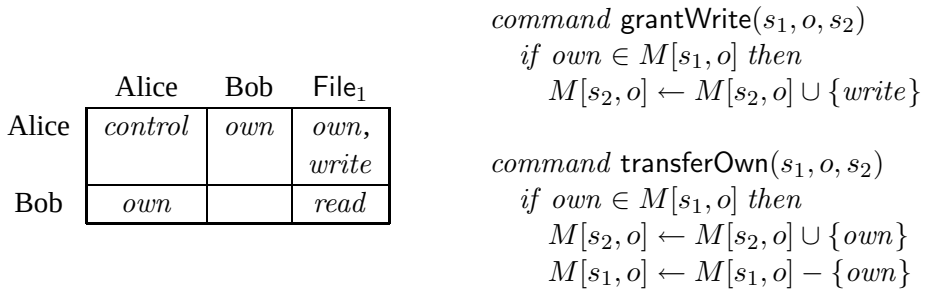


Figure 1.1. An access control system with the state represented as an access matrix, M, and state-change rule represented as a set of commands. M has two subjects, Alice and Bob, and an object File_1 besides the two subjects. There are two command associated with this system, grantWrite and transferOwn. Each command in the state-change rule takes parameters, each of which is instantiated by a subject or an object.

objects. For example, the transferOwn command is used by the subject s_1 to transfer the own right over the object o to the subject s_2.

An access control system is in an *access control scheme*; a scheme is specified by a double $\langle \Gamma, \Psi \rangle$, where Γ is a (possibly infinite) set of states, and Ψ is a (possibly infinite) set of state-change rules. A system $\langle \gamma, \psi \rangle$ is said to be in a scheme $\langle \Gamma, \Psi \rangle$ when $\gamma \in \Gamma$ and $\psi \in \Psi$. The access control system in Figure 1.1 is in the scheme proposed by Harrison et al. [2] that we call the HRU scheme. A scheme is in an *access control model*. An access control model is generally associated with the representation of the state for schemes in the model. An example of an access control model is the access matrix model [2–4], in which a state is represented by a matrix in which each cell, indexed by a $\langle subject, object \rangle$ pair, contains a set of rights. A scheme in the access matrix model has as its set of states Γ all possible instances of access matrices.

1.1 The need for safety and security analysis

Given an access control system in a scheme, a natural question that arises is the "state reachability" question; that is, whether the system can reach a particular state based on its start state γ and state-change rule ψ. For example, in the system in Figure 1.1, we may ask

whether the system can reach a state in which the subject Bob has the $write$ right to File_1. The question of whether a subject can acquire a right over an object is called the *safety* question, and the corresponding analysis in the context of systems in a scheme is called safety analysis [2]. More generally, the question of whether an access control system in a scheme can reach a state in which a *query* is true is called *security analysis* [5–7]. A query specifies some property of the state.

The need for safety and security analysis is articulated by Jones [8]: "Security policies are generally formulated as predicates relating the subjects and passive objects of a protection state. In contrast, most [access control systems] are phrased in a procedural, not a predicate, form. Though procedural definitions make individual system state transitions easy to understand and to implement, they combine to form a system that exhibits complex behavior. It is difficult to intuit and to express the behavior of a procedurally defined system. . . . the predicate defines a security policy. Thus, we bridge the gap between mechanism and policy."

A query in security analysis is a predicate; examples of queries in the context of an access matrix system are: "does Alice have access to an object, o?", "does Bob not have the 'own' right over o?" and "does some subject have some right over some object?" Safety and security analysis are forms of policy verification; they determine whether policies, stated as queries, hold in some reachable state, or in all reachable states.

The assertion in Jones [8] on the difficulty of intuiting the behavior of an access control system is substantiated by the work of Harrison et al. [2], in which it is demonstrated that the safety question of whether some subject can gain a particular right to some object is undecidable for systems in what appears to be a relatively simple and natural access matrix scheme, the HRU scheme.

1.2 On comparing access control models and schemes

There are three compelling motivations for comparing access control models and schemes with one another.

The first is simply that there exists more than one model, and therefore it is natural to ask how two models compare with one another. For example, in introducing the Schematic Protection Model (SPM) [9], Sandhu [10] considers its *expressive power*. Expressive power in this context is informally characterized as the range of policies a model can express. Sandhu [10] specifically compares SPM to the HRU scheme [2]. Similarly, Osborn et al. [11] compares the Role-Based Access Control (RBAC) model [12, 13] to Mandatory [14, 15] and Discretionary Access Control [3, 4, 16] (MAC and DAC, respectively) models.

The motivation discussed above applies especially when two models are claimed to be "policy neutral"; that is, when it is claimed that a wide variety of security policies can be expressed by the two models. An example is the comparison of RBAC with the access matrix model.

The second motivation is to understand whether a particular feature adds expressive power to a scheme. For example, Ammann et al. [17] addresses the question of whether the Extended Schematic Protection Model (ESPM) is more expressive than SPM. ESPM is exactly like SPM except for one feature: the ability to specify two "parents" in the creation of a new object (SPM allows for the specification of only one parent). Similarly, Sandhu and Ganta [18] addresses a similar question in the context of the Augmented Typed Access Matrix (ATAM) scheme in comparison to the Typed Access Matrix (TAM) scheme. The only difference between ATAM and TAM is that ATAM allows us to check for the absence of rights in a cell of the access matrix, while TAM does not.

The third motivation for comparing access control models and schemes is the need to infer results regarding safety and security analysis in a scheme, given that results are known in another scheme. For example, given that safety is undecidable in the HRU scheme [2], a mapping that preserves the results of safety analysis to another scheme would indicate that safety is undecidable in the latter scheme as well. We use this approach in Chapter 4 to infer complexity results for a variety of queries in two RBAC schemes, based on known results for corresponding analysis problems in a trust management scheme.

We argue that for a scheme to be at least as expressive as another, it must express all policies that the latter can. Our notion of a policy is articulated in Section 1.1; it is those predicates that are considered to be meaningful for systems in the scheme. Consequently, we say that an access control scheme B is at least as expressive as a scheme A if there exists a mapping of systems from A to B that preserves the results of security analysis in A. When we say preserves the results of security analysis, we mean that if security analysis returns "true" for a query in the system in A, then it must return "true" for the corresponding query in the system in B and vice versa. To compare models, we compare the schemes in the models. This characterizes our notion of expressive power in the context of access control schemes and models.

What we seek is a kind of *reduction*. Our notion of a reduction is similar to that used, for example, in structural complexity theory [19, 20]. In our case, however, a reduction is a mapping that preserves results of security analysis. It is not necessarily efficient; we seek only that the mapping is computable.

1.3 Thesis statement

Our thesis is that there exists a theory based on reductions that preserve the results of security analysis for comparing the expressive power of access control models. Our approach to proving the thesis is by construction; we present two such reductions in Chapter 5. We demonstrate the effectiveness of the reductions by applying them in several cases to compare access control schemes.

1.4 Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we discuss related work on access control models and schemes, safety and security analysis, and comparing access control models and schemes. In Chapter 3 we present a new result on safety analysis in DAC and critique a mapping from one DAC scheme to another that has been presented in the literature. This chapter serves as motivation for our theory. In

Chapter 4 we present new results on security analysis in RBAC. In this chapter, we use a weaker version of a reduction from our theory. In Chapter 5 we introduce our theory, justify its design and present applications. We conclude with Chapter 6.

2 RELATED WORK

In this chapter, we survey work related to our thesis. In Section 2.1, we discuss access control models schemes that have been proposed in the literature. In Section 2.2, we discuss work on safety and security analysis. In Section 2.3, we discuss work on comparing access control models and schemes.

2.1 Access control models and schemes

The first formal access control model to have been proposed in the literature is by Lampson [3]. It introduces the access matrix model, and discusses how the protection state may be changed by subjects in the system. Graham and Denning [4] has built upon Lampson [3] and provides a comprehensive description of a DAC scheme. In a DAC scheme, subjects grant rights to objects they own at their discretion. Graham and Denning [4] discusses two distinguished rights, own and $control$, that empower subjects to make changes to the protection state at their discretion. The characterization of DAC that we use in Chapter 3 is from [16], an earlier version of which appears as a research paper [21]. Griffiths and Wade [22] introduces a DAC scheme for relational database systems. Solworth and Sloan [23] introduces a DAC scheme based on labels and relabelling rules and argues that it captures all known DAC schemes. We discuss the scheme and the claim in detail in Chapter 3.

Harrison et al. [2] presents a different access matrix scheme from that proposed by Graham and Denning [4]. In the HRU scheme [2], there are no distinguished rights, and a state-change rule consists of an arbitrary set of commands each of a particular form.

Subsequently, several access control schemes have been proposed. Jones et al. [24,25] introduces the take-grant scheme. Sandhu introduces SPM [9]. Ammann and Sandhu [26] introduces ESPM, which extends SPM with multi-parent creation. Sandhu [27] intro-

duces TAM, which extends the HRU scheme [2] by requiring that subjects and objects be strongly typed. Soshi et al. [28, 29] introduces the dynamic typed access matrix model, in which the types of subject and objects may change.

Ferraiolo et al. [30, 31] introduces RBAC. Sandhu et al. [13] discusses the RBAC96 family of models, which provides a more precise characterization of the RBAC protection state. RBAC is described in detail also by Ferraiolo et al. [12]. Koch et al. [32, 33] presents a graph-based formalism for RBAC, and Nyanchama and Osborn [34] presents the role graph model, both of which use graphs to represent the RBAC protection state.

Sandhu et al. [35–37] introduces the ARBAC97 administrative scheme for RBAC. ARBAC97 [37] is the first and most comprehensive administrative scheme to have been proposed for RBAC, and we discuss its safety properties in detail in Chapter 4. Subsequently, administrative schemes for RBAC have been proposed by Crampton and Loizou [38–40], and by Koch et al. [32, 33, 41] in the context of a graph-based formalism for RBAC, and Wang and Osborn [42] for the role graph framework.

There has been work also on MAC schemes [14, 15, 43, 44]. In MAC systems, rights cannot be granted to subjects at the discretion of other subjects. MAC schemes can certainly be compared to DAC and RBAC schemes. We argue however that this is not as interesting, as MAC schemes are designed to enforce a specific policy, for example, multilevel security [45].

2.2 Safety and security analysis

Safety is a fundamental property that was first proposed in the context of access control by Harrison et al. [2]. That work demonstrates that safety is undecidable in general for the HRU scheme [2]. Harrison and Ruzzo [46] presents safety analysis results for several restricted cases of the HRU scheme. Subsequently, there has been considerable work on safety in various contexts related to security [5, 7, 9, 10, 23–29, 47–51].

Jones et al. [24, 25] shows that safety is decidable in linear time in the take-grant scheme. Sandhu [51] shows that safety is undecidable in SPM. Sandhu [9] demonstrates

that safety is decidable for a restricted version of SPM (with only "acyclic creation" allowed). Ammann and Sandhu [47] discusses safety analysis in ESPM. Soshi et al. [28,29] discusses safety in the dynamic typed access matrix scheme. Solworth and Sloan [23] shows that safety is decidable in the DAC scheme introduced in that work.

Koch et al. [41] shows that safety is decidable in a restricted scheme in the graph based formalism for RBAC [32, 33]. Li and Tripunitara [7] discusses security analysis in two RBAC schemes based on the ARBAC97 scheme [37]. The analysis there is based on results from security analysis in a trust management framework [5, 6].

Safety analysis has been studied in other contexts as well, such as grammatical protection systems [48] and trust negotiation [52].

2.3 Comparing access control models and schemes

Comparing the expressive power of access control models is recognized as a fundamental problem in computer security and is studied extensively in the literature [10,11,17, 18, 53–55]. A common methodology used for comparing access control models in previous work is *simulation*. When a scheme A is simulated in a scheme B, each system in A is mapped to a corresponding system in B. If every scheme in one model can be simulated by some scheme in another model, then the latter model is considered to be at least as expressive as the former. Furthermore, if there exists a scheme in the latter model that cannot be simulated by any scheme in the former, then the latter model is strictly more expressive than the former. Different definitions for simulations are used in the literature on comparing access control models. We discuss related work on this problem by identifying three axes along which the definitions used in such work differ.

- The first axis is whether the simulation maps only the state, or also the state-change rule. The approach of Bertino et al. [56] is to map only the states of two access control models to a common language based on mathematical logic, and to compare the results to determine whether one model is at least as expressive as the other, or whether the two models are incomparable. Other work, such as [10, 17, 18, 53, 54]

however, require both the state and the state-change rule to be mapped under the simulation.

An advantage with an approach such as the one that is adopted by Bertino et al. [56] is that it captures "structural" differences in how the protection state is represented in a system based on an access control model. For instance, it is observed in [56] that the existence of an indirection (the notion of a role) between users and permissions in RBAC gives it more expressive power than an access matrix model. Such "structural" differences are not captured by our theory, or other approaches that consider both the state and the state-change rule.

We point out, however, that the state-change rule is an important component of an access control system, and therefore assert that a meaningful theory for expressive power must consider it as well. In fact, it is often the case that it is the state-change rule that endows considerable power to an access control system. Consider, for example, the access matrix schemes proposed by Graham and Denning [4] and by Harrison et al. [2]. In both schemes, the state is represented by an access matrix. However, the state-change rules are quite different: in the Graham-Denning scheme [4], there are only specific ways in which rights may be transferred, while in the HRU scheme [2], one may define arbitrary commands in a state-change rule. It has also been demonstrated [57] that safety is decidable in polynomial time in the Graham-Denning scheme, while it is known to be undecidable [2] in the HRU scheme. Such differences cannot be captured by an approach that does not consider both the state and the state-change rule.

- The second axis is whether a simulation is required to preserve safety properties. In the comparison of different schemes based on the access matrix model [10, 17, 18, 54], the preservation of safety properties is required. If a scheme A is simulated in a scheme B, then a system in scheme A reaches an unsafe state if and only if the image of the system under the simulation (which is a system in scheme B)

reaches an unsafe state. By an "unsafe state" we mean a state in which a particular unauthorized subject has a right that she must not possess.

On the other hand, the preservation of safety properties is not required in the simulations used for comparing MAC (Mandatary Access Control), DAC (Discretionary Access Control), and RBAC (Role-Based Access Control) [11, 55, 58]. Nor is it required in the simulations used for the comparison of Access Control Lists (ACL), Capabilities, and Trust Management (TM) systems [53]. In these comparisons, the requirement for a simulation of A in B is that it should be possible to use an implementation of the scheme B to implement the scheme A. We call this the *implementation paradigm* of simulations.

- The third axis is whether to restrict the number of state-transitions that the simulating scheme needs to make in order to simulate one state-transition in the scheme being simulated. [53] define the notions of strong and weak simulations. A strong simulation of A in B requires that B makes one state-transition when A makes one state-transition. A weak simulation requires that B makes a bounded (by a constant) number of state-transitions to simulate one state-transition in A. A main result in [53] is that a specific TM scheme considered there is more expressive than ACL because there exists no (strong or weak) simulation of the TM scheme in ACL. The proof is based on the observation that an unbounded (but still finite) number of state-transitions in ACL is required to simulate one state-transition in the TM scheme.

  On the other hand, an unbounded number of state-transitions is allowed by [18]. That work uses a simulation that involves an unbounded number of state-transitions to prove that ATAM (Augmented Typed Access Matrix) is equivalent in expressive power to TAM (Typed Access Matrix).

Existing work on comparing access control models and schemes has severe shortcomings. First, different definitions of simulations make it impossible to put different results and claims about expressive power of access control models into a single context. For example, the result that RBAC is at least as expressive as DAC [11, 58] is qualitatively

different from the result that TAM is at least as expressive as ATAM [18], as the former does not require the preservation of safety properties. These results are again qualitatively different from the result that ACL is less expressive than Trust Management [53], as the latter requires a bounded number of state-transitions in simulations.

Second, some definitions of simulations that are used in the literature are too weak to distinguish access control models from one another in a meaningful way. Sandhu et al. [11,55,58] argues that various forms of DAC (including ATAM, in which simple safety is undecidable) can be simulated in RBAC, using the notion of simulations derived from the implementation paradigm. However, no concrete properties are articulated for the notion of simulations used in such work. Thus, this notion of simulations is not useful in differentiating models based on expressive power.

Finally, the rationale for some choices made in existing definitions of simulations is often not clearly stated or justified. It is unclear why certain requirements are made or not made for simulations when comparing the expressive power of access control models. For instance, when a simulation involves an unbounded number of state-transitions, Ganta [54] considers this to be a “weak” simulation, while Chander et al. [53] does not consider this to be a simulation at all. Neither decision is justified in Ganta [54] and Chander et al. [53].

3 SAFETY ANALYSIS IN DISCRETIONARY ACCESS CONTROL

In this chapter, we consider safety analysis in DAC. We reproduce the characterization of DAC from [16], and argue that the Graham-Denning DAC scheme [4] subsumes all known DAC schemes from the standpoint of safety analysis. We present the new result that safety is efficiently decidable in the Graham-Denning scheme [4] and thereby counter the assertion in the literature that safety is undecidable in DAC [23, 59]. We then assess a mapping from the literature that is used to argue that a new DAC scheme proposed by Solworth and Sloan [23] captures all known DAC schemes. Apart from presenting new results, this chapter provides additional background and motivation for our theory that we introduce in Chapter 5.

3.1 Characterizing DAC and safety analysis

The NCSC guide titled 'A Guide To Understanding Discretionary Access Control in Trusted Systems' [16], portions of which were published as a research paper [21], states that "the basis for (DAC) is that an individual user, or program operating on the user's behalf, is allowed to specify explicitly the types of access other users (or programs executing on their behalf) may have to information under the user's control." We point out two specific properties from this characterization of DAC: (1) The notion of "control" – there is a notion that users exercise control over resources in that a user that controls a resource gets to dictate the sorts of rights other users have over the resource, and (2) the notion of initiation of an action by a user to change the protection state – such state changes occur because particular users initiate such changes. A representation of a DAC scheme needs to capture both these properties.

Some literature (for example, [59,60]) appears to equate DAC with the HRU scheme [2]. This is incorrect, as there exist systems based on the HRU scheme that are not DAC sys-

tems. For instance, consider an HRU system in which there is only one command, and that command has no condition. This system is not a DAC system as it does not have the first property from above on the control of resources by a subject. In addition, there are DAC schemes that do not have natural representations as HRU schemes. For instance, the Graham-Denning scheme [4] (see Section 3.2.1) is a DAC scheme in which a subject may be 'owned' or 'controlled' by at most one other subject. A system based on the HRU scheme cannot capture this feature in a natural way.

Definition 3.1.1 (Safety Analysis) Given a DAC scheme $\langle \Gamma, \Psi \rangle$, let the set of subjects that can exist in a system based on the scheme be $\mathcal{S}$, let the set of objects be $\mathcal{O}$, and let the set of rights be $\mathcal{R}$. Assume that there exists a function $\mathsf{hasRight}\colon \mathcal{S} \times \mathcal{O} \times \mathcal{R} \rightarrow \{true, false\}$ such that $\mathsf{hasRight}(s, o, r)$ returns $true$ if in the current state, s and o exist, r is a right in the system, and s has the right r over o, and false otherwise. A safety analysis instance has the form $\langle \gamma, \psi, \mathcal{T}, \Box\neg\mathsf{hasRight}(s, o, r) \rangle$ for some $\gamma \in \Gamma$, $\psi \in \Psi$, $\mathcal{T} \subset \mathcal{S}$, $s \in \mathcal{S}$, $o \in \mathcal{O}$ and $r \in \mathcal{R}$. The safety analysis instance is true if $\mathsf{hasRight}(s, o, r)$ is false in every reachable state, with no state change initiated by a user from $\mathcal{T}$, and false otherwise.

In the above definition, $\Box$ stands for "in the current and all future states," and is an operator from temporal logic [61].

Each instance of the analysis is associated with a set $\mathcal{T}$ of trusted subjects. The meaning of a trusted subject is that we preclude state-changes initiated by any subject from $\mathcal{T}$ in our analysis. The intuition is that we expect these subjects to be "well-behaved". That is, while such subjects may effect state-changes, they do so in such a way that the state that results from the state-changes they effect satisfies desirable properties (that is, safety). Harrison et al. [2] does consider trusted subjects as part of their safety analysis. However, as pointed out by Li et al. [5], the way [2] deals with trusted subjects is incorrect. Harrison et al. [2] requires that we delete the rows and columns corresponding to trusted subjects prior to the analysis. While a trusted subject is not allowed to initiate a state-change, she may be used as an intermediary, and the way Harrison et al. [2] deals with trusted subjects does not consider this possibility. We require only that a member of the set of trusted

subjects not initiate a state-change. In all other ways, these subjects continue to be part of the system.

3.2 Safety analysis in the Graham-Denning scheme

In this section, we study safety analysis in the Graham-Denning DAC scheme [4]. We first present a description of the scheme in the following section. Our description clearly describes the states and state-change rules in the scheme. In Section 3.2.2, we present an algorithm to decide safety in the scheme, and show that the algorithm is correct. We assert also that the algorithm is efficient.

3.2.1 The Graham-Denning Scheme

In this section, We present a precise representation for the Graham-Denning scheme. We define what data are stored in a protection state, and how a state-change rule changes a state.

Assumptions We postulate the existence of the following countably infinite sets: $\mathcal{O}$, the set of objects; $\mathcal{S}$, the set of subjects ($\mathcal{S} \subset \mathcal{O}$); and $\mathcal{R}$, the set of rights.

Note that the set of objects (or subjects) in any given state in the Graham-Denning scheme is finite; however, the number of objects that could be added in some future state is unbounded. Similarly, the set of rights in any given access control system is finite; however, different access control systems may use different sets of rights. Therefore, we assume $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{R}$ are countably infinite.

We assume a naming convention so that we can determine, in constant time, whether a given object, o, is a subject (i.e., $o \in \mathcal{S}$) or not (i.e., $o \in \mathcal{O} - \mathcal{S}$). There exists a special "universal subject" $\mathsf{u} \in \mathcal{S}$; the role of u will be explained later. The set of rights $\mathcal{R}$ contains two special rights, own and $control$, a countably infinite set $\mathcal{R}_b$ of "basic" rights, and a countably infinite set $\mathcal{R}_b^*$ of basic rights with the copy flag denoted by $*$, i.e., $\mathcal{R}_b^* = \{r^* | r \in \mathcal{R}_b\}$. In other words, $\mathcal{R} = \{own, control\} \cup \mathcal{R}_b \cup \mathcal{R}_b^*$. The meaning of the

copy flag is clarified when we discuss the state-change rules for the scheme. An access control system based on the Graham-Denning scheme is associated with a protection state, and a state-change rule.

States, Γ A state in the Graham-Denning scheme, γ, is associated with the tuple $\langle O_\gamma, S_\gamma, M_\gamma[\,]\rangle$, where $O_\gamma \subset \mathcal{O}$ is a finite set of objects that exist in the state γ, $S_\gamma \subset \mathcal{S}$ is a finite set of subjects that exist in γ, and S_γ is a subset of O_γ. $M_\gamma[\,]$ is the access matrix, and $M_\gamma[\,]$: $S_\gamma \times O_\gamma \rightarrow 2^{\mathcal{R}}$. That is, $M_\gamma[s, o] \subset \mathcal{R}$ is the finite set of rights the subject $s \in S_\gamma$ has over the object $o \in O_\gamma$.

Every state, $\gamma = \langle O_\gamma, S_\gamma, M_\gamma[\,]\rangle$, in the Graham-Denning scheme satisfies the following seven properties.

1. Every object must be owned by at least one subject, i.e., $\forall\, o \in O_\gamma\, \exists s \in S_\gamma(own \in M_\gamma[s, o])$.

2. Objects are not controlled, only subjects are, i.e., $\forall\, o \in (O_\gamma - S_\gamma)\, \forall\, s \in S_\gamma(control \notin M_\gamma[s, o])$.

3. The special subject u exists in the state, is not owned by any subject, and is not controlled by any other subject, i.e., $\mathsf{u} \in S_\gamma \wedge \forall\, s \in S_\gamma(own \notin M_\gamma[s, \mathsf{u}]) \wedge \forall\, s \in S_\gamma - \{\mathsf{u}\}(control \notin M_\gamma[s, \mathsf{u}])$.

4. A subject other than u is owned by exactly one other subject, i.e., for every $s \in S_\gamma - \{\mathsf{u}\}$, there exists exactly one $s' \in S_\gamma$ such that $s' \neq s$ and $own \in M_\gamma[s', s]$;

5. Every subject controls itself, i.e., $\forall s \in S_\gamma(control \in M_\gamma[s, s])$.

6. A subject other than u is controlled by at most one other subject, i.e., for every $s \in S_\gamma - \{\mathsf{u}\}$, there exists at most one $s' \in S_\gamma$ such that $s' \neq s$ and $control \in M_\gamma[s', s]$.

7. There exists no set of subjects such that they form a "cycle" in terms of ownership of each other (and in particular, a subject does not own itself), i.e., $\neg(\exists\, \{s_1, \ldots, s_n\} \subseteq S_\gamma(own \in M_\gamma[s_2, s_1] \wedge own \in M_\gamma[s_3, s_2] \wedge \cdots \wedge own \in M_\gamma[s_n, s_{n-1}] \wedge own \in M_\gamma[s_1, s_n]))$.

These state invariants are maintained by the state-change rules.

State-Change Rules, Ψ Each member, ψ, of the set of state-change rules, Ψ, in the Graham-Denning scheme, is a set of commands parameterized by a set of rights, R_ψ. These commands are shown in Figure 3.1. Where possible, we use the syntax for commands from the HRU scheme [2], but as we mention in Section 3.1, we cannot represent all aspects of DAC schemes using only constructs for commands in the HRU scheme. We use some additional well-known constructs such as $\forall$ and $\exists$ in these commands. A state-change is the successful execution of one of the commands. We assume that the state subsequent to the execution of a command is γ'. We denote such a state-change as $\gamma \mapsto_{\psi(s)} \gamma'$, where s is the initiator of the command. When the rule and initiator are implied or not important, we write simply $\gamma \mapsto \gamma'$, and to denote zero or more state-changes, we write $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$. We point out that for each command, unless specified otherwise, $S_{\gamma'} = S_\gamma$, $O_{\gamma'} = O_\gamma$, and $M_{\gamma'}[s, o] = M_\gamma[s, o]$ for every $s \in S_\gamma$ and $o \in O_\gamma$. We use $\leftarrow$ to denote assignment, i.e., $x \leftarrow y$ means that the value in x is replaced with the value in y. The commands in the Graham-Denning scheme are the following. The first parameter to each command is named i, and is the subject that is the initiator of the execution of the command.

- transfer_r(i, s, o) This command is used to grant the right r by an initiator that has the right r^* over o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. The initiator, i, must possess the right r^* over o, and the subject s must exist for this command execution to succeed.

- transfer_r$^*(i, s, o)$ This command is used to grant the right r^* by an initiator that has the right r^* over o. There is one such command for every $r^* \in R_\psi \cap \mathcal{R}_b^*$. The initiator, i, must possess the right r^* over o, and the subject s must exist for this command execution to succeed.

- transfer_own(i, s, o) This command is used to transfer ownership over o from i to s. For this command to succeed, i must have the own right over o, s must exist, and the

transfer of ownership must not violate invariant (7) from the list of state invariants we discuss above. After the execution of this command, i will no longer have the own right over o (but s will).

- grant_r(i, s, o) This command is used to grant the right r over o by the owner of o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. For this command execution to succeed, i must have the own right over o, and s must exist.

- grant_r$^*(i, s, o)$ This command is very similar to the previous command, except the the owner grants $r^* \in R_\psi \cap \mathcal{R}_b^*$.

- grant_control(i, s, o) This command is used to grant the $control$ right over o by its owner. For the execution of this command to succeed, i must have the right $control$ over o, s must exist, o must be a subject, and another subject must not already have the right $control$ over o. These checks are needed to maintain the state invariants related to the $control$ right that we discuss above.

- grant_own(i, s, o) This command is used to grant the own right over o. This is different from the transfer_own command in that in this case, i retains (joint) ownership over o. For the execution of this command to succeed, i must have the right own over o, o must not be a subject, and s must exist.

- delete_r(i, s, o) This command is used to delete a right a subject has over o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. For the execution of this command to succeed, i must have the right own over o, and s must exist.

- delete_r$^*(i, s, o)$ This command is similar to the previous command, except that a right $r^* \in R_\psi \cap \mathcal{R}_b^*$ is deleted.

- create_object(i, o) This command is used to create an object that is not a subject. For the execution of this command to succeed, i must exist, and o must be an object that is not a subject, that does not exist. An effect of this command is that i gets the own right over o in the new state.

- destroy_object(i, o) This command is used to destroy an object that exists. For the execution of this command to succeed, i must have the right own over o, and o must be an object that is not a subject.

- create_subject(i, s) This command is used to create a subject. For the execution of this command to succeed, i must exist, and s must be a subject that does not exist. In the new state, i has the own right over s, and s has the $control$ right over itself.

- destroy_subject(i, s) This command is used to destroy a subject. For the execution of this command to succeed, i must have the own right over s. An effect of this command is that ownership over any object owned by s is transferred to i.

3.2.2 Safety analysis

An algorithm to decide whether a system based on the Graham-Denning scheme is safe is shown in Figure 3.2. A system based on the Graham-Denning scheme is characterized by a start-state, γ, and state-change rule, ψ (which is a set of commands). The algorithm takes as input γ, ψ, a triple, $\omega = \langle s, o, x \rangle \in \mathcal{S} \times \mathcal{O} \times \mathcal{R}$, and a finite set, $\mathcal{T} \subset \mathcal{S}$, of trusted subjects. The algorithm outputs "true" if the system satisfies the safety property with respect to the subject s, object o and right x, and "false" otherwise. We first discuss the algorithm, and then its correctness and time-complexity.

In lines 5-10 of the algorithm, we check the cases for which we do not have to consider potential state-changes before we are able to decide whether the system is safe or not. In lines 5-6, we consider the case that a subject may have (or acquire) the right with the copy flag. For this, we need to exclude own and $control$ from consideration, as those rights do not have counterparts with the copy flag. We use the mnemonic $invalid$ to indicate this. In line 7, we check that the right x is indeed in the system. In line 8, we check whether we are being asked whether s can get the $control$ right over o, where o is an object that is not a subject (we know s does not have and cannot get the right, by property (2) of the seven properties we discuss in the previous section). In line 9, we check whether the

command transfer_r(i, s, o)
  if $r^* \in M_\gamma[i, o] \wedge s \in S_\gamma$ *then*
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r\}$

command transfer_r$^*(i, s, o)$
  if $r^* \in M_\gamma[i, o] \wedge s \in S_\gamma$ *then*
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r^*\}$

command transfer_own(i, s, o)
  if $own \in M_\gamma[i, o] \wedge o \in S_\gamma \wedge s \in S_\gamma$ *then*
    if $\nexists \{s_1, \ldots, s_n\} \in S_\gamma$ *such that*
    $own \in M_\gamma[s_1, s] \wedge own \in M_\gamma[s_2, s_1]$
    $\wedge \cdots \wedge own \in M_\gamma[s_n, s_{n-1}]$
    $\wedge own \in M_\gamma[o, s_n]$ *then*
      $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{own\}$
      $M_{\gamma'}[i, o] \leftarrow M_\gamma[i, o] - \{own\}$

command grant_r(i, s, o)
  if $own \in M_\gamma[i, o] \wedge s \in S_\gamma$ *then*
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r\}$

command grant_r$^*(i, s, o)$
  if $own \in M_\gamma[i, o] \wedge s \in S_\gamma$ *then*
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r^*\}$

command grant_control(i, s, o)
  if $own \in M_\gamma[i, o] \wedge o \in S_\gamma \wedge s \in S_\gamma$ *then*
    if $\nexists\, s' \in S_\gamma$ *such that*
    $s' \neq o \wedge control \in M_\gamma[s', o]$ *then*
      $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{control\}$

command grant_own(i, s, o)
  if $own \in M_\gamma[i, o] \wedge o \notin S_\gamma$
  $\wedge\, s \in S_\gamma$ *then*
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{own\}$

command delete_r(i, s, o)
  if $(own \in M_\gamma[i, o] \wedge s \in S_\gamma)$
  $\vee\, control \in M_\gamma[i, s]$ *then*
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] - \{r\}$

command delete_r$^*(i, s, o)$
  if $(own \in M_\gamma[i, o] \wedge s \in S_\gamma)$
  $\vee\, control \in M_\gamma[i, s]$ *then*
    $M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] - \{r^*\}$

command create_object(i, o)
  if $o \notin O_\gamma \wedge i \in S_\gamma \wedge o \in \mathcal{O} - \mathcal{S}$ *then*
    $O_{\gamma'} \leftarrow O_\gamma \cup \{o\}$
    $M_{\gamma'}[i, o] \leftarrow own$

command destroy_object(i, o)
  if $own \in M_\gamma[i, o] \wedge o \notin S_\gamma$ *then*
    $O_{\gamma'} \leftarrow O_\gamma - \{o\}$

command create_subject(i, s)
  if $s \notin O_\gamma \wedge i \in S_\gamma \wedge s \in \mathcal{S}$ *then*
    $O_{\gamma'} \leftarrow O_\gamma \cup \{s\}$
    $S_{\gamma'} \leftarrow S_\gamma \cup \{s\}$
    $M_{\gamma'}[i, s] \leftarrow \{own\}$
    $M_{\gamma'}[s, s] \leftarrow \{control\}$

command destroy_subject(i, s)
  if $own \in M_\gamma[i, s] \wedge s \in S_\gamma$ *then*
    $\forall\, o \in O_\gamma$, *if* $own \in M_\gamma[s, o]$ *then*
      $M_{\gamma'}[i, o] \leftarrow M_\gamma[i, o] \cup \{own\}$
    $O_{\gamma'} \leftarrow O_\gamma - \{s\}$
    $S_{\gamma'} \leftarrow S_\gamma - \{s\}$

Figure 3.1. The set of commands that constitutes the state-change rule, ψ, for a system based on the Graham-Denning scheme. Each command has a name (e.g., transfer_own), and a sequence of parameters. The first parameter is always named i, and is the initiator of the command, i.e., the subject that executes the command. There is one transfer_r, grant_r, and delete_r command for each $r \in R_\psi \cap \mathcal{R}_b$, and one transfer_r*, grant_r*, and delete_r* command for each $r^* \in R_\psi \cap \mathcal{R}_b^*$.

```
 1 Subroutine isSafeGD(γ, ψ, ω, 𝒯)
 2   /* inputs: γ, ψ, ω = ⟨s, o, x⟩, 𝒯 ⊆ 𝒮 */
 3   /* output: true or false */
 4   if x ∈ ℛ_b^* then let y ← x
 5   else if x ≠ own ∧ x ≠ control then let y ← x*
 6   else let y ← invalid /* No copy flags for own or control */
 7   if x ∉ R_ψ then return true
 8   if x = control ∧ o ∈ 𝒪 − 𝒮 then return true
 9   if x ∈ M_γ[s, o] then return false
10   if y ∈ M_γ[s, o] then return false
11   if 𝒯 ⊇ S_γ then return true
12   if o ∉ O_γ then return false
13   if ∃ŝ ∈ S_γ − 𝒯 such that y ∈ M_γ[ŝ, o] then return false
14   for each sequence 𝒰, s_n, ..., s_2, s_1 such that
15   own ∈ M_γ[s_1, o] ∧ ··· ∧ own ∈ M_γ[s_n, s_{n−1}] ∧ own ∈ M_γ[u, s_n] do
16     if ∃s_i ∈ {s_1, ..., s_n} such that s_i ∈ S_γ − 𝒯 then return false
17   return true
```

Figure 3.2. The subroutine isSafeGD returns "true" if the system based on the Graham-Denning scheme, characterized by the start-state, γ, and state-change rule, ψ, satisfies the safety property with respect to ω and $\mathcal{T}$. Otherwise, it returns "false". In line 6, we assign some invalid value to y, as there is not corresponding right with the copy flag for the rights own and $control$. In this case, the algorithm will not return in line 10 or 13. The subject u appears in line 15 only to emphasize that the "chain" of ownership is terminal.

right x has already been acquired by s over o. In line 10, we check that if the right y has already been acquired by s over o (the check in line 10 is needed when $x \in \mathcal{R}_b$, as then, the possession of x^* implies the possession of x; in the case that $x \in \mathcal{R}_b^*$, the lines 9 and 10 are identical). When $x = own$ or $x = control$, the condition of line 10 will never be true, and we will not return from that line. In the remainder of the algorithm, we consider those cases in which a state-change is needed before s can get x over o (if it can at all). In line 11, we check whether there is at least one subject that can initiate state-changes, and if not, we know that the system is safe. In line 12, we check whether o exists, and if it does not, given that there exists a subject that can create o (from our check in line 11), the subject can then grant x to s over o. In line 13, we check whether there is a subject that can initiate state-changes, and that has x with the copy-flag (or x itself, if $x \in \mathcal{R}_b^*$). If $x = own$ or $x = control$, the condition of line 13 cannot be true. In lines 14-16, we check whether there is a sequence of subjects with the particular property that each owns the next in the sequence, and the last subject in the sequence owns o. If any one of those subjects can initiate state-changes, then we conclude that the system is not safe and return false. In all other cases, we conclude that the system is safe, and return true.

The following lemma asserts that the algorithm is correct. Theorem 3.2.2 summarizes our results with respect to safety analysis in the Graham-Denning scheme.

Lemma 3.2.1 *A system based on the Graham-Denning scheme, that is characterized by the start-state, γ, and state-change rule, ψ, is safe with respect to $\omega = \langle s, o, x \rangle$ and $\mathcal{T} \subset \mathcal{S}$ (where $\mathcal{T}$ is finite) if and only if* $\mathsf{isSafeGD}(\gamma, \psi, \omega, \mathcal{T})$ *returns true.*

Proof The "if" part: we need to show that if $\mathsf{isSafeGD}(\gamma, \psi, \omega, \mathcal{T})$ returns true, then the system is safe with respect to ω and $\mathcal{T}$. We show, equivalently, that if the system is not safe with respect to ω and $\mathcal{T}$, then $\mathsf{isSafeGD}(\gamma, \psi, \omega, \mathcal{T})$ returns false. Assume that the system is not safe with respect to ω and $\mathcal{T}$. We have two cases. The first case is that in the start-state, γ, s has x over o. This case consists of two subcases: either (1) $x \in M_\gamma[s, o]$, or (2) $x \in \mathcal{R}_b$ and $x^* \in M_\gamma[s, o]$ (possession of x^* implies possession of x). If both (1) and (2) are true, we consider either one of those two subcases. If subcase (1) is true, then we

know that $x \in R_\psi$, and if $x = control$ and $o \in \mathcal{O} - \mathcal{S}$, then $x \notin M_\gamma[s, o]$ (by property (2) from the previous section that objects that are not subjects cannot have the $control$ right over them). Therefore, the 'if' conditions of lines 7 and 8 are not satisfied, and line 9 of the algorithm returns false, and we are done. For subcase (2), in line 5 we instantiate y to x^*. We know that $x, y \in R_\psi$, and that $x \neq control$. Therefore, the 'if' conditions for lines 7 and 8 are not satisfied. The 'if' condition for line 9 may be satisfied and if it is, the algorithm returns false and we are done. Otherwise, the algorithm returns false in line 10.

The second case is that s does not have x over o in the start-state, i.e., $x \notin M_\gamma[s, o]$ and if $x \in \mathcal{R}_b$, then $x^* \notin M_\gamma[s, o]$. In this case, as the system is not safe, there exists a finite sequence of state-changes $\gamma \mapsto_{\psi(s_1)} \gamma_1 \mapsto_{\psi(s_2)} \cdots \mapsto_{\psi(s_n)} \gamma_n$ where n is an integer and $n \geq 1$, such that either $x \in M_{\gamma_n}[s, o]$, or if $x \in \mathcal{R}_b$, then $x^* \in M_{\gamma_n}[s, o]$. Each $s_i \in S_{\gamma_{i-1}} - \mathcal{T}$ and the s_i's are not necessarily distinct from one another. We point out also that if $s_i \in S_{\gamma_j} - \mathcal{T}$ for some i and j, and $s_i \in S_{\gamma_k}$ for some $k \neq j$, then $s_i \in S_{\gamma_k} - \mathcal{T}$, because $\mathcal{T}$ is specified a-priori and does not change with changes in the state. We now show that if such a sequence of state-changes exists, then the algorithm returns false. We show this by induction on n. For the base case, if there exists a sequence of length 1, then $\gamma \mapsto_{\psi(s_1)} \gamma_1$, and $x \notin M_\gamma[s, o]$ and $x^* \notin M_\gamma[s, o]$ if $x \in \mathcal{R}_b$, and $x \in M_{\gamma_1}[s, o]$, or $x \in \mathcal{R}_b$ and $x^* \in M_{\gamma_1}[s, o]$. In this case, the state-change is the execution of one of the following commands, and we show that the algorithm returns false in each case. The state-change has to be the execution of one of these commands because these are the only commands that enter a right in to a cell of the access matrix.

transfer_r – in this case we know that $x \in \mathcal{R}_b \cap R_\psi$, $x^* \in R_\psi$, $x^* \in M_\gamma[s_1, o]$ for some $s_1 \in S_\gamma - \mathcal{T}$, and $s \in S_\gamma$. The algorithm will not return in any of the lines 7-11 as the respective 'if' conditions are not satisfied. If $o \notin O_\gamma$, then the algorithm returns false in line 12, and we are done. If $o \in O_\gamma$, then the conditions for line 13 are met (y is instantiated to x^*), and the algorithm returns false.

transfer_r* – we have two subcases to consider: either (1) $x \in \mathcal{R}_b^* \cap R_\psi$, or, (2) $x \in \mathcal{R}_b \cap R_\psi$. In case (2), let y be x^*, and in case (1), let y be x. We know in either

case that $y \in M_\gamma[s_1, o]$ for some $s_1 \in S_\gamma - \mathcal{T}$, and $s \in S_\gamma$ (otherwise s would not get the right x over o after the execution of the command). The algorithm will not return in any of the lines 7-11 as the respective 'if' conditions are not satisfied. If $o \notin O_\gamma$, then the algorithm returns false in line 12, and we are done. If $o \in O_\gamma$, then the conditions for line 13 are met and the algorithm returns false.

transfer_own – in this case we know that $x = own$, $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$, $o \in S_\gamma$ and $s \in S_\gamma$. The 'if' conditions for each of lines 7-13 are not met (for line 11, we know that $own^* \notin R_\psi$). Consider lines 14-16. We know that such a sequence of subjects exists (as i has the own right over o in S_γ), and furthermore, $i \in S_\gamma - \mathcal{T}$. Therefore, the conditions to return false in lines 14-16 are met, and the algorithm returns false.

grant_r – in this case, we know that $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$ and $x \in \mathcal{R}_b \cap R_\psi$ (in particular, $x \neq control$ and $x \neq own$ – there are other commands to grant those rights). The 'if' conditions for each of lines 7-11 are not met. If $o \notin O_\gamma$, the algorithm returns false in line 12, and we are done. If $o \in O_\gamma$, the conditions for line 13 may be met, and if they are, the algorithm returns false and we are done. If the conditions in line 13 are not met, then we observe that the conditions for lines 14-16 are met (the sequence of subjects contains i, as i has the own right over o in S_γ), and the algorithm returns false.

grant_r* – we have two subcases to consider. Either (1) $x \in \mathcal{R}_b \cap R_\psi$, or, (2) $x \in \mathcal{R}_b^* \cap R_\psi$. For case (1), let y be x^* and for case (2), let y be x. In either case, we know that $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$. The 'if' conditions for lines 7-11 are not met. If $o \notin O_\gamma$, then the algorithm returns false in line 12, and we are done. Otherwise, the conditions for line 13 may be met, and if they are, the algorithm returns false, and we are done. Otherwise, we observe that the conditions for lines 14-16 are met (the sequence of subjects contains i, as i has the own right over o in S_γ), and the algorithm returns false.

grant_control – in this case, we know that $x = own$, $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$ and $o \in S_\gamma$. Therefore, the 'if' conditions for lines 7-12 are not met. The 'if' conditions for line 13 are not met because we know that $y \notin R_\psi$. But, we observe that the conditions for lines 14-16 are met, because the subject i that is not trusted exists in γ, and i has the own right over o. Therefore, the algorithm returns false in line 16.

grant_own – in this case, we know that $x = own$ and $own \in M_\gamma[i, o]$ for some $i \in S_\gamma - \mathcal{T}$. The 'if' conditions for lines 7-11 are not satisfied. If $o \notin O_\gamma$, then the algorithm returns false in line 12 and we are done. Otherwise, the condition in line 13 is not satisfied, but, we observe that the conditions for lines 14-16 are satisfied, and the algorithm returns false.

create_object – in this case, we know that $x = own$ and $o \notin O_\gamma$. The 'if' conditions for lines 7-11 are not met, but the 'if' condition for line 12 is met, and the algorithm returns false.

create_subject – in this case, we know that $\exists i \in S_\gamma - \mathcal{T}$, and either $x = own$ or $x = control$. Furthermore, we know that $o \notin O_\gamma$. The reason is that in the body of the command, we enter a right only in the column corresponding to the subject that is created in the execution of the command, and not any other object. Therefore, for $\omega = \langle s, o, x \rangle$, we know that o must be the subject that is created in the execution of the create_subject command. We know also that $o \notin \mathcal{O} - \mathcal{S}$, because the object that is created is a subject. Therefore, the respective 'if' conditions for lines 7-11 are not satisfied, but the 'if' condition for line 12 is satisfied, and the algorithm returns false.

destroy_subject – in this case, we know that $x = own$, and $own \in M_\gamma[s, s']$, where $\omega = \langle s, o, x \rangle$ and s' is the subject that is destroyed in the execution of the command. The reason is that we enter a right only in the row corresponding to such a subject s. Furthermore, we know that $o \in O_\gamma$ and $own \in M_\gamma[s', o]$, because the only columns

in which a right is entered in the execution of the command are columns with that property. We know also that $s \in S_\gamma - \mathcal{T}$ as s is the initiator of the command-execution. Given these facts, we know that the 'if' conditions for lines 7-12 are not satisfied. The conditions for line 13 may be met, and if they are, the algorithm returns false and we are done. Otherwise, we observe that the conditions for lines 14-16 are satisfied; the sequence of subjects contains s and s' with s' being the last member of the sequence, and s immediately preceding s' in the sequence. As $s \in S_\gamma - \mathcal{T}$, the algorithm returns false in line 16.

For the induction hypothesis, we assume that if there exists a state-change sequence $\gamma \vdash_{\psi(s_1)} \gamma_1 \vdash_{\psi(s_2)} \cdots \vdash_{\psi(s_{k-1})} \gamma_{k-1}$ of length $k-1$ (for $k-1 \geq 1$) such that $x \notin M_\gamma[s,o]$ and if $x \in \mathcal{R}_b$, $x^* \notin M_\gamma[s,o]$, and either $x \in M_{\gamma_{k-1}}[s,o]$ or, if $x \in \mathcal{R}_b$, $x^* \in M_{\gamma_{k-1}}[s,o]$, then the algorithm returns false. Now assume that there exists a state-change sequence $\gamma \vdash_{\psi(s_1)} \cdots \vdash_{\psi(s_k)} \gamma_k$ of length k (for $k \geq 2$) such that $x \notin M_\gamma[s,o]$ and if $x \in \mathcal{R}_b$, $x^* \notin M_\gamma[s,o]$, and either $x \in M_{\gamma_k}[s,o]$ or, if $x \in \mathcal{R}_b$, $x^* \in M_{\gamma_k}[s,o]$. We need to show that the algorithm returns false for $\omega = \langle s,o,x \rangle$.

We have two cases. The first case has two subcases: either (a) $x \in M_{\gamma_{k-1}}[s,o]$, or, (b) $x \in \mathcal{R}_b$ and $x^* \in M_{\gamma_{k-1}}[s,o]$. In either case, we have a state-change sequence of length $k-1$ with the appropriate properties, and by the induction hypothesis, we know that the algorithm returns false. In the second case, we assume that $x \notin M_{\gamma_{k-1}}[s,o]$ and if $x \in \mathcal{R}_b$, $x^* \notin M_{\gamma_{k-1}}[s,o]$, and either $x \in M_{\gamma_k}[s,o]$ or $x \in \mathcal{R}_b$ and $x^* \in M_{\gamma_k}[s,o]$. We need to show that the algorithm returns false in this case. We consider the state-change $\gamma_{k-1} \vdash_{\psi(s_k)} \gamma_k$. It must be the execution of one of the following commands (the same as those we considered for the base case), as those are the only commands that add a right to a cell in the access matrix. We consider each in turn. We point out that as $k \geq 2$, we have at least 3 states in our state-change sequence, including the start-state, i.e., we know that at least the states $\gamma_{k-2}, \gamma_{k-1}$ and γ_k (where the start-state, $\gamma = \gamma_0$) exist in the state-change sequence.

transfer_r – in this case, we know that $x \in \mathcal{R}_b \cap R_\psi$ and $x^* \in M_{\gamma_{k-1}}[s_k,o]$. Let $\omega^k = \langle s_k,o,x^* \rangle$. Then, we know by the induction hypothesis that isSafeGD$(\gamma,\psi,\omega^k,\mathcal{T})$

returns false (as there exists a state-change sequence of length $k-1$ with the appropriate properties). We refer to the execution of the algorithm for the input $(\gamma, \omega, \mathcal{T})$ as e, and for the input $(\gamma, \omega^k, \mathcal{T})$ as e^k. Consider the following cases.

- e^k returns in line 9: in this case, we know that $x^* \in M_\gamma[s_k, o]$. Now, e cannot return in lines 7 or 8 (because $x \in \mathcal{R}_b \cap R_\psi$). e may return false in line 9 or line 10, in which case we are done. If not, e will not return in lines 11-12 as $s_k \in S_\gamma - \mathcal{T}$ and $o \in O_\gamma$. Finally, e will return false in line 13, because $s_k \in S_\gamma - \mathcal{T}$, and $y \in M_\gamma[s_k, o]$.
- e^k returns in line 10: this cannot happen as, in this case, e^k would have returned in line 9. Therefore, the arguments for the previous case apply.
- e^k returns in line 12: in this case, e will not return in any of the lines 7-11, but will return false in line 12.
- e^k returns in line 13: in this case, we know that $\exists\ \widehat{s} \in S_\gamma - \mathcal{T}$ such that $y \in M_\gamma[\widehat{s}, o]$ where $y = x^*$. e will not return in lines 7-8, but may return false in one of the lines 9 or 10, in which case we are done. Otherwise, e will not return in line 11 (as $\widehat{S}$ exists in γ) or in line 12 ($o \in O_\gamma$). But, e will return false in line 13, as the condition is met ($\widehat{S}$ is such a subject).
- e^k returns in line 16: in this case, e will not return in lines 7-8 but may return in line 9, in which case we are done. Otherwise, e will not return in lines 10-13. We know that e will return false in line 16, just as e^k does, because the same condition is true for e as well.

transfer_r* – in this case, we know that $x \in \mathcal{R}_b^* \cap R_\psi$, and $x \in M_{\gamma_{k-1}}[s_k, o]$ where $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. Let $\omega^k = \langle s_k, o, x \rangle$, e^k be the execution of the algorithm isSafeGD for the input $(\gamma, \psi, \omega^k, \mathcal{T})$, and e be the execution for the input $(\gamma, \omega, \mathcal{T})$. Then we know that e^k returns false by the induction hypothesis. We now have exactly the same arguments as in the previous case for why e returns false.

transfer_own – in this case we know that $x = own$ and $own \in M_{\gamma_{k-1}}[s_k, o]$ where $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. For $\omega^k = \langle s_k, o, own \rangle$, we know that, e^k, the execution of the algorithm on input $(\gamma, \omega^k, \mathcal{T})$, returns false, by the induction hypothesis. We consider all the cases in which e^k can return false.

- e^k returns in line 9: in this case, we know that $own \in M_\gamma[s_k, o]$ and $s_k \in S_\gamma - \mathcal{T}$. Now, e does not return in any of the lines 7-8. e may return in line 9, in which case we are done. e cannot return in line 10 (as $y \notin R_\psi$), or in line 11, but may return in line 12, in which case we are done. e cannot return in line 13. Finally, we observe that the conditions in lines 14-16 are satisfied, and therefore, e returns in line 16.
- e^k returns in line 10: this cannot happen because when $x = own$, $y \notin R_\psi$.
- e^k returns in line 12: in this case, we know that e does not return in lines 7-11, but returns false in line 12.
- e^k returns in line 13: this cannot happen because when $x = own$, $y \notin R_\psi$.
- e^k returns in line 16: in this case, e does not return in lines 7-8, but may return in line 9, in which case we are done. Otherwise, e cannot return in lines 10-13, but returns false in line 16 based on the same conditions that e^k satisfies to return in line 16.

grant_r – in this case, we know that $x \in \mathcal{R}_b \cap R_\psi$ and $own \in M_{\gamma_{k-1}}[s_k, o]$, where $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. We know also that e^k, the execution of the algorithm, on input $(\gamma, \omega^k, \mathcal{T})$ returns false, where ω^k $tuples_k, o, own$. Let e be the execution of the algorithm for the input $(\gamma, \omega, \mathcal{T})$. We have the following cases.

- e^k returns in line 9: in this case, we know also that $own \in M_\gamma[s_k, o]$ where $s_k \in S_\gamma - \mathcal{T}$. Therefore, e does not return in lines 7-8, but may return false in either line 9 or line 10, in which case we are done. Otherwise, e does not return in lines 11-12, but may return false in line 13, in which case we are

done. Finally, e returns false in line 16, because the conditions for returning in line 16 are satisfied (s_k is such a subject).

- e^k returns in line 10: this is not possible as when $x = own$, $y \notin R_\psi$.
- e^k returns in line 12: in this case, e does not return in lines 7-11, but returns false in line 12.
- e^k returns in line 13: this is not possible as when $x = own$, $y \notin R_\psi$.
- e^k returns in line 16: in this case, we know that e does not return in lines 7-8, but may return in one of the lines 9-10, in which case we are done. Otherwise, e does not return in lines 11-12, but may return in line 13, in which case we are done. Finally, e returns in line 16 as the conditions for which e^k returns in line 16 apply to e as well.

grant_r* – in this case, we know that $x \in \mathcal{R}_b^* \cap R_\psi$ and $own \in M_\gamma[s_k, o]$ for $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. The argument now proceeds exactly as for the previous case, and we are able to show that isSafeGDreturns false on the input $(\gamma, \psi, \omega, \mathcal{T})$.

grant_control – in this case, we know that $x = control$ and $own \in M_{\gamma_{k-1}}[s_k, o]$ for $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. Let $\omega^k = \langle s_k, o, own \rangle$, and e^k be the execution of the algorithm on the input $(\gamma, \omega^k, \mathcal{T})$. We know, by the induction hypothesis, that e^k returns false. Let e be the execution of the algorithm on the input $(\gamma, \omega, \mathcal{T})$. We have the following cases.

- e^k returns in line 9: in this case we know also that $own \in M_\gamma[s_k, o]$ and $s_k \in S_\gamma - \mathcal{T}$. Therefore, e does not return in lines 7-8 (for line 8, we know that $o \notin \mathcal{O} - \mathcal{S}$, as otherwise, we would not be able to grant the $control$ right to s over o in the final state-change in our sequence), and e may return false in line 9, in which case we are done. Otherwise, e does not return in lines 10-13 (for lines 10 and 13, $y \notin R_\psi$). Finally, e returns false in line 16 because we know that s_k, a subject that is not trusted, exists in γ, and has the own right over o.
- e^k returns in line 10: this is not possible as when $x = own$, $y \notin R_\psi$.

- e^k returns in line 12: in this case, e does not return in lines 7-11, but returns false in line 12.
- e^k returns in line 13: this is not possible as when $x = own$, $y \notin R_\psi$.
- e^k returns in line 16: in this case, e does not return in lines 7-8, but may return in lines 9-10, in which case we are done. Otherwise, e does not return in lines 11-12, but may return in line 13, in which case we are done. Finally, e returns in line 16 as the conditions for which e^k returns in line 16 apply to e as well.

grant_own – in this case, we know that $x = own$ and $own \in M_{\gamma_{k-1}}[s_k, o]$ for $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. We show that the execution of the algorithm on input $(\gamma, \omega, \mathcal{T})$ returns false using the same arguments as the ones we use for the previous case.

create_object – in this case, we know that $x = own$, $s = s_k$ and $s_k \in S_{\gamma_{k-1}} - \mathcal{T}$. We consider the following cases (and sub-cases).

- $s \in S_{\gamma_{k-2}}$: in this case we need to consider the following two sub-cases.
  - $o \in O_{\gamma_{k-2}}$: in this case, we know that the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma_{k-1}$ is destroy_object of object o by s_{k-1}. Therefore, we know that $own \in M_{\gamma_{k-2}}[s_{k-1}, o]$ and $s_{k-1} \in S_{\gamma_{k-2}} - \mathcal{T}$. If $s = s_{k-1}$, then we have a state-change sequence of length $k-2$ with the appropriate properties, and we know that the algorithm returns false. Otherwise, we have a state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma'_{k-1}$ which is the execution of either the command transfer_own (if $o \in \mathcal{S}$), or the command grant_own (if $o \in \mathcal{O} - \mathcal{S}$), by s_{k-1} to s, which results in $own \in M_{\gamma'_{k-1}}[s, o]$. As there exists a state-change sequence of length $k-1$, we know that the algorithm returns false by the induction hypothesis.
  - $o \notin O_{\gamma_{k-2}}$: in this case, there exists a state-change $\gamma_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$ which is the execution of the command create_object of o by s, which results in $own \in M_{\gamma'_{k-1}}[s, o]$. As there exists a state-change sequence of

length $k-1$, we know that the algorithm returns false by the induction hypothesis.

- $s \notin S_{\gamma_{k-2}}$: in this case, we know that the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma_{k-1}$ is the execution of create_subject to create s. Also, we know that $o \notin O_{\gamma_{k-2}}$. If $\gamma_{k-2} = \gamma$, then we know that, on input $(\gamma, \omega, \mathcal{T})$, the algorithm will not return in lines 7-11, but will return false in line 12, and we would be done in this case. Otherwise, there exists at least one prior state, γ_{k-3} in the sequence of state-changes. We have the following sub-cases.
  - $s \in S_{\gamma_{k-3}}$, but $o \notin O_{\gamma_{k-3}}$: in this case, we know that the state-change $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma_{k-2}$ is the execution of destroy_subject of s by s_{k-2}. Consider the alternate state-changes $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s_{k-2})} \gamma'_{k-1}$, where the first state-change is the execution of create_object of o by s_{k-2}, and the second is the execution of transfer_own (if $o \in \mathcal{S}$) or grant_own (if $o \in \mathcal{O} - \mathcal{S}$) of the object o by s_{k-2} to s. We have a desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.
  - $s \notin S_{\gamma_{k-3}}$, but $o \in O_{\gamma_{k-3}}$: in this case, we know that the state-change $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma_{k-2}$ is the execution of destroy_object of o by s_{k-2}. Consider instead the state-changes $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s_{k-2})} \gamma'_{k-1}$, where the first state-change is the execution of create_subject of s by s_{k-2} and the second is the execution of transfer_own (if $o \in \mathcal{S}$) or grant_own (if $o \in \mathcal{O} - \mathcal{S}$) of the object o to s by s_{k-2}. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.
  - $s \notin S_{\gamma_{k-3}}$, and $o \notin O_{\gamma_{k-3}}$: we know that $s \notin \mathcal{T}$ (otherwise s would not be able to execute create_object as the last state-change in our state-change sequence of length k). We know also that $s_{k-2} \in S_{\gamma_{k-3}} - \mathcal{T}$. Consider the following state-changes: $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(\psi(s)} \gamma'_{k-1}$ where the first state-change is the execution of create_subject of s by s_{k-2} and the

second is the execution of create_object of o by s. We have the desired state-change sequence of length $k-1$, and the algorithm return false.

- $s \in S_{\gamma_{k-3}}$, and $o \in O_{\gamma_{k-3}}$: this case cannot happen, as then, we would need to first destroy each of s and o, which requires two state-changes (we know that $s \neq o$, because otherwise, s would not be able to create o in the last state-change in our sequence of length k). We have already fixed two additional state-changes (create_subject of s, and create_object of o as our last two steps in our state-change sequence of length k). As there do not exist four state changes between γ_{k-3} and γ_k, we know that this case cannot happen.

create_subject – in this case, we know that $o \in S_{\gamma_k}$, and either $s = o$ (and $x = control$), or $s = s_k$ (and $x = own$). We know also that $o \notin S_{\gamma_{k-1}}$. We have the following cases.

- $s = o$: we have the following sub-cases.
  - $o \in S_{\gamma_{k-2}}$: in this case, we know that $s = o \in S_{\gamma_{k-2}}$ and $control \in M_{\gamma_{k-2}}[s, o]$, and therefore we have a state-change sequence of length $k-2$ with the appropriate properties, and therefore by the induction hypothesis, the algorithm returns false.
  - $o \notin S_{\gamma_{k-2}}$: in this case, consider the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-1})} \gamma'_{k-1}$ which is the execution of create_subject of $o = s$ by s_{k-2} (we know that $s_{k-2} \in S_{\gamma_{k-2}} - \mathcal{T}$). We have the desired state-change sequence of length $k-1$ and the algorithm returns false by the induction hypothesis.
- $s = s_k$: we have the following sub-cases.
  - $o \in S_{\gamma_{k-2}}$: in this case, we know that the state-change $\gamma_{k-2} \mapsto_{\psi(s_{k-2})} \gamma_{k-1}$ is the execution of destroy_subject of o by $s_{k-1} \in S_{\gamma_{k-2}} - \mathcal{T}$. We know also, in this case, that $s \in S_{\gamma_{k-2}}$, where $s = s_k$. Therefore, we have the state-change $\gamma_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$ which is the execution of create_subject of

o by s. We have the desired state-change sequence of length $k-1$, and by the induction hypothesis, the algorithm returns false.

– $o \notin S_{\gamma_{k-2}}$: in this case, if $\gamma_{k-2} = \gamma$, then the algorithm does not return in lines 7-11, but returns false in line 12, and we are done. Otherwise, we know that there exists a prior state, γ_{k-3}. We have the following sub-sub-cases.

  * $s \in S_{\gamma_{k-2}}$: in this case, consider the state-change $\gamma_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$ which is the execution of create_subject of o by s. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.

  * $s \notin S_{\gamma_{k-2}}$, $s \in S_{\gamma_{k-3}}$ and $o \in S_{\gamma_{k-3}}$: this cannot happen as we know that $o \notin S_{\gamma_{k-2}}$ and $s \notin S_{\gamma_{k-2}}$, and we cannot create both o and s in a single state-change.

  * $s \notin S_{\gamma_{k-2}}$, $s \notin S_{\gamma_{k-3}}$ and $o \in S_{\gamma_{k-3}}$: in this case, we know that the state-change $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma_{k-2}$ is the execution of destroy_subject of o by s_{k-2}. We consider, instead the state-changes $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s_{k-2})} \gamma'_{k-1}$, where the first state-change is the execution of create_subject of s by s_{k-2}, and the second is the execution of transfer_own of o to s by s_{k-2}. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.

  * $s \notin S_{\gamma_{k-2}}$, $s \in S_{\gamma_{k-3}}$ and $o \notin S_{\gamma_{k-3}}$: in this case, consider the state-change $\gamma_{k-3} \mapsto_{\psi(s)} \gamma'_{k-2}$ which is the execution of create_subject of o by s. We have the desired state-change sequence of length $k-2$, and the algorithm returns false by the induction hypothesis.

  * $s \notin S_{\gamma_{k-2}}$, $s \notin S_{\gamma_{k-3}}$ and $o \notin S_{\gamma_{k-3}}$: in this case, we know that $s_{k-2} \in S_{\gamma_{k-3}} - \mathcal{T}$. Consider the following state-changes: $\gamma_{k-3} \mapsto_{\psi(s_{k-2})} \gamma'_{k-2} \mapsto_{\psi(s)} \gamma'_{k-1}$, where the first state-change is the execution of create_subject of s by s_{k-2}, and the second is the execution of create_subject

of o by s. We have the desired state-change sequence of length $k-1$, and the algorithm returns false by the induction hypothesis.

destroy_subject – in this case, we know that $x = own$, $s = s_k$, $s \neq o$ (as in state γ_k, s has the own right over o), $own \in M_{\gamma_{k-1}}[\widehat{s}, o]$ for some $\widehat{s} \in S_{\gamma_{k-1}}$ with $\widehat{s} \neq s$, and $own \in M_{\gamma_{k-1}}[s, \widehat{s}]$. The state-change is the execution of destroy_subject of $\widehat{s}$ by s to aquire own over o. Let $\widehat{\omega} = \langle \widehat{s}, o, own \rangle$, and $\widehat{e}$ be the execution of the algorithm for the input $(\gamma, \widehat{\omega}, \mathcal{T})$. Then we know that $\widehat{e}$ returns false, by the induction hypothesis. We observe that $\widehat{e}$ cannot return either in line 10 or line 13, because when in $\widehat{e}$, $y \notin R_\psi$. Similarly, let $\omega^s = \langle s, \widehat{s}, own \rangle$, and e^s be the execution of the algorithm for the input $(\gamma, \omega^s, \mathcal{T})$. Then, we know that e^s returns false by the induction hypothesis, but not in line 10 or line 13 (as in the case of e^s as well, $y \notin R_\psi$). Let e be the execution of the algorithm for the input $(\gamma, \omega, \mathcal{T})$. We have the following cases and sub-cases.

- $\widehat{e}$ returns in line 9: in this case, we know that e^s cannot return in line 12, because $\widehat{s} \in O_\gamma$. Therefore, we have the following two sub-cases.
  - e^s returns in line 9: in this case, e does not return in lines 7-8, but may return false in line 9, in which case we are done. Otherwise, e does not return in lines 10-13, but e returns false in line 16, because the conditions are satisfied: we have $\widehat{s}$ that owns o, and $s \in S_\gamma - \mathcal{T}$ that owns $\widehat{s}$.
  - e^s returns in line 16: in this case, e does not return in lines 7-8, but may return false in line 9, in which case we are done. Otherwise, e does not return in lines 10-13. Finally, e returns false in line 16, because the conditions are satisfied: we know that $\widehat{s}$ owns o in γ, and that we have a sequence of subjects as needed in lines 14-16, the first of which owns $\widehat{s}$.
- $\widehat{e}$ returns in line 12: in this case e does not return in lines 7-11, but returns false in line 12 (in particular, we know that e does not return in line 11 because e^s either returns in line 9, which means that $s \in S_\gamma - \mathcal{T}$, or returns in either line 12 or 16, which means that $\exists\, s' \in S_\gamma - \mathcal{T}$).

- $\widehat{e}$ returns in line 16: in this case, e does not return in lines 7-8, but may return in line 9, in which case we are done. Otherwise, e does not return in lines 10-13, but returns in line 16, because the same conditions that cause $\widehat{e}$ to return in line 16 cause e to return in line 16 as well.

The "only if" part: we need to show that if the system is safe with respect to ω and $\mathcal{T}$, then isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ returns true. We show, equivalently, that if isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ returns false, then the system is not safe with respect to ω and $\mathcal{T}$. We do this by considering each case that the algorithm returns false, and showing (by construction) that a sequence of state-changes $\gamma \mapsto_{\psi(s_1)} \gamma_1 \mapsto_{\psi(s_2)} \cdots \mapsto_{\psi(s_n)} \gamma_n$ such that $x \in M_{\gamma_n}[s, o]$ exists (each $s_i \in S_{\gamma_{i-1}} - \mathcal{T}$, and the s_i's may not be distinct from one another). We have the following cases.

- The algorithm returns in line 9: in this case, we have a state-change sequence of length 0 (i.e., simply γ), as we know that $x \in M_\gamma[s, o]$.

- The algorithm returns in line 10: in this case, we again have a state-change sequence of length 0 (i.e., simply γ), as we know that if $x \in \mathcal{R}_b \cap R_\psi$, then $x^* \in M_\gamma[s, o]$ (and possession of x^* implies possession of x), and if $x \in \mathcal{R}_b^* \cap R_\psi$, then $x \in M_\gamma[s, o]$. There are no other cases that the algorithm returns in line 10.

- The algorithm returns in line 12: in this case, we know from the check on line 11 that $\exists\, s' \in S_\gamma - \mathcal{T}$. Therefore, if $s \notin S_\gamma$, we have the following state-change sequence: $\gamma \mapsto_{\psi(s')} \gamma_1 \mapsto_{\psi(s')} \gamma_2 \mapsto_{\psi(s')} \gamma_3$, where the first state-change is the execution of create_subject of s by s', the second state-change is the execution of create_object of o (if $o \in \mathcal{O} - \mathcal{S}$) or create_subject of o (if $o \in \mathcal{S}$) by s', and the last state-change is the execution of one of the following:

  - transfer_own, if $o \in \mathcal{S}$ and $x = own$
  - grant_own, if $o \in \mathcal{O} - \mathcal{S}$ and $x = own$
  - grant_control, if $o \in \mathcal{S}$ and $x = control$

  - grant_r, if $x \in \mathcal{R}_b \cap R_\psi$
  - grant_r*, if $x \in \mathcal{R}_b^* \cap R_\psi$

  If $s \in S_\gamma$, then we simply use the same sequence as above, but without the first state-change (i.e., $\gamma \mapsto_{\psi(s')} \gamma_2 \mapsto_{\psi(s')} \gamma_3$).

- The algorithm returns in line 13: in this case, we know that $x \neq own$ and $x \neq control$. If $s \notin S_\gamma$, our state-change sequence is $\gamma \mapsto_{\psi(\widehat{s})} \gamma_1 \mapsto_{\psi(\widehat{s})} \gamma_2$, where the first state-change is the execution of create_subject of s by $\widehat{s}$, and the second state-change is the execution of transfer_r of x to s over o if $x \in \mathcal{R}_b \cap R_\psi$, or transfer_r* to s over o if $x \in \mathcal{R}_b^* \cap R_\psi$. If $s \in S_\gamma$, then we have simply exclude the first state-change (creation of s) from our state-change sequence.

- The algorithm returns in line 16: Let $\sigma = \{s_1, \ldots, s_n\}$ be the set of subjects alluded to in line 16, and let $s_i \in \sigma$ be such that $s_i \in S_\gamma - \mathcal{T}$, for some integer i such that $1 \leq i \leq n$. We know that $o \in O_\gamma$. If $s \notin S_\gamma$, then the first state-change in our state-change sequence is the execution of create_subject of s by s_i. If $s \in S_\gamma$, we exclude this state-change.

  We then have $i-1$ executions of destroy_subject of each subject s_j such that $j < i$, so that if γ' is the state at the end of the $i-1$ executions, we have $own \in M_{\gamma'}[s_i, o]$. Finally, we have the following cases.

  - $o \in \mathcal{S}$ and $x = own$: in this case, we have the execution of transfer_own of o by s_i to s.
  - $o \in \mathcal{O} - \mathcal{S}$ and $x = own$: in this case, we have the execution of grant_own of o by s_i to s.
  - $o \in \mathcal{S}$, $x = control$ and $\exists\ s'$ such that $control \in M_{\gamma'}[s', o]$: in this case, we have two state-changes, both initiated by s_i. We first have the execution of delete_r of the $control$ right over o from s', and then the execution of grant_control over o to s.

- $o \in \mathcal{S}$, $x = \mathit{control}$ and $\nexists\ s'$ such that $\mathit{control} \in M_{\gamma'}[s', o]$: in this case, we have the execution of grant_control over o to s by s_i.
- $x \in \mathcal{R}_b \cap R_\psi$: in this case we have the execution of grant_r of x over o to s by s_i.
- $x \in \mathcal{R}_b^* \cap R_\psi$: in this case we have the execution of grant_r* of x over o to s by s_i.

■

Theorem 3.2.2 *Safety is efficiently decidable in a system based on the Graham-Denning scheme. In particular,* isSafeGD *runs in time at worst cubic in the size of the components of the start state and the set of rights in the system.*

Proof We make the following observations about the running time of isSafeGD in terms of its input, namely, $S_\gamma, O_\gamma, R_\psi$, $M_\gamma[\,]$, ω and $\mathcal{T}$, by considering each line in the algorithm as follows. Each of the lines 5-10 runs in time at worst linear in the size of the input. In particular, as we mention in the previous section, we adopt a naming convention for subjects and objects that enables us to perform the check $o \in \mathcal{O} - \mathcal{S}$ in line 8, in constant time. Line 11 runs in time at worst quadratic in the size of the input ($|S_\gamma| \times |\mathcal{T}|$), line 12 runs in time at worst linear ($|O_\gamma|$), and line 13 runs in time at worst quadratic ($|S_\gamma| \times |R_\psi|$). As each subject is owned only by one other subject, each sequence to which line 14 refers is of size at most $|S_\gamma|$. Furthermore, there are at most $|S_\gamma|$ such sequences. Therefore, lines 14-16 run in time at worst cubic in the size of the input. The fact that isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ runs in time polynomial in the size of the input in conjunction with Lemma 3.2.1 proves our assertion. ■

.

We observe that cubic running time is only an upper-bound, and is not necessarily a tight upper-bound on the time-complexity of the algorithm. It may be possible, for instance, to store the "chains" of owners in some auxiliary data structure to get a faster running time.

3.3 The Solworth-Sloan scheme and a mapping for DAC schemes

Solworth and Sloan [23] presents a new DAC scheme based on labels and relabelling rules, and we call it the Solworth-Sloan scheme. While the presentation in [23] does not clearly specify what information is maintained in a state and how states may change, we were able to infer what is intended.

In this section, we give a precise characterization of the Solworth-Sloan scheme as a state transition system. Our objective in doing so is to represent the Solworth-Sloan scheme sufficiently precisely to enable comparisons to other DAC schemes. In particular, our intent is to assess the mapping of DAC schemes to the Solworth-Sloan scheme that is discussed by Solworth and Sloan [23]. Solworth and Sloan [23] refers to the DAC schemes discussed by Osborn et al. [11] and asserts that it presents a general access control model that is sufficiently expressive to implement each of these DAC models. In this section, we show that this claim is incorrect.

We reiterate that the DAC schemes discussed by Osborn et al. [11] are either subsumed by, or are minor extensions of the Graham-Denning scheme that we discuss in Section 3.2. We have shown in Section 3.2.2 that safety is efficiently decidable in the Graham-Denning scheme, and our algorithm can be used with relatively minor modifications to decide safety in these schemes. Thereby, Solworth and Sloan's [23] other assertion in reference to the DAC schemes discussed by Osborn et al. [11], that "...every published general access control model...either is insufficiently expressive to represent the full range of DACs or has an undecidable safety problem...", is rendered invalid.

3.3.1 The Solworth-Sloan scheme

Overview There exists the following countably infinite sets of constants:

- a set $\mathcal{S}$ of subjects
- a set $\mathcal{O}$ of objects
- a set $\mathcal{R}$ of rights

- a set $\mathcal{G}$ of groups
- a set $\mathcal{T}^o$ of object tags
- a set $\mathcal{T}^g$ of group tags

An *object label* is a pair $\langle s, t \rangle$, where $s \in \mathcal{S}$ is a subject and $t \in \mathcal{T}^o$ is a object tag.

Which rights a subject has over a particular object are determined indirectly in the following three steps.

1. There is a labelling function label that assigns an object label to each object.

   An object's label may be changed by object relabelling rules, which determine whether an action rewriting one object label into another succeeds or not. For example, when the object label $\ell_1 = \langle s_1, t_1 \rangle$ is relabelled to $\ell_2 = \langle s_2, t_2 \rangle$, all objects that originally have the label ℓ_1 now have the label ℓ_2.

2. There is an authorization function auth that maps each object label and each right to a group. For each object label ℓ and each right r, members of the group identified by $\mathsf{auth}(\ell, r)$ have right r over objects that are assigned the label ℓ.

3. Which subjects are members of a group is determined by native group sets (NGS's), which are complicated structures that we describe below. We define a function $members$ that maps each group to a set of subjects.

We schematically illustrate the steps to determine whether a subject can access an object or not as follows.

$$\text{objects} \xrightarrow{\mathsf{label}} \text{object labels} \xrightarrow{\mathsf{auth}} \text{groups} \xrightarrow{members} \text{subjects}$$

States, Γ A state, γ, is characterized by a 9-tuple $\langle\ S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma \rangle$.

- S_γ is the set of subjects in the state γ; O_γ is the set of objects in the state γ; R_γ is the set of rights in the state γ, and G_γ is the set of groups in state γ.

There is a distinguished right wr, which exists in every state, i.e., $\mathsf{wr} \in R_\gamma$. The role of wr is explained in our discussion of the state-change rules.

- $L_\gamma \subset S_\gamma \times \mathcal{T}^o$ is the finite set of object labels in the state γ.
- label_γ: $O_\gamma \longrightarrow L_\gamma$ assigns a unique object label to each object in the current state.
- auth_γ: $(L_\gamma \times \mathcal{R}_\gamma) \longrightarrow G_\gamma$ maps each pair of an object label and a right to a group. For example, $\mathsf{auth}_\gamma[\ell, \mathsf{re}] = g_1$ means that the group g_1 has the re right over all objects labelled ℓ.
- ORS_γ is an ordered sequence of object relabelling rules, each rule has the form of $\mathsf{rl}(p_1, p_2) = h$, where rl is a keyword, and p_1, p_2 are object patterns. An *object pattern* is a pair, where the first element is a subject in $\mathcal{S}$ or one of the three special symbols $*$, $*\mathsf{u}$, and $*\mathsf{w}$, and the second element is an object tag in $\mathcal{T}^o$ or the special symbol $*$. In the rule $\mathsf{rl}(p_1, p_2) = h$, h is a group, a subject, or one of the four following sets: $\{\}, \{*\}, \{*\mathsf{u}\}, \{*\mathsf{w}\}$. When h is $\{*\mathsf{u}\}$ (resp., $\{*\mathsf{w}\}$), $\{*\mathsf{u}\}$ (resp., $\{*\mathsf{w}\}$) must appear in p_1 or p_2.

  For example, the following is an ORS_γ, in which s_1 is a subject, t_1 is an object tag, and g_1 is a group:

$$\begin{aligned}
\mathsf{rl}(\langle *\mathsf{u}, t_1\rangle,\ \langle s_1, *\rangle) &= g_1 \\
\mathsf{rl}(\langle s_1, *\rangle,\ \langle *\mathsf{u}, t_2\rangle) &= \{*\} \\
\mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{u}, *\rangle) &= \{*\mathsf{u}\} \\
\mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{w}, *\rangle) &= \{\}
\end{aligned}$$

- E_γ is a finite set of native group sets (NGS's) that exist in the state, γ. Each $e \in E_\gamma$ is characterized by the 7-tuple $\langle e.G,\ e.T^g,\ e.\mathsf{gtag},\ e.nt^g,\ e.\mathsf{admin},\ e.\mathsf{patterns},\ e.\mathsf{GRS}\rangle$.
  - $e.G \subseteq G_\gamma$ is the set of groups that are defined in this NGS.
  - $e.T^g \subseteq \mathcal{T}^g$ is the set of group tags that are used in this NGS.
  - The function $e.\mathsf{gtag} : S_\gamma \longrightarrow e.T^g$ assigns a unique tag to each subject in the current state.

- $e.nt^g$ is a group tag in $e.T^g$; it determines when a new subject is added to the state, which tag is assigned to that subject. That is, if a subject s is added, then $e.\mathsf{gtag}[s]$ would be set to $e.nt^g$.
- $e.\mathsf{admin}$ points to one NGS in E_γ; it identifies an NGS in the current state as the administrative group set of the NGS e; $e.\mathsf{admin}$ could be e, in which case e is the administrative group set for itself.
- $e.\mathsf{patterns}$ is a function mapping each group in $e.G$ to a (possibly empty) set of group patterns. Each *group pattern* is a pair where the first element is either a subject in the current state or a special symbol $*\mathsf{u}$, and the second element is a group tag in $e.T^g$. In other words, the set of all group patterns that can be used in e, denoted by $e.P^g$, is $(S_\gamma \cup \{*\mathsf{u}\}) \times e.T^g$, and the signature of $e.\mathsf{patterns}$ is $e.G \longrightarrow 2^{e.P^g}$, where $2^{e.P^g}$ denote the powerset of $e.P^g$.

  For any group $g \in e.G$, $e.\mathsf{patterns}[g]$ gives a set of patterns for determining memberships of the group. Intuitively, the label $\langle *\mathsf{u}, t^g \rangle$ is in $e.\mathsf{patterns}[g]$ means that any subject who is assigned (via the e.gtag function) the group tag t^g is a member of the group; and the label $\langle s, t^g \rangle$ is in $e.\mathsf{patterns}[g]$ means that the subject s is a member of the group if it is assigned the group tag t^g.
- $e.\mathsf{GRS}$ is a set of group relabelling rules, each has the form $Relabel(t_1^g, t_2^g) = g$, where $Relabel$ is a keyword, $t_1^g, t_2^g \in e.T^g$ are two group tags used in this NGS, and g is a group defined in the administrative group set $e.\mathsf{admin}$ (i.e., $g \in e.\mathsf{admin}.G$). The role of a member of $e.\mathsf{GRS}$ is explained in the following discussion of state-change rules in the context of group_tag_relabel.

An additional constraint on the state γ is that each group is defined in exactly one NGS and each group tag can be used in at most one NGS, i.e.,

$$\forall e_1 \in E_\gamma \forall e_2 \in E_\gamma \, (\, e_1.G \cap e_2.g = \emptyset \ \wedge \\ e_1.T^g \cap e_2.T^g = \emptyset \,)$$

We define the following auxiliary function $members_\gamma[\,]: G_\gamma \longrightarrow S_\gamma$ such that $members_\gamma[g]$ gives the set of all subjects that are members of the group g. To determine whether a subject s is in $members_\gamma[g]$, we first determine the unique NGS e, such that $g \in e.G$. Now, $s \in members_\gamma[g]$ if and only if the tag t^g assigned to s (via $e.\mathsf{gtag}$) satisfies the condition that at least one of the two group labels $\langle s, t^g\rangle$ and $\langle *\mathsf{u}, t^g\rangle$ are in the patterns for g, i.e.,

$$\begin{aligned}\exists\, t^g \in e.T^g\ (\ & e.\mathsf{gtag}(s) = t^g\ \wedge \\ & (\ \langle s, t^g\rangle \in e.\mathsf{patterns}[g]\ \vee \\ & \langle *\mathsf{u}, t^g\rangle \in e.\mathsf{patterns}[g]\ )\ )\end{aligned}$$

As an example, consider an NGS e where

$$\begin{aligned}
e.G &= \{\ g_{emp},\ g_{mgr},\ g_{exe}\ \} \\
e.T^g &= \{\ Boss,\ Worker\ \} \\
e.\mathsf{gtag}[s_1] &= Boss \\
e.\mathsf{gtag}[s_2] &= Boss \\
e.\mathsf{gtag}[s_3] &= Worker \\
e.nt^g &= Worker \\
e.\mathsf{admin} &= e \\
e.\mathsf{patterns}[g_{exe}] &= \{\ \langle s_1, Boss\rangle\ \} \\
e.\mathsf{patterns}[g_{mgr}] &= \{\ \langle *\mathsf{u}, Boss\rangle\ \} \\
e.\mathsf{patterns}[g_{emp}] &= \\
&\quad \{\ \langle *\mathsf{u}, Boss\rangle, \langle *\mathsf{u},\ Worker\rangle\ \} \\
e.\mathsf{GRS} &= \\
&\{\ Relabel(Worker, Boss) = g_{mgr} \\
&\ \ Relabel(Boss, Worker) = g_{exe}\ \}
\end{aligned}$$

In this NGS, three groups are defined: executives (g_{exe}), managers (g_{emp}), and employees (g_{mgr}). There are two tags: $Boss$ and $Worker$. There are three subjects; s_1 and s_2 are assigned the tag $Boss$ and s_3 is assigned the tag $Worker$. The new subject tag is $Worker$, so each newly added subject will automatically be assigned the tag

$Worker$. The administrative NGS is e itself. According to the patterns, members of the three groups are as follows:

$$\begin{aligned} members_\gamma[g_{exe}] &= \{s_1\} \\ members_\gamma[g_{mgr}] &= \{s_1, s_2\} \\ members_\gamma[g_{mgr}] &= \{s_1, s_2, s_3\} \end{aligned}$$

The group relabeling rules are such that managers can change a subject's tag from $Worker$ to $Boss$ and executives can change a subject's tag from $Boss$ to $Worker$.

State-Change Rules, Ψ There is a single state transition rule ψ in this scheme; ψ consists of six actions that can result in state changes. These actions are mentioned in Section 3.4 of [23] without precise definitions. (We break up the "Relabel an object" operation in [23] into two relabelling actions.) We describe the actions and their effects when applying them to a state $\gamma = \langle S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma \rangle$. We use γ' to denote the state after the change.

1. $\mathsf{create\_object}(s, o, \ell = \langle s_1, t_1^o \rangle)$: the subject s creates the object o and assigns the object label ℓ to the object o.

   This action succeeds when $s \in S_\gamma$, $o \notin O_\gamma$, $\ell \in L_\gamma$ and the subject s has the distinguished right wr on the object label ℓ, i.e., $s \in members_\gamma[\mathsf{auth}_\gamma(\ell, \mathsf{wr})]$.

   Effects of the action are $O_{\gamma'} = O_\gamma \cup \{o\}$ and the function label is extended so that $\mathsf{label}_{\gamma'}(o) = \langle s_1, t_1^o \rangle$.

2. $\mathsf{create\_label}(s, \ell = \langle s, t_1 \rangle, g_1, g_2, \cdots, g_k)$, where $k = |R_\gamma|$ is the number of rights in γ: the subject s creates the new object label ℓ, and assigns the groups $g_1, g_2, \cdots, g_k$ to have the rights over ℓ, .

   This action succeeds when $s \in S_\gamma$, $\ell \notin L_\gamma$, the subject in ℓ is s, and $g_1, \cdots, g_k \in G_\gamma$.

   The effects of this action are follows. Let $r_1, r_2, \cdots, r_k$ be the k rights in R_γ. Then $L_{\gamma'} = L_\gamma \cup \{\ell\}$ and the function auth is extended such that $\mathsf{auth}_{\gamma'}(\ell, r_i) = g_i$ for $1 \leq i \leq k$.

3. $\mathsf{create\_subject}(s, s')$: the subject s creates a new subject s'.

   This action succeeds when $s \in S_\gamma$ and $s' \notin S_\gamma$.

   The effects of this action are $S_{\gamma'} = S_\gamma \cup \{s'\}$ and for every NGS $e \in E_\gamma$, $e.\mathsf{gtag}$ is extended so that in γ', $e.\mathsf{gtag}(s') = e.nt^g$.

4. $\mathsf{object\_relabel}(s, \ell_1 = \langle s_1, t_1\rangle, \ell_2 = \langle s_2, t_2\rangle)$: the subject s relabels objects having label ℓ_1 to have the label ℓ_2.

   This action succeeds when the first relabelling rule in the object relabelling rule sequence ORS_γ that *matches* (ℓ_1, ℓ_2) is $\mathsf{rl}(p_1, p_2) = h$ and $s \in value[h]$ (the function $value[\,]$ is defined below). The rule $\mathsf{rl}(p_1, p_2) = h$ matches (ℓ_1, ℓ_2) when p_1 matches ℓ_1 and p_2 matches ℓ_2 at the same time. When the pattern $\langle *\mathsf{u}, *\rangle$ matches the label $\langle s_1, t_1\rangle$, we say that $*\mathsf{u}$ is unified with the subject s_1. Note that when $*\mathsf{u}$ occurs more than one times in p_1, p_2, they should be unified with the same subject.

   Recall that h maybe a group g, a subject s', or one of the four sets: $\{\}$, $\{*\}$, $\{*u\}$, $\{*w\}$. The function $value$ is defined as follows: $value[g] = members_\gamma[g]$; $value[s'] = \{s'\}$; $value[\{\}] = \emptyset$, $value[\{*\}] = S_\gamma$; $value[\{*\mathsf{u}\}]$ is the subject that is unified with $*\mathsf{u}$.

   Consider the following ORS_γ.

$$
\begin{aligned}
\mathsf{rl}(\langle *\mathsf{u}, t_1\rangle,\ \langle s_1, *\rangle) &= g_1 \\
\mathsf{rl}(\langle s_1, *\rangle,\ \langle *\mathsf{u}, t_2\rangle) &= \{*\} \\
\mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{u}, *\rangle) &= \{*\mathsf{u}\} \\
\mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{w}, *\rangle) &= \{\}
\end{aligned}
$$

   The action $\mathsf{object\_relabel}(s, \langle s_2, t_1\rangle, \langle s_1, t_2\rangle)$ would match the first relabelling rule, and it would succeed when s is a member of the group g_1. The action $\mathsf{object\_relabel}(s, \langle s_1, t_1\rangle, \langle s_2, t_2\rangle)$ would match the second relabelling rule and always succeeds. The action $\mathsf{object\_relabel}(s, \langle s_2, t_2\rangle, \langle s_2, t_1\rangle)$ would match the third relabelling rule and fail, because $*\mathsf{u}$ is unified with s_2. The action $\mathsf{object\_relabel}(s, \langle s_2, t_2\rangle, \langle s_1, t_1\rangle)$ would match the fourth relabelling rule and fail.

The effect of the relabel action is in the function label. For every object o such that $\mathsf{label}_\gamma[o] = \ell_1$, in the new state, $\mathsf{label}_{\gamma'}[o] = \ell_2$.

5. group_tag_relabel(s, s', t_1^g, t_2^g): the subject s relabels the group tag for the subject s' from t_1^g to t_2^g.

   This action succeeds when there is an NGS $e \in E_\gamma$ such that t_1^g and t_2^g are used in e, the subject s' has the group tag t_1^g in e, there is a corresponding group relabelling rule in $e.\mathsf{GRS}$, and s is a member of the group that can use the relabelling rule. More precisely, the action succeeds when

$$\begin{aligned} \exists e \in E_\gamma \,(\; & e.\mathsf{gtag}[s'] = t_1^g \;\wedge \\ & \text{“}Relabel(t_1^g, t_2^g) = g\text{”} \in e.\mathsf{GRS} \;\wedge \\ & s \in members_\gamma[g] \;) \end{aligned}$$

   Note that the tags t_1^g and t_2^g can appear only in one NGS and they must appear in the same NGS for the action to succeed. The effect of this action is such that the function $e.\mathsf{gtag}$ is changed such that in γ', $e.\mathsf{gtag}[s'] = t_2^g$.

6. create_ngs(s, e): the subject s creates a new NGS e.

   To perform this action, one must provide the complete description of a new NGS e, i.e., the 7-tuple $\langle e.G,\ e.T^g,\ e.\mathsf{gtag},\ e.nt^g,\ e.\mathsf{admin},\ e.\mathsf{patterns},\ e.\mathsf{GRS}\rangle$. For this action to succeed, the groups defined in e and the group tags in e must be new, i.e., they do not appear in any existing NGS's in γ.

   The effects are that $G_{\gamma'} = G_\gamma \cup e.G$ and $E_{\gamma'} = E_\gamma \cup e$.

Given the above state transition rule, we make the following observations. No removal of subjects, objects, labels, or groups is defined. Given a state $\langle S_\gamma,\ O_\gamma,\ R_\gamma,\ G_\gamma,\ L_\gamma,\ \mathsf{label}_\gamma,\ \mathsf{auth}_\gamma,\ \mathsf{ORS}_\gamma,\ E_\gamma\rangle$, S_γ (the set of subjects), O_γ (the set of objects), and G_γ (the set of groups) may change as a result of create_subject, create_object, and create_label, respectively. R_γ, the set of rights, is fixed for the system and does not change. G_γ, the set of groups, may change when a new NGS is added by the create_ngs action. The function

label_γ: $O_\gamma \longrightarrow L_\gamma$ is extended when a new object is added and is changed when an object relabelling action object_relabel happens. The function auth_γ is extended when a new object label is created; existing assignments do not change. ORS_γ, the object relabelling rule sequence, always stay the same. E_γ is extended when a new NGS is added.

3.3.2 Encoding a simple DAC scheme in the Solworth-Sloan scheme

In this section, we encode a relatively simple DAC scheme in the Solworth-Sloan scheme. The DAC scheme we consider is a sub-scheme of the Graham-Denning scheme. It is called Strict DAC with Change of Ownership (SDCO) and is one of the DAC schemes discussed by Osborn et al. [11]. Our construction is based on comments by Solworth and Sloan [23] on how various DAC schemes can be encoded in the Solworth-Sloan scheme. As the presentation in that paper is not detailed, we offer a more detailed construction. Our construction lets us assess the utility of the Solworth-Sloan scheme in encoding SDCO. After we present the encoding, we discuss its deficiencies from the standpoints of correctness, and the overhead it introduces.

Strict DAC with Change of Ownership (SDCO) As we mention above, SDCO is a sub-scheme of the Graham-Denning scheme (see Section 3.2.1). In SDCO, there is a distinguished right, own, but no $control$ right. Also, there are no rights with the copy flag. The state-change rules in SDCO are the commands grant_r (for each $r \in R_\psi$), delete_r (for each $r \in R_\psi$), grant_own, create_object and create_subject. We do not consider commands to destroy subjects or objects as their counterparts are not specified for the Solworth-Sloan scheme.

For simplicity, we consider an SDCO scheme that has only three rights $\mathsf{own}, \mathsf{re}, \mathsf{wr}$. In the Solworth-Sloan scheme, if two objects o_1 and o_2 have the same label, then o_1 and o_2 always have the same access characteristics. That is, in every state, the set of subjects having a right r over o_1 is the same as the set of subjects having the right r over o_2. In SDCO, one can reach states in which o_1 and o_2 have different access characteristics. Therefore, each object needs to be assigned a distinct label.

Therefore, before creating an object, one has to create a new label. When creating a new label ℓ, one has to assign a group to $\mathsf{auth}(\ell, \mathsf{own})$ and a group to $\mathsf{auth}(\ell, \mathsf{re})$; and a group to $\mathsf{auth}(\ell, \mathsf{wr})$. Each pair $\langle \ell, r \rangle$ determines a unique access class. Therefore, a distinct group needs to be created. We use $g(o, r)$ to denote the group that will be assigned to have the right r over object o.

To keep track of which subset of rights a subject has over an object, we need 8 group tags, one corresponding to each subset of $\{own, \mathsf{re}, \mathsf{wr}\}$, we use $t^g(o, x)$, where x is a 3-bit string to denote these tags.

For a subject s to create an object o, s needs to do the following:

1. Create an NGS $e = \langle e.G,\ e.T^g,\ e.\mathsf{gtag},\ e.nt^g,\ \ e.\mathsf{admin},\ e.\mathsf{patterns},\ e.\mathsf{GRS} \rangle$ as follows.

   - $e.G = \{g(o, \mathsf{own}), g(o, \mathsf{re}), g(o, \mathsf{wr})\}$
   - $e.T^g = \{t^g(o, 000),\ t^g(o, 001),\ t^g(o, 010),\ t^g(o, 011),\ t^g(o, 100),\ t^g(o, 101),$ $t^g(o, 110), t^g(o, 111)\}$.
   - $e.\mathsf{gtag}[s] = t^g(o, 100)$ and $e.\mathsf{gtag}[s'] = t^g(o, 000)$ for every $s' \in S_\gamma$ s.t. $s' \neq s$.
   - $e.nt^g = t^g(o, 000)$
   - $e.\mathsf{admin} = e$
   - $e.\mathsf{patterns}[g(o, \mathsf{own})] =$
     $\{\langle *u, t^g(o, 100)\rangle, \langle *u, t^g(o, 101)\rangle,$
     $\langle *u, t^g(o, 110)\rangle, \langle *u, t^g(o, 111)\rangle\}$
     $e.\mathsf{patterns}[g(o, \mathsf{re})] =$
     $\{\langle *u, t^g(o, 010)\rangle, \langle *u, t^g(o, 011)\rangle,$
     $\langle *u, t^g(o, 110)\rangle, \langle *u, t^g(o, 111)\rangle\}$
     $e.\mathsf{patterns}[g(o, \mathsf{wr})] =$
     $\{\langle *u, t^g(o, 001)\rangle, \langle *u, t^g(o, 011)\rangle,$
     $\langle *u, t^g(o, 101)\rangle, \langle *u, t^g(o, 111)\rangle\}$

     That is, in each tag, the first bit corresponds to own, the second to re, and the third to wr. In the set of patterns for the group that corresponds to own, the

first bit is always set in each tag, and similarly for the groups that correspond to re and wr respectively.

- $e.\mathsf{GRS} =$
  $\{Relabel(g(o, b_1b_2b_3), g(o, b'_1b'_2b'_3)) =$
  $g(o, \mathsf{own})$
  $\mid b_1b_2b_3, b'_1b'_2b'_3 \in \{0,1\}^3 \wedge b_1b_2b_3$ and $b'_1b'_2b'_3$ differ in exactly one bit $\}$

2. Use $\mathsf{create\_label}(s, \langle s, t(o)\rangle, g(o, \mathsf{re}), g(o, \mathsf{wr}))$ to create the label $\ell(o)$.

3. Use the action $\mathsf{create\_object}(s, o, \langle s, t(o)\rangle)$ to create the object o and label it with $\ell(o)$.

To grant or revoke a right, one uses group relabelling. For instance, suppose s is a subject, and for the NGS, e, $e.\mathsf{gtag}[s] = t^g(o, 000)$. Then, we know that s is not a member of any of the groups $g(o, \mathsf{own})$, $g(o, \mathsf{re})$ or $g(o, \mathsf{wr})$. The subject would be granted the right re by relabelling $\langle s, t^g(o, 000)\rangle$ to the label $\langle s, t^g(o, 010)\rangle$. The execution of this relabelling results in the subject becoming a member of the group $g(o, \mathsf{re})$, thereby giving him the right re over the object o. Similarly, the subject would have the right re revoked by relabelling $\langle s, t^g(o, 010)\rangle$ to the label $\langle s, t^g(o, 000)\rangle$. These operations can be carried out only by a subject that is a member of the group $g(o, \mathsf{own})$.

We make the following observations about the above mapping.

- The above mapping does not capture the state invariant in SDCO that in every state, there is exactly one owner for every object that exists. In the Solworth-Sloan system that results from the above mapping, one can perform relabelling operations and reach states in which there are mutiple owners for an object, or no owner for an object. For instance, suppose that there already exists a subject s such that $s \in members_\gamma[g(o, \mathsf{own})]$. Given the above relabelling rules, there is nothing that precludes another subject from also becoming a member of the group $g(o, \mathsf{own})$ while s continues to maintain membership in that group. It is also possible to remove the membership of s in the group $g(o, \mathsf{own})$ thereby leaving the object with

no owner. It is unclear how we would prevent such situations from occuring in a system based on the Solworth-Sloan scheme.

- We are unable to capture destruction of subjects and objects as such constructs have not been specified for the Solworth-Sloan scheme. Destruction of subjects and objects is generally considered to be an important component of any access control system. We point out that a state-change rule to destroy a subject or an object in the Solworth-Sloan scheme must be carefully designed, as there are several components of the state (such as tags) of which we must keep track. Therefore, adding such state-change specifications does not appear to be a trivial task. In particular, it is unclear how and with what overhead we can capture in the Solworth-Sloan scheme, the notion of transfer of ownership over objects owned by a subject that is being destroyed.

- There is considerable overhead in implementing a relatively simple DAC scheme (SDCO) in the Solworth-Sloan scheme. For each object, we need to create a set of labels whose size is linear in the number of the subjects in the state. We also need to create a set of tags whose size is exponential in the number rights in the system. These tags are used to define groups, and therefore the number of entries in all the sets of patterns is also exponential in the number of rights in the system. This is considerable overhead considering the simplicity of SDCO, and the fact that one can “directly” implement it, with efficiently decidable safety.

Our conclusion is that several of the claims made by Solworth and Sloan [23] are incorrect. In particular, not only is the motivation (decidable safety) for the creation of the new scheme invalid, but it is also not effective in implementing relatively simple DAC schemes.

4 SECURITY ANALYSIS IN ROLE-BASED ACCESS CONTROL

In this chapter, we discuss security analysis. In particular, we give a precise definition of a family of security analysis problems in RBAC. In this family, we consider queries that are more general than queries that are considered in safety analysis(such as in the previous chapter, and work such as [2,9,25,41]). We show that two classes of the security analysis problems in RBAC can be reduced to similar ones in $\mathsf{RT}[\twoheadleftarrow, \cap]$, a role-based trust-management language for which security analysis has been studied [5]. The reduction gives efficient algorithms for answering most kinds of queries in these two classes and establishes the complexity bounds for the intractable cases. The kinds of reductions we employ provide an introduction to the kinds of reductions we introduce in the next chapter.

4.1 The need for security analysis in RBAC

The administration of large Role-Based Access Control (RBAC) systems is a challenging problem. A case study carried out with Dresdner Bank, a major European bank, resulted in an RBAC system that has around 40,000 users and 1300 roles [62]. In systems of such size, it is impossible for a single system security officer (SSO) to administer the entire system. Several administrative models for RBAC have been proposed in recent years, e.g., ARBAC97 [37], ARABC02 [63], and CL03 (Crampton and Loizou) [40]. In all these models, delegation is used to decentralize the administration tasks.

A major advantage that RBAC has over discretionary access control (DAC) is that if an organization uses RBAC as its access control model, then the organization (represented by the SSO in the system) has central control over its resources. This is different from DAC, in which the creator of a resource determines who can access the resource. In most organizations, even when a resource is created by an employee, the resource is still owned by the organization and the organization wants some level of control over how the

resource is to be shared. In most administrative models for RBAC, the SSO delegates to other users the authority to assign users to certain roles (thereby granting those users certain access permissions), to remove users from certain roles (thereby revoking certain permissions those users have), etc. While the use of delegation in the administration of an RBAC system greatly enhances flexibility and scalability, it may reduce the control that the organization has over its resources, thereby diminishing a major advantage RBAC has over DAC. As delegation gives a certain degree of control to a user that may be only partially trusted, a natural security concern is whether the organization nonetheless has some guarantees about who can access its resources. The effect of delegation on the persistence of security properties in RBAC has not been considered in the literature as such.

In this chapter, we propose to use security analysis techniques [5] to maintain desirable security properties while delegating administrative privileges. In security analysis, one views an access control system as a state-transition system. In an RBAC system, state changes occur via administrative operations. Security analysis techniques answer questions such as whether an undesirable state is reachable, and whether every reachable state satisfies some safety or availability properties. Examples of undesirable states are a state in which an untrusted user gets access and a state in which a user who is entitled to an access permission does not get it.

4.2 Problem definition and main results

In [5], an abstract version of security analysis is defined in the context of trust management. In this section we restate the definition in the context of general access control schemes. We extend our definition of access control schemes from Chapter 1 to suit security analysis.

Definition 4.2.1 *(*Access Control Schemes*)* An access control scheme is modelled as a state-transition system $\langle \Gamma, Q, \vdash, \Psi \rangle$, in which Γ is a set of states, Q is a set of queries, Ψ is a set of state-change rules, and $\vdash$: $\Gamma \times Q \rightarrow \{true, false\}$ is called the entailment relation,

determining whether a *query* is true or not in a given state. A *state*, $\gamma \in \Gamma$, contains all the information necessary for making access control decisions at a given time. When a query, $q \in Q$, arises from an access request, $\gamma \vdash q$ means that the access corresponding to the request q is granted in the state γ, and $\gamma \nvdash q$ means that the access corresponding to q is not granted. One may also ask queries other than those corresponding to a specific request, e.g., whether every principal that has access to a resource is an employee of the organization. Such queries are useful for understanding the properties of a complex access control system.

A state-change rule, $\psi \in \Psi$, determines how the access control system changes state. Given two states γ and γ_1 and a state-change rule ψ, we write $\gamma \mapsto_\psi \gamma_1$ if the change from γ to γ_1 is allowed by ψ, and $\gamma \overset{*}{\mapsto}_\psi \gamma_1$ if a sequence of zero or more allowed state changes leads from γ to γ_1. If $\gamma \overset{*}{\mapsto}_\psi \gamma_1$, we say that γ_1 is ψ-*reachable* from γ, or simply γ_1 is *reachable*, when γ and ψ are clear from the context.

An example of an access control scheme is the HRU scheme, that is derived from the work by Harrison et al. [2]. The HRU scheme is based on the access matrix model [3, 4]. We assume the existence of three countably infinite sets: $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{R}$, which are the sets of all possible subjects, objects, and rights. We assume further that $\mathcal{S} \subseteq \mathcal{O}$. In the HRU scheme:

- Γ is the set of all possible access matrices. Formally, each $\gamma \in \Gamma$ is identified by three finite sets, $S_\gamma \subset \mathcal{S}$, $O_\gamma \subset \mathcal{O}$, and $R_\gamma \subset \mathcal{R}$, and a function $M_\gamma[\,]$: $S_\gamma \times O_\gamma \rightarrow 2^{R_\gamma}$, where $M_\gamma[s, o]$ gives the set of rights s has over o. An example of a state, γ, is one in which $S_\gamma = \{\mathsf{Admin}\}, O_\gamma = \{\mathsf{employeeData}\} \cup S_\gamma$, $R_\gamma = \{own, r\}$, and $M_\gamma[\mathsf{Admin}, \mathsf{Admin}] = \emptyset$, and $M_\gamma[\mathsf{Admin}, \mathsf{employeeData}] = \{own, r\}$. In this state, two objects exist, of which one is a subject, and the system is associated with the two rights, own and r.

- Q is the set of all queries of the form: $r \in [s, o]$, where $r \in \mathcal{R}$ is a right, $s \in \mathcal{S}$ is a subject, and $o \in \mathcal{O}$ is an object. This query asks whether the right r exists in the cell corresponding to subject s and object o.

- The entailment relation is defined as follows: $\gamma \vdash r \in [s, o]$ if and only if $s \in S_\gamma$, $o \in O_\gamma$, and $r \in M_\gamma[s, o]$. For example, let the query q_1 be $r \in M[\mathsf{admin}, \mathsf{employeeData}]$. and the query q_2 be $own \in M[\mathsf{admin}, \mathsf{admin}]$ Then, for the state, γ, discussed above, $\gamma \vdash q_1$ and $\gamma \nvdash q_2$.

- Each state-transition rule ψ is given by a set of commands. Given ψ, the change from γ to γ_1 is allowed if there exists command in ψ such that the execution of the command in the state γ results in the state γ_1. An example of ψ is the following set of commands.

$$
\begin{array}{ll}
command\ \ createObject(s, o) & command\ \ grant\_r(s, s', o) \\
\quad create\ object\ o & \quad if\ own \in [s, o] \\
\quad enter\ own\ into\ [s, o] & \quad\quad enter\ r\ into\ [s', o]
\end{array}
$$

The set of queries is not explicitly specified in [2]. It is conceivable to consider other classes of queries, e.g., comparing the set of all subjects that have a given right over a given object with another set of subjects. In our framework, HRU with different classes of queries can be viewed as different schemes.

Definition 4.2.2 *(*Security Analysis in an Abstract Setting*)* Given an access control scheme $\langle \Gamma, Q, \vdash, \Psi \rangle$, a security analysis instance takes the form $\langle \gamma, q, \psi, \Pi \rangle$, where $\gamma \in \Gamma$ is a state, $q \in Q$ is a query, $\psi \in \Psi$ is a state-change rule, and $\Pi \in \{\exists, \forall\}$ is a quantifier. An instance $\langle \gamma, q, \psi, \exists \rangle$ asks whether there exists γ_1 such that $\gamma \overset{*}{\mapsto}_\psi \gamma_1$ and $\gamma_1 \vdash q$. When the answer is affirmative, we say q is *possible* (given γ and ψ). An instance $\langle \gamma, q, \psi, \forall \rangle$ asks whether for every γ_1 such that $\gamma \overset{*}{\mapsto}_\psi \gamma_1$, $\gamma_1 \vdash q$. If so, we say q is *necessary* (given γ and ψ).

For our example HRU scheme from above, adopt γ as the start state. In γ, there is only one subject (namely, Admin) and the access matrix is empty. The system is associated with the two rights, own and r. Let the query q be $r \in M[\mathsf{Alice}, \mathsf{employeeData}]$ for $\mathsf{Alice} \in \mathcal{S}$ and $\mathsf{employeeData} \in \mathcal{O}$. Let the state-change rule ψ be the set of two commands $createObject$ and $grant\_r$. Then, the security analysis instance $\langle \gamma, q, \psi, \exists \rangle$ is true. The

reason is that although in the start state γ, Alice does not have the r right over the object employeeData, there exists a reachable state from γ in which she has such access. The security analysis instance $\langle \gamma, q, \psi, \forall \rangle$ is false, as there exists at least one state reachable from γ (γ itself) that does not entail the query.

Security analysis generalizes safety analysis. As we discuss in the following section, with security analysis we can study not only safety, but also several other interesting properties, such as availability and mutual-exclusion.

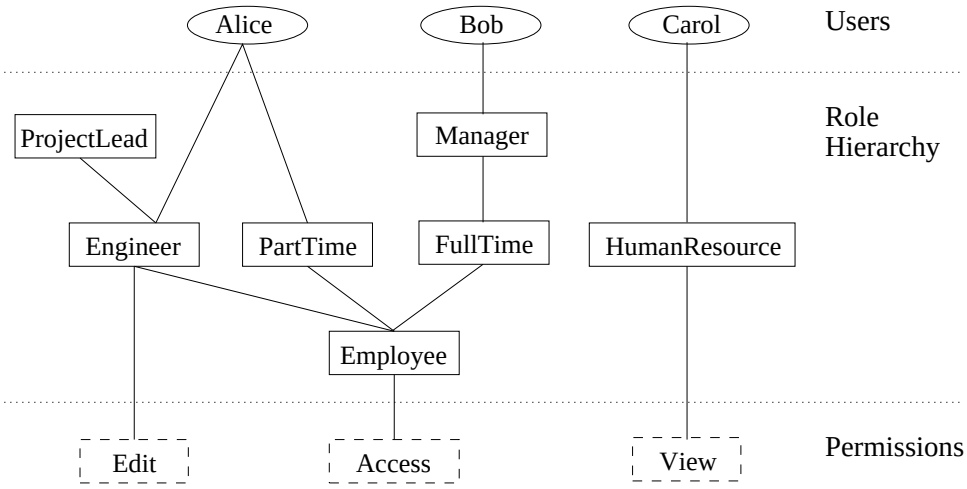
4.2.1 A family of security analysis problems in RBAC

We now define a family of security analysis problems in the context of RBAC by specifying Γ, Q, and $\vdash$, while leaving Ψ abstract. By considering different possibilities for Ψ, one obtains different classes of RBAC security analysis problems in this family. We consider two specific instances of Ψ in sections 4.2.3 and 4.2.4.

An introduction to RBAC is provided in [13, 64]. We assume that there are three countable sets: $\mathcal{U}$ (the set of all possible users), $\mathcal{R}$ (the set of all possible roles), and $\mathcal{P}$ (the set of all possible permissions). The family of analysis problems is given by specializing the analysis problem defined in Definition 4.2.2 to consider access control schemes that have Γ, Q, and $\vdash$ specified as follows.

States (Γ): Γ is the set of all RBAC states. An RBAC state, γ, is a 3-tuple $\langle UA, PA, RH \rangle$, in which the user assignment relation $UA \subseteq \mathcal{U} \times \mathcal{R}$ associates users with roles, the permission assignment relation $PA \subseteq \mathcal{P} \times \mathcal{R}$ associates permissions with roles, and the role hierarchy relation $RH \subseteq \mathcal{R} \times \mathcal{R}$ is a partial order among roles in $\mathcal{R}$. We denote the partial order by $\succeq$. $r_1 \succeq r_2$ means that every user who is a member of r_1 is also a member of r_2 and every permission that is associated with r_2 is also associated with r_1.

Example 1 Figure 4.1 is an example of an RBAC state. It reflects an organization that has engineers, and whose human-resource needs are outsourced (i.e., human-resource personnel are not employees). Everyone in the organization is an employee, and therefore a member of the role Employee. Some of the employees are full-time (members of the role



$$
\begin{aligned}
RH &= \{\, (\mathsf{Engineer}, \mathsf{Employee}) \,, (\mathsf{FullTime}, \mathsf{Employee}), \\
&\quad\ (\mathsf{PartTime}, \mathsf{Employee}) \,, (\mathsf{ProjectLead}, \mathsf{Engineer}), \\
&\quad\ (\mathsf{Manager}, \mathsf{FullTime}) \,\}. \\
PA &= \{\, (\mathsf{Access}, \mathsf{Employee}) \,, (\mathsf{View}, \mathsf{HumanResource}), \\
&\quad\ (\mathsf{Edit}, \mathsf{Engineer}) \,\}. \\
UA &= \{\, (\mathsf{Alice}, \mathsf{PartTime}) \,, (\mathsf{Alice}, \mathsf{Engineer}), \\
&\quad\ (\mathsf{Bob}, \mathsf{Manager}) \,, (\mathsf{Carol}, \mathsf{HumanResource}) \,\}.
\end{aligned}
$$

Figure 4.1. An example RBAC state with a role hierarchy, users and permissions. Roles are shown in solid boxes, permissions in dashed boxes and users in ovals. A line segment represents a role-role relationship, the assignment of a permission to a role or the assignment of a user to a role.

FullTime), and the others are part-time (members of the role PartTime). All managers are full-time employees. All employees have access to the office, and therefore have the permission Access. Engineers may edit code (have the permission Edit), and human resource personnel may view employee-details (have the permission View).

We now discuss some example members of UA, PA and RH. The user Alice is an engineer who is a part-time employee. Therefore, $(\mathsf{Alice}, \mathsf{Engineer})$ and $(\mathsf{Alice}, \mathsf{PartTime})$ are members of UA. All employees have access to the office, and therefore, $(\mathsf{Access}, \mathsf{Employee})$ is a member of PA. Project leads are engineers, and therefore $(\mathsf{ProjectLead}, \mathsf{Engineer})$ is a member of RH (i.e., $\mathsf{ProjectLead} \succeq \mathsf{Engineer}$).

Given a state γ, every role has a set of users who are members of that role and every permission is associated with a set of users who have that permission. We formalize this by having every state γ define a function $\mathsf{users}_\gamma : R \cup P \rightarrow 2^U$, as follows. For any $r \in R$ and $u \in U$, $u \in \mathsf{users}_\gamma[r]$ if and only if either $(u, r) \in UA$ or there exists r_1 such that $r_1 \succeq r$ and $(u, r_1) \in UA$. For any $p \in \mathcal{P}$ and $u \in U$, $u \in \mathsf{users}_\gamma[p]$ if and only if there exists r_1 such that $(p, r_1) \in PA$ and $u \in \mathsf{users}_\gamma[r_1]$. Note that the effect of permission propagation through the role hierarchy is already taken into consideration by the definition of $\mathsf{users}_\gamma[r_1]$.

Example 2 Let the RBAC state shown in Figure 4.1 be γ. Then, for the role Engineer, $\mathsf{users}_\gamma[\mathsf{Engineer}] = \{\mathsf{Alice}\}$. Similarly, for the permission Access, $\mathsf{users}_\gamma[\mathsf{Access}] = \{\mathsf{Alice}, \mathsf{Bob}\}$.

Queries (Q): A query q has the form $s_1 \sqsupseteq s_2$, where $s_1, s_2 \in S$, and S is the set of all *user sets*, defined to be the least set satisfying the following conditions: (1) $R \cup P \subseteq S$, i.e., every role r and every permission p is a user set; (2) $\{u_1, u_2, \cdots, u_k\} \in S$, where $k \geq 0$ and $u_i \in U$ for $1 \leq i \leq k$, i.e., a finite set of users is a user set; and (3) $s_1 \cup s_2, s_1 \cap s_2, (s_1) \in S$, where $s_1, s_2 \in S$, i.e., the set of all user sets is closed with respect to union, intersection and paranthesization. We extend the function users_γ in a straightforward way to give a valuation for all user sets. The extended function users_γ: $S \rightarrow 2^U$ is defined as follows: $\mathsf{users}_\gamma[\{u_1, u_2, \cdots, u_k\}] = \{u_1, u_2, \cdots, u_k\}$, $\mathsf{users}_\gamma[(s)] = \mathsf{users}_\gamma[s]$, $\mathsf{users}_\gamma[s_1 \cup s_2] =$

$\mathsf{users}_\gamma[s_1] \cup \mathsf{users}_\gamma[s_2]$, and $\mathsf{users}_\gamma[s_1 \cap s_2] = \mathsf{users}_\gamma[s_1] \cap \mathsf{users}_\gamma[s_2]$. We say a query $s_1 \sqsupseteq s_2$ is *semi-static* if one of s_1, s_2 can be evaluated independent of the state, i.e., no role or permission appears in it. The reason we distinguish semi-static queries is that (as we assert in Sections 4.4.1 and 4.4.2) a security analysis instance involving only such queries can be solved efficiently.

Entailment ($\vdash$): Given a state γ and a query $s_1 \sqsupseteq s_2$, $\gamma \vdash s_1 \sqsupseteq s_2$ if and only if $\mathsf{users}_\gamma[s_1] \supseteq \mathsf{users}_\gamma[s_2]$.

Example 3 Continuing from the previous examples, an example of a query, q, is $\mathsf{FullTime} \cap \mathsf{Access} \sqsupseteq \{\mathsf{Alice}\}$, for the role FullTime, the permission Access and the user Alice. This query is semi-static; the user set $\{\mathsf{Alice}\}$ can be evaluated (to itself) independent of the state.

The query q asks whether Alice is a full-time employee that has access to the office. To find out whether γ entails q or not, we evaluate q as follows. We evaluate the user set FullTime to the set of users $\{\mathsf{Bob}\}$. We evaluate the user set Access to the set of users $\{\mathsf{Alice}, \mathsf{Bob}\}$. We intersect the two sets of users to obtain the set of users $\{\mathsf{Bob}\}$. The user set $\{\mathsf{Alice}\}$ does not need further evaluation; it is already a set of users. We now check whether the set of users $\{\mathsf{Alice}\}$ is a subset of the set of users $\{\mathsf{Bob}\}$ and determine that $\gamma \nvdash q$. If another query q' is $\mathsf{Edit} \sqsupseteq \mathsf{ProjectLead}$ (i.e., whether project leads can edit code), then $\gamma \vdash q'$.

The state of an RBAC system changes when a modification is made to a component of $\langle UA, PA, RH \rangle$. For example, a user may be assigned to a role, or a role hierarchy relationship may be added. In existing RBAC models, both constraints and administrative models affect state changes in an RBAC system. For example, a constraint may declare that roles r_1 and r_2 are mutually exclusive, meaning that no user can be a member of both roles. If a user u is a member of r_1 in a state, then the state is not allowed to change to a state in which u is a member of r_2 as well. An *administrative model* includes administrative relations that dictates who has the authority to change the various components of an RBAC state and what are the requirements these changes have to satisfy. Thus, in RBAC

security analysis, a state-change rule may include constraints, administrative relations, and possibly other information.

In this section, we leave the state-change rule abstract for the following reasons. First, there are several competing proposals for constraint languages [65–67] and for administrative models in RBAC [37, 40, 63, 68]; a consensus has not been reached within the community. Furthermore, RBAC is used in diverse applications. It is conceivable that different applications would use different classes of constraints and/or administrative models; therefore different classes of problems in this family are of interest.

Given a state γ and a state-change rule ψ, one can ask the following questions using security analysis.

- *Simple Safety* is $s \sqsupseteq \{u\}$ possible? This asks whether there exists a reachable state in which the user set s includes the (presumably untrusted) user u. A 'no' answer means that the system is safe.

- *Simple Availability* is $s \sqsupseteq \{u\}$ necessary? This asks whether in every reachable state, the (presumably trusted) user u is always included in the user set s. A 'yes' answer means that the resources associated with the user set s are always available to the user u.

- *Bounded Safety* is $\{u_1, u_2, \ldots, u_n\} \sqsupseteq s$ necessary? This asks whether in every reachable state, the user set s is bounded by the set of users $\{u_1, u_2, \ldots, u_n\}$. A 'yes' answer means that the system is safe. A special case of bounded safety is *Mutual Exclusion*, which asks: is $\emptyset \sqsupseteq (s_1 \cap s_2)$ necessary? This asks whether in every reachable state, no user is a member of both user sets s_1 and s_2. A 'yes' answer means that the two user sets are mutually exclusive.

- *Liveness* is $\emptyset \sqsupseteq s$ possible? This asks whether the user set s always has at least one user. A 'no' answer means that the liveness of the resources associated with s holds in the system.

- *Containment* is $s_1 \sqsupseteq s_2$ necessary? This asks whether in every reachable state, every user in the user set s_2 is in the user set s_1. Containment can be used to express a safety property, in which case, a 'yes' answer means that the safety property holds. An example of containment for the RBAC state in Figure 4.1 and some state-change rule is: "is Employee $\sqsupseteq$ Access necessary?", for the role Employee and the permission Access. This asks whether in every reachable state, every user who has the permission Access (i.e., has access to the office) is a member of the role Employee (i.e., is an employee). A 'yes' answer means that our desired safety property holds.

  Containment can express availability properties also. E.g., "is Access $\sqsupseteq$ Employee necessary?" asks whether the permission Access (i.e., access to the office) is always available to members of the role Employee (i.e., employees). A 'yes' answer means that the availability property holds.

We point out that that all the above properties (except for containment) use semi-static queries, and therefore, as we mention in the context of queries in this section, we can efficiently determine whether those properties are satisfied.

4.2.2 Usage of RBAC security analysis

In an RBAC security analysis instance $\langle \gamma, q, \psi, \Pi \rangle$, the state γ fully determines who can access which resources. In addition to administrative policy information, the state-change rule ψ also contains information about which users are trusted. In any access control system there are *trusted users*; these are users who have the authority to take the system to a state that violates security requirements but are trusted not to do so. A Senior Security Office (SSO) is an example of a trusted user.

Security analysis provides a means to ensure that security requirements (such as safety and availability) are always met, as long as users identified as trusted behave according to the usage patterns discussed in this section. In other words, security analysis helps ensure that the security of the system does not depend on users other than those that are trusted.

Each security requirement is formalized as a security analysis instance, together with an answer that is acceptable for secure operation. For example, in the context of the RBAC system whose state in shown in Figure 4.1, a security requirement may be that only employees may access the office. This can be formalized as an instance $\langle \gamma, q, \psi, \forall \rangle$, where γ is the current state, q is $\mathsf{Employee} \sqsupseteq \mathsf{Access}$, and ψ specifies administrative policy information. The rule ψ should precisely capture the capabilities of users that are not trusted. In other words, any change that could be made by such users should be allowed by ψ. The rule ψ could restrict the changes that trusted users can make, because these are trusted not to make a change without verifying that desirable security properties are maintained subsequent to the change. For the example discussed above, the acceptable answer is "yes", as we want to ensure that everyone who has the permission Access is an employee. The goal is to ensure that such a security requirement is always satisfied.

Suppose that the system starts in a state γ such that the answer to $\langle \gamma, q, \psi, \forall \rangle$ is "yes". Further, suppose a trusted user (such as the SSO) attempts to make a change that is not allowed by ψ, e.g., the SSO decides to grant certain administrative privileges to a user u. Before making the change, SSO performs security analysis $\langle \gamma', q, \psi', \forall \rangle$, where γ' and ψ' result from the prospective change. Only if the answer is "yes" does the SSO actually make the change. The fact that ψ limits the SSO from making changes does not mean that we require that the SSO never make such changes. It reflects the requirement that the SSO perform security analysis and make only those changes that do not violate security properties.

This way, as long as trusted users are cooperating, the security of an access control system is preserved. One can delegate administrative privileges to partially trusted users with the assurance that desirable security properties always hold. By using different ψ's, one can evaluate which sets of users are trusted for a given security property. In general, it is impossible to completely eliminate the need to trust people. However, security analysis enables one to ensure that the extent of this trust is well understood.

4.2.3 Assignment and trusted users (AATU)

In this chapter, we present solutions to two classes of security analysis problems in RBAC. Both classes use variants of the URA97 component of the ARBAC97 administrative model for RBAC [37]. URA97 specifies how the UA relation may change.

The first class is called Assignment And Trusted Users (AATU), in which a state-change rule ψ has the form $\langle can\_assign, T\rangle$. The relation $can\_assign \subseteq R \times C \times 2^R$ determines who can assign users to roles and the preconditions these users have to satisfy. C is the set of conditions, which are expressions formed using roles, the two operators $\cap$ and $\cup$, and parentheses. $\langle r_a, c, rset\rangle \in can\_assign$ means that members of the role r_a can assign any user whose role memberships satisfy the condition c, to any role $r \in rset$. For example, $\langle r_0, (r_1 \cup r_2) \cap r_3, \{r_4, r_5\}\rangle \in can\_assign$ means that a user that is a member of the role r_0 is allowed to assign a user that is a member of at least one of r_1 and r_2, and is also a member of r_3, to be a member of r_4 or r_5. $T \subseteq U$ is a set of trusted users; these users are assumed not to initiate any role assignment operation for the purpose of security analysis. The set T is allowed to be empty.

Definition 4.2.3 *(*Assignment And Trusted Users – AATU*)* The class AATU is given by parameterizing the family of RBAC analysis problems in Section 4.2.1 with the following set of state-change rules. Each state-change rule ψ has the form $\langle can\_assign, T\rangle$ such that a state change from $\gamma = \langle UA, PA, RH\rangle$ to $\gamma_1 = \langle UA_1, PA_1, RH_1\rangle$ is allowed by $\psi = \langle can\_assign, T\rangle$ if $PA = PA_1$, $RH = RH_1$, $UA_1 = UA \cup \{(u, r)\}$, where $(u, r) \notin UA$ and there exists $(r_a, c, rset) \in can\_assign$ such that $r \in rset$, u satisfies c, and $\mathsf{users}_\gamma[r_a] \not\subseteq T$ (i.e., there exists at least one user who is a member of the role r_a and is not in T, so that such a user can perform the assignment operation).

Example 4 For the state, γ, shown in Figure 4.1 and discussed in the previous examples, a state-change rule, ψ, in the class AATU is $\langle can\_assign, T\rangle$, where

$$\begin{aligned} can\_assign &= \{\langle \mathsf{Manager}, \mathsf{Engineer} \wedge \mathsf{FullTime}, \{\mathsf{ProjectLead}\}\rangle, \\ &\qquad \langle \mathsf{HumanResource}, true, \{\mathsf{FullTime}, \mathsf{PartTime}\}\rangle\} \end{aligned}$$

$$T = \{\mathsf{Carol}\}$$

That is, ψ authorizes managers to assign a user to the role ProjectLead provided that the user is a member of the roles Engineer and FullTime. In addition, ψ authorizes anyone that is a member of the role HumanResource to assign users to the roles FullTime and PartTime. Setting T to $\{\mathsf{Carol}\}$ implies that we wish to analyze what kinds of states can be reached via changes made by users other than Carol.

Let q be the query $\mathsf{ProjectLead} \sqsupseteq \{\mathsf{Alice}\}$. Then, $\gamma \nvdash q$. The analysis instance $\langle \gamma, q, \psi, \exists\rangle$ asks whether there exists a reachable state in which Alice is a project lead. The instance is false. This is because for Alice to become a member of ProjectLead, she would first need to be a full-time employee, and only Carol can grant anyone membership to FullTime. As Carol is in T, she cannot initiate any operation. If we consider, instead, the state-change rule ψ', with the same can_assign as ψ from above, but with $T = \emptyset$, then the analysis instance $\langle \gamma, q, \psi', \exists\rangle$ is true.

Main results for AATU

- If q is semi-static (see Section 4.2.1), then an AATU instance $\langle \gamma, q, \psi, \Pi\rangle$ can be answered efficiently, i.e., in time polynomial in the size of the instance. This is asserted by Theorem 4.4.2 in Section 4.4.1.

- Answering general AATU instances $\langle \gamma, q, \psi, \forall\rangle$ is decidable but intractable (**coNP**-hard). This is asserted by Theorem 4.4.3 in Section 4.4.1.

4.2.4 Assignment and revocation (AAR)

In this class, a state-change rule ψ has the form $\langle can\_assign, can\_revoke\rangle$, where can_assign is the same as in AATU, and $can\_revoke \subseteq R \times 2^R$ determines who can remove users from roles. That $\langle r_a, rset\rangle \in can\_revoke$ means that the members of role r_a can remove a user from a role $r \in rset$. No explicit set of trusted users is specified in AAR, unlike AATU. In AATU and AAR, the relations can_assign and can_revoke are fixed in ψ. This means that we are assuming that changes to these two relations are made only by trusted users.

Definition 4.2.4 *(*Assignment And Revocation – AAR*)* The class AAR is given by parameterizing the family of RBAC analysis problems in Section 4.2.1 with the following set of state-change rules. Each state-change rule ψ has the form $\langle can\_assign, can\_revoke\rangle$ such that a state-change from $\gamma = \langle UA, PA, RH\rangle$ to $\gamma_1 = \langle UA_1, PA_1, RH_1\rangle$ is allowed by $\psi = \langle can\_assign, can\_revoke\rangle$ if $PA = PA_1$, $RH = RH_1$, and either (1) $UA_1 = UA \cup \{(u, r)\}$ where $(u, r) \notin UA$ and there exists $(r_a, c, rset) \in can\_assign$ such that $r \in rset$, u satisfies c, and $\mathsf{users}_\gamma[r_a] \neq \emptyset$, i.e., the user u being assigned to r is not already a member of r and satisfies the precondition c, and there is at least one user that is a member of the role r_a that can perform the assignment operation; or (2) $UA_1 \cup (u, r) = UA$ where $(u, r) \notin UA_1$, and there exists $(r_a, rset) \in can\_revoke$ such that $r \in rset$ and $\mathsf{users}_\gamma[r_a] \neq \emptyset$, i.e., there exists at least one user in the role r_a that can revoke the user u's membership in the role r.

We assume that an AAR instance satisfies the following three properties. (1) The administrative roles are not affected by can_assign and can_revoke. The administrative roles are given by those that appear in the first component of any can_assign or can_revoke tuple. These roles should not appear in the last component of any can_assign or can_revoke tuple. This condition is easily satisfied in URA97, as it assumes the existence of a set of administrative roles that is disjoint from the set of normal roles. (2) If a role is an administrative role (i.e., appears as the first component of a can_assign or can_revoke tuple), then it has at least one user assigned to it. This is reasonable, as an administrative role with no

members has no effect on the system's protection state. (3) If a can_assign tuple exists for a role, then a can_revoke tuple also exists for that role.

Example 5 For the state, γ, from Figure 4.1, an example of a state-change rule in AAR is $\psi = \langle can\_assign, can\_revoke \rangle$, where

$$\begin{aligned} can\_assign &= \{\langle \mathsf{Manager}, \mathsf{Engineer} \wedge \mathsf{FullTime}, \{\mathsf{ProjectLead}\}\rangle, \\ &\quad \langle \mathsf{HumanResource}, true, \{\mathsf{FullTime}, \mathsf{PartTime}\}\rangle\} \end{aligned}$$

$$\begin{aligned} can\_revoke &= \{\langle \mathsf{Manager}, \{\mathsf{ProjectLead}, \mathsf{Engineer}\}\rangle, \\ &\quad \langle \mathsf{HumanResource}, \{\mathsf{FullTime}, \mathsf{PartTime}\}\rangle\} \end{aligned}$$

We point out that the can_assign we use in this example is the same as the can_assign we use in Example 4. Then, if q is the query $\mathsf{ProjectLead} \sqsupseteq \mathsf{Access}$ (i.e., only project leads have access to the office), the AAR analysis instance $\langle \gamma, q, \psi, \exists \rangle$ is true. If q' is the query $\mathsf{Edit} \sqsupseteq \{\mathsf{Alice}\}$ (i.e., Alice can edit code), then the analysis instance $\langle \gamma, q', \psi, \forall \rangle$ is false.

Main results for AAR

- If q is semi-static (see Section 4.2.1), then an AAR instance $\langle \gamma, q, \psi, \Pi \rangle$ can be answered efficiently, i.e., in time polynomial in the size of the instance. This is asserted by Theorem 4.4.5 in Section 4.4.2.
- Answering general AAR instances $\langle \gamma, q, \psi, \forall \rangle$ is $\mathbf{coNP}$-complete. This is asserted by Theorem 4.4.6 in Section 4.4.2.

4.2.5 Discussion of the definitions

Our specifications of can_assign and can_revoke are from URA97, which is one of the three components of ARBAC97 [37]. The state-change rules considered in AAR are similar to those in URA97, but they differ in the following two ways. One, URA97 allows negation of roles to be used in a precondition; AAR does not allow this. Two, URA97 has

separate administrative roles; AAR does not require the complete separation of administrative roles from ordinary roles. AATU differs from URA97 in two additional ways. One, AATU does not have revocation rules. Two, AATU has a set of trusted users, which does not exist in URA97.

The other components of ARBAC97 are PRA97 and RRA97, for administering permission-role assignment/revocation, and the role hierarchy, respectively. In this chapter, we study the effect of decentralizing user-role assignment and revocation, and assume that changes to the permission-role assignment relation and the role hierarchy are centralized, i.e, made only by trusted users. In other words, whoever is allowed to make changes to permission-role assignment and the role hierarchy will run the security analysis and only make changes that do not violate the security properties. The administration of the user-role relation is most likely to be delegated, as that is the component of an RBAC state that changes most frequently.

AATU and AAR represent two basic cases of security analysis in RBAC. Although we believe that they are useful cases, they are only the starting point. Many other more sophisticated cases of security analysis in RBAC remain open. For example, it is not clear how to deal with negative preconditions in role assignment, and how to deal with constraints such as mutually exclusive roles.

4.3 Overview of security analysis in $\mathsf{RT}[\twoheadleftarrow, \cap]$

In [5], Li et al. studies security analysis in the context of the RT family of Role-based Trust-management languages [69, 70]. In particular, security analysis in $\mathsf{RT}[\twoheadleftarrow, \cap]$ and its sub-languages is studied. $\mathsf{RT}[\twoheadleftarrow, \cap]$ is a slightly simplified (yet expressively equivalent) version of the RT_0 language introduced in [70] ($\mathsf{RT}[\twoheadleftarrow, \cap]$ is called *SRT* in [5]). In this section we summarize the results for security analysis in $\mathsf{RT}[\twoheadleftarrow, \cap]$. In Section 4.4 we reduce security analysis in AATU and AAR to that in $\mathsf{RT}[\twoheadleftarrow, \cap]$.

Syntax of $\mathsf{RT}[\twoheadleftarrow, \cap]$ The most important concept in the RT languages is also that of *roles*. A role in $\mathsf{RT}[\twoheadleftarrow, \cap]$ is denoted by a principal (corresponding to a user in RBAC)

Simple Member

syntax:	$K.r \longleftarrow K_1$
meaning:	$\mathsf{members}(K.r) \supseteq \{K_1\}$
LP clause:	$m(K, r, K_1)$

Simple Inclusion

syntax:	$K.r \longleftarrow K_1.r_1$
meaning:	$\mathsf{members}(K.r) \supseteq \mathsf{members}(K_1.r_1)$
LP clause:	$m(K, r, ?Z) :- m(K_1, r_1, ?Z)$

Linking Inclusion

syntax:	$K.r \longleftarrow K.r_1.r_2$
meaning:	$\mathsf{members}(K.r) \supseteq \bigcup_{K_1 \in K.r_1} \mathsf{members}(K_1.r_2)$
LP clause:	$m(K, r, ?Z) :- m(K, r_1, ?Y),\ m(?Y, r_2, ?Z)$

Intersection Inclusion

syntax:	$K.r \longleftarrow K_1.r_1 \cap K_2.r_2$
meaning:	$\mathsf{members}(K.r) \supseteq \mathsf{members}(K_1.r_1) \cap \mathsf{members}(K_2.r_2)$
LP clause:	$m(K, r, ?Z) :- m(K_1, r_1, ?Z),\ m(K_2, r_2, ?Z)$

Figure 4.2. Statements in $\mathsf{RT}[\twoheadleftarrow, \cap]$. There are four types of statements. For each type, we give the syntax, the intuitive meaning of the statement, and the LP (Logic-Programming) clause corresponding to the statement. The clause uses one ternary predicate m, where $m(K, r, K_1)$ means that K_1 is a member of the role $K.r$ Symbols that start with "$?$" represent logical variables.

followed by a role name, separated by a dot. For example, when K is a principal and r is a role name, $K.r$ is a role. Each principal has its own name space for roles. For example, the 'employee' role of one company is different from the 'employee' role of another company. A *role* has a value which is a set of principals that are members of the role.

Each principal K has the authority to designate the members of a role of the form $K.r$. Roles are defined by *statements*. Figure 4.2 shows the four types of statements in $\mathsf{RT}[\twoheadleftarrow, \cap]$; each corresponds to a way of defining role membership. A simple-member statement $K.r \longleftarrow K_1$ means that K_1 is a member of K's r role. This is similar to a user assignment in RBAC. A simple inclusion statement $K.r \longleftarrow K_1.r_1$ means that K's r role includes (all members of) K_1's r_1 role. This is similar to a role-role dominance relationship $K_1.r_1 \succeq K.r$. A linking inclusion statement $K.r \longleftarrow K.r_1.r_2$ means that $K.r$ includes $K_1.r_2$ for every K_1 that is a member of $K.r_1$. An intersection inclusion statement $K.r \longleftarrow K_1.r_1 \cap K_2.r_2$ means that $K.r$ includes every principal who is a member of both $K_1.r_1$ and $K_2.r_2$. Linking and intersection inclusion statements do not directly correspond to constructs in RBAC, but they are useful in expressing memberships in roles that result from administrative operations. Our reduction algorithms in Sections 4.4.1 and 4.4.2 use linking and intersection inclusion statements to capture user-role memberships affected by administrative operations.

States An $\mathsf{RT}[\twoheadleftarrow, \cap]$ state γ^T consists of a set of $\mathsf{RT}[\twoheadleftarrow, \cap]$ statements. The semantics of $\mathsf{RT}[\twoheadleftarrow, \cap]$ is given by translating each statement into a datalog clause. (Datalog is a restricted form of logic programming (LP) with variables, predicates, and constants, but without function symbols.) See Figure 4.2 for the datalog clauses corresponding to $\mathsf{RT}[\twoheadleftarrow, \cap]$ statements. We call the datalog program resulting from translating each statement in γ^T into a clause that is the *semantic program* of γ^T, denoted by $SP(\gamma^T)$.

Given a datalog program, $\mathcal{DP}$, its semantics can be defined through several equivalent approaches. The model-theoretic approach views $\mathcal{DP}$ as a set of first-order sentences and

uses the minimal Herbrand model as the semantics. We write $SP(\gamma^T) \models m(K, r, K')$ when $m(K, r, K')$ is in the minimal Herbrand model of $SP(\gamma^T)$.

State-change Rules A state-change rule is of the form $\psi^T = (\mathcal{G}, \mathcal{S})$, where $\mathcal{G}$ and $\mathcal{S}$ are finite sets of roles.

- Roles in $\mathcal{G}$ are called *growth-restricted* (or *g-restricted*); no statements defining these roles can be added. (A statement defines a role if it has the role to the left of '$\longleftarrow$'.) Roles not in $\mathcal{G}$ are called *growth-unrestricted* (or *g-unrestricted*).

- Roles in $\mathcal{S}$ are called *shrink-restricted* (or *s-restricted*); statements defining these roles cannot be removed. Roles not in $\mathcal{S}$ are called *shrink-unrestricted* (or *s-unrestricted*).

Queries Li et al. [5] considers the following three forms of queries:

- *Membership*: $A.r \sqsupseteq \{D_1, \ldots, D_n\}$

  Intuitively, this means that all the principals $D_1, \ldots, D_n$ are members of $A.r$. Formally, $\gamma^T \vdash A.r \sqsupseteq \{D_1, \ldots, D_n\}$ if and only if $\{Z \mid SP(\gamma^T) \models m(A, r, Z)\} \supseteq \{D_1, \ldots, D_n\}$.

- *Boundedness*: $\{D_1, \ldots, D_n\} \sqsupseteq A.r$

  Intuitively, this means that the member set of $A.r$ is bounded by the given set of principals. Formally, $\gamma^T \vdash \{D_1, \ldots, D_n\} \sqsupseteq A.r$ if and only if $\{D_1, \ldots, D_n\} \supseteq \{Z \mid SP(\gamma^T) \models m(A, r, Z)\}$.

- *Inclusion*: $X.u \sqsupseteq A.r$

  Intuitively, this means that all the members of $A.r$ are also members of $X.u$. Formally, $\gamma^T \vdash X.u \sqsupseteq A.r$ if and only if $\{Z \mid SP(\gamma^T) \models m(X, u, Z)\} \supseteq \{Z \mid SP(\gamma^T) \models m(A, r, Z)\}$.

Each form of query can be generalized to allow compound role expressions that use linking and intersection. These generalized queries can be reduced to the forms above by adding new roles and statements to the state. For instance, $\{\} \sqsupseteq A.r \cap A_1.r_1.r_2$ can

be answered by adding $B.u_1 \longleftarrow A.r \cap B.u_2$, $B.u_2 \longleftarrow B.u_3.r_2$, and $B.u_3 \longleftarrow A_1.r_1$ to γ^T, in which $B.u_1$, $B.u_2$, and $B.u_3$ are new g/s-restricted roles, and by posing the query $\{\} \sqsupseteq B.u_1$.

Main results for security analysis in $\mathsf{RT}[\twoheadleftarrow, \cap]$

Membership and boundedness queries (both whether a query is possible and whether a query is necessary) can be answered in time polynomial in the size of the input. The approach taken in [5] uses logic programs to derive answers to those security analysis problems. This approach exploits the fact that $\mathsf{RT}[\twoheadleftarrow, \cap]$ is monotonic in the sense that more statements will derive more role membership facts. This follows from the fact that the semantic program is a positive logic program.

Inclusion queries are more complicated than the other two kinds. In [5], only the $\forall$ case (i.e., whether an inclusion query is necessary) is studied. It is not clear what the security intuition is of an $\exists$ inclusion query (whether an inclusion query is possible); therefore, it is not studied in [5]. The problem of deciding whether an inclusion query is necessary, i.e., whether the set of members of one role is always a superset of the set of members of another role is called *containment analysis*. It turns out that the computational complexity of containment analysis depends on the language features. In $\mathsf{RT}[\,]$, the language that allows only simple member and simple inclusion statements, containment analysis is in $\mathbf{P}$. It becomes more complex when additional policy language features are used. Containment analysis is $\mathbf{coNP}$-complete for $\mathsf{RT}[\cap]$ ($\mathsf{RT}[\,]$ plus intersection inclusion statements), $\mathbf{PSPACE}$-complete for $\mathsf{RT}[\twoheadleftarrow]$ ($\mathsf{RT}[\,]$ plus linking inclusion statements), and decidable in $\mathbf{coNEXP}$ for $\mathsf{RT}[\twoheadleftarrow, \cap]$.

4.4 Reducing AATU and AAR to security analysis in $\mathsf{RT}[\twoheadleftarrow, \cap]$

In this section, we solve AATU (Definition 4.2.3) and AAR (Definition 4.2.4). Our approach is to reduce each of them to security analysis in $\mathsf{RT}[\twoheadleftarrow, \cap]$.

4.4.1 Reduction for AATU

The reduction algorithm AATU_Reduce is given in Figure 4.4; it uses the subroutines defined in Figure 4.3. Given an AATU instance $\langle \gamma = \langle UA, PA, RH \rangle,\ q = s_1 \sqsupseteq s_2,\ \psi = \langle can\_assign, T \rangle,\ \Pi \in \{\exists, \forall\} \rangle$, AATU_Reduce takes $\langle \gamma, q, \psi \rangle$ and outputs $\langle \gamma^T, q^T, \psi^T \rangle$ such that the RT$[\twoheadleftarrow, \cap]$ analysis instance $\langle \gamma^T, q^T, \psi^T, \Pi \rangle$ has the same answer as the original AATU instance.

In the reduction, we use one principal for every user that appears in γ, and the special principal Sys to represent the RBAC system. The RT$[\twoheadleftarrow, \cap]$ role names used in the reduction include the RBAC roles and permissions in γ and some additional temporary role names. The RT$[\twoheadleftarrow, \cap]$ role Sys.r represents the RBAC role r and the RT$[\twoheadleftarrow, \cap]$ role Sys.p represents the RBAC permission p. Each $(u, r) \in UA$ is translated into the RT$[\twoheadleftarrow, \cap]$ statement Sys.$r \longleftarrow u$. Each $r_1 \succeq r_2$ is translated into the RT$[\twoheadleftarrow, \cap]$ statement Sys.$r_2 \longleftarrow$ Sys.r_1 (as r_1 is senior to r_2, any member of r_1 is also a member of r_2.) Each $(p, r) \in PA$ is translated into Sys.$p \longleftarrow$ Sys.r (each member of the role r has the permission p.)

The translation of the can_assign relation is less straightforward. Each $\langle r_a, r_c, r \rangle \in can\_assign$ is translated into the RT$[\twoheadleftarrow, \cap]$ statement Sys.$r \longleftarrow$ Sys.$r_a.r \cap$ Sys.r_c. The intuition is that a user u_a who is a member of the role r_a assigning the user u to be a member of the r role is represented as adding the RT$[\twoheadleftarrow, \cap]$ statement $u_a.r \longleftarrow u$. As u_a is a member of the Sys.r_a role, the user u is added as a member to the Sys.r role if and only if the user u is also a member of the r_c role.

In the reduction, all the Sys roles (i.e., Sys.x) are fixed (i.e., both g-restricted and s-restricted). In addition, for each trusted user u in T, all the roles starting with u are also g-restricted; this is because we assume that trusted users will not perform operations to change the state (i.e., user-role assignment operations). We may also make roles starting with trusted users s-restricted; however, this has no effect as no statement defining these roles exists in the initial state.

```
 1 Subroutine Trans(s, γ^T) {
 2   /* Trans(s,γ^T) returns an RT[↞,∩] role corresponding
 3       to the user set s*/
 4   if s is an RBAC role then return Sys.s;
 5   else if s is an RBAC permission then return Sys.s;
 6   else if s is a set of users then {
 7     name=newName();  foreach u ∈ s {
 8       γ^T += Sys.name ⟵ u;}
 9       return Sys.name; }
10   else if (s = s1 ∪ s2) then {
11     name=newName();  γ^T +=Sys.name ⟵ Trans(s1, γ^T);
12     γ^T += Sys.name ⟵ Trans(s2, γ^T);
13     return Sys.name; }
14   else if (s = s1 ∩ s2) then {
15     name=newName();
16     γ^T +=Sys.name ⟵ Trans(s1, γ^T) ∩ Trans(s2, γ^T);
17     return Sys.name; }
18 } /* End Trans */
19
20 Subroutine QTrans(s, γ^T) {
21   /* Translation for users sets that are used at top
22      level in a query */
23   if s is a set of users then return s;
24   else return Trans(s,γ^T);
25 } /* End QTrans */
26
27 Subroutine HTrans(s, γ^T) {
28   if s is an RBAC role then return HSys.s;
29   else if (s = s1 ∪ s2) then {
30     name=newName();  γ^T += Sys.name ⟵ HTrans(s1, γ^T);
31     γ^T += Sys.name ⟵ HTrans(s2, γ^T); return Sys.name; }
32   else if (s = s1 ∩ s2) then {
33     name=newName();
34     γ^T +=Sys.name ⟵ HTrans(s1, γ^T) ∩ HTrans(s2, γ^T);
35     return Sys.name; }
36 } /* End HTrans */
```

Figure 4.3. Subroutines Trans, QTrans, and HTrans are used by the two reduction algorithms. We assume call-by-reference for the parameter γ^T.

```
37 AATU_Reduce (⟨ γ = ⟨UA, PA, RH⟩,  q = s1 ⊒ s2,  ψ = ⟨can_assign, T⟩ ⟩)
38 {
39   /* Reduction algorithm for AATU */
40
41   γ^T = ∅;  q^T = QTrans(s1, γ^T) ⊒ QTrans(s2, γ^T);
42   foreach (ui, rj) ∈ UA { γ^T += Sys.rj ⟵ ui; }
43   foreach (ri, rj) ∈ RH { γ^T += Sys.rj ⟵ Sys.ri; }
44   foreach (pi, rj) ∈ PA { γ^T += Sys.pi ⟵ Sys.rj; }
45   foreach (ai, s, rset) ∈ can_assign {
46     if (s==true) { foreach r ∈ rset {
47         γ^T += Sys.r ⟵ Sys.ai.r; } }
48     else { tmpRole=Trans(s, γ^T);
49       foreach r ∈ rset { name=newName();
50         γ^T += Sys.name ⟵ Sys.ai.r;
51         γ^T += Sys.r ⟵ Sys.name ∩ tmpRole
52   } } }
53   foreach RT role name x appearing in γ^T {
54     G+=Sys.x;  S+=Sys.x;  foreach user u ∈ T { G+=u.x; } }
54   return ⟨γ^T, q^T, (G, S)⟩;
55 } /* End AATU_Reduce */
```

Figure 4.4. Reduction Algorithm for AATU

Example 6 Consider the state-change rule ψ we discuss in Example 4, in which can_assign consists of the two tuples $\langle \mathsf{Manager}, \mathsf{Engineer} \wedge \mathsf{FullTime}, \mathsf{ProjectLead} \rangle$ and $\langle \mathsf{HumanResource}, true, \{\mathsf{FullTime}, \mathsf{PartTime}\} \rangle$, and $T = \{\mathsf{Carol}\}$. Let γ be the RBAC state shown in Figure 4.1, and let q be the query $\mathsf{ProjectLead} \sqsupseteq \mathsf{Alice}$. Then, we represent the output of $\mathsf{AATU\_Reduce}(\langle \gamma, q, \psi \rangle)$ as $\langle \gamma^T, q^T, \psi^T \rangle$. q^T is $\mathsf{Sys.ProjectLead} \sqsupseteq \{\mathsf{Alice}\}$. The following RT statements in γ^T result from UA:

$\mathsf{Sys.Engineer} \longleftarrow \mathsf{Alice}$ $\quad$ $\mathsf{Sys.PartTime} \longleftarrow \mathsf{Alice}$

$\mathsf{Sys.Manager} \longleftarrow \mathsf{Bob}$ $\quad$ $\mathsf{Sys.HumanResource} \longleftarrow \mathsf{Carol}$

The following statements in γ^T result from RH:

$\mathsf{Sys.Employee} \longleftarrow \mathsf{Sys.Engineer}$ $\quad$ $\mathsf{Sys.Employee} \longleftarrow \mathsf{Sys.FullTime}$

$\mathsf{Sys.Employee} \longleftarrow \mathsf{Sys.PartTime}$ $\quad$ $\mathsf{Sys.Engineer} \longleftarrow \mathsf{Sys.ProjectLead}$

$\mathsf{Sys.FullTime} \longleftarrow \mathsf{Sys.Manager}$

The following statements in γ^T result from PA:

$\mathsf{Sys.View} \longleftarrow \mathsf{Sys.HumanResource}$ $\quad$ $\mathsf{Sys.Access} \longleftarrow \mathsf{Sys.Employee}$

$\mathsf{Sys.Edit} \longleftarrow \mathsf{Sys.Engineer}$

The following statements in γ^T result from can_assign. The first two statements reflect the ability of a member of HumanResource to assign users to FullTime and PartTime with no precondition, and the remaining statements reflect the ability of a member of Manager to assign users to ProjectLead provided that they are already members of FullTime and Engineer.

$\mathsf{Sys.FullTime} \longleftarrow \mathsf{Sys.HumanResource.FullTime}$

$\mathsf{Sys.PartTime} \longleftarrow \mathsf{Sys.HumanResource.PartTime}$

$\mathsf{Sys.NewRole}_1 \longleftarrow \mathsf{Sys.Engineer} \cap \mathsf{Sys.FullTime}$

$\mathsf{Sys.NewRole}_2 \longleftarrow \mathsf{Sys.Manager.ProjectLead}$

$\mathsf{Sys.ProjectLead} \longleftarrow \mathsf{Sys.NewRole}_1 \cap \mathsf{Sys.NewRole}_2$

$\gamma^T = \langle G, S \rangle$, where G is the growth-restricted set of roles, and S is the shrink-restricted set of roles. G consists of every role of the form Sys.x and every role of the form Carol.x. The latter is included in G because Carol is in the set of trusted users T. S consists of every role of the form Sys.x. It is clear that the security analysis instance $\langle \gamma^T, q^T, \psi^T, \exists \rangle$ is false, as Alice can never become a member of Sys.ProjectLead. If we adopt as the state-change rule ψ_1^T, that is the same as ψ^T except that $T = \emptyset$, then roles of the form Carol.x would be growth-unrestricted. And there exists a state γ_1^T that is reachable from γ^T which has the following statements in addition to all the statements in γ^T.

Carol.FullTime $\longleftarrow$ Alice Bob.ProjectLead $\longleftarrow$ Alice

These statements are necessary and sufficient for Sys.ProjectLead $\longleftarrow$ Alice to be inferred in γ_1^T. Thus, the security analysis instance $\langle \gamma^T, q^T, \psi_1^T, \exists \rangle$ is true.

The following proposition asserts that the reduction is sound, meaning that one can use RT security analysis techniques to answer RBAC security analysis problems.

Proposition 4.4.1 *Given an AATU instance* $\langle \gamma, q, \psi, \Pi \rangle$, *let* $\langle \gamma^T, q^T, \psi^T \rangle = \mathsf{AATU\_Reduce}(\langle \gamma, q, \psi \rangle)$, *then:*

- Assertion 1: *For every RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$, *there exists an* RT$[\twoheadleftarrow, \cap]$ *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$ *and* $\gamma' \vdash q$ *if and only if* $\gamma^{T\prime} \vdash q^T$.

- Assertion 2: *For every* RT$[\twoheadleftarrow, \cap]$ *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$, *there exists an RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$ *and* $\gamma' \vdash q$ *if and only if* $\gamma^{T\prime} \vdash q^T$.

Proof *For Assertion 1:* A state change in AATU occurs when a user assignment operation is successfully performed. For every RBAC state γ' such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$, let $\gamma_0, \gamma_1, \cdots, \gamma_m$ be RBAC states such that $\gamma = \gamma_0 \mapsto_{\psi} \gamma_1 \mapsto_{\psi} \cdots \mapsto_{\psi} \gamma_m = \gamma'$. We construct a sequence of RT$[\twoheadleftarrow, \cap]$ states $\gamma_0^T, \gamma_1^T, \cdots, \gamma_m^T$ as follows: $\gamma_0^T = \gamma^T$; for each $i = [0..m-1]$, consider the assignment operation that changes γ_i to γ_{i+1}, let it be the operation in which a user

u_1 adds (u, r) to the user-role assignment relation; the state γ_{i+1}^T is obtained by adding $u_1.r \longleftarrow u$ to γ_i^T. Let $\gamma^{T\prime}$ be γ_m^T.

Step one: Prove that if $\gamma' \vdash q$ then $\gamma^{T\prime} \vdash q^T$. It is sufficient to prove the following: for each $i \in [0..m]$, if γ_i implies that a certain user u is a member of a role r (or has the permission p), then γ_i^T implies that u is a member of the $\mathsf{RT}[\twoheadleftarrow, \cap]$ role $\mathsf{Sys}.r$ (or $\mathsf{Sys}.p$). We use induction on i to prove this. The base case (i=0) follows directly from the AATU_Reduce algorithm; lines 42–44 reproduces UA, RH, PA in the $\mathsf{RT}[\twoheadleftarrow, \cap]$ state γ_0^T. For the step, assumes that the induction hypothesis holds for $\gamma_0, \cdots, \gamma_i$, consider γ_{i+1}. Let the operation leading to γ_{i+1} be one in which u_1 assigns u to a role r. As both sequences of states are increasing, we only need to consider role memberships implied by γ_{i+1} but not γ_i; these are caused (directly or indirectly) by this assignment. There must exist a $\langle r_a, c, r\rangle \in can\_assign$ to enable this assignment; thus in γ_i, u_1 is a member of the role r_a and u satisfies the condition c. By induction hypothesis, in γ_i^T, u_1 is a member of $\mathsf{Sys}.r_a$ and u satisfies the condition c. From the translation and the construction of γ_{i+1}^T, γ_{i+1}^T has the following statements: $u_1.r \longleftarrow u$, $\mathsf{Sys}.r \longleftarrow \mathsf{Sys}.r_a.r$, and $\mathsf{Sys}.r \longleftarrow \mathsf{Sys}.name \cap tmpRole$ (where $tmpRole$ corresponds to the precondition c). Furthermore, in γ_{i+1}^T, u_1 is a member of the role r_a and u satisfies the condition c. Therefore, u is a member of the $\mathsf{Sys}.r$ role in γ_{i+1}^T.

Step two: Prove that if $\gamma^{T\prime} \vdash q^T$ then $\gamma' \vdash q$. It is sufficient to show that if an $\mathsf{RT}[\twoheadleftarrow, \cap]$ role membership is implied by $\gamma^{T\prime}$, then the corresponding RBAC role membership (or permission possession) is also implied. A detailed proof uses induction on the number of rounds in which a bottom-up datalog evaluation algorithm outputs a ground fact. Here, we only point out the key observations. (For details of similar proofs, see the Appendix in [5].) A $\mathsf{RT}[\twoheadleftarrow, \cap]$ role membership is proved by statements generated on lines 42–52. The first three cases correspond to the UA, RH, PA. For the last case, there must exist a statement $u_1.r \longleftarrow u$ in $\gamma^{T\prime}$, and it implies that u is a member of the role $\mathsf{Sys}.r$. By the construction of $\gamma^{T\prime}$, the user u has been assigned to the role r during the changes leading to γ'.

For Assertion 2: Given an RT$[\twoheadleftarrow, \cap]$ state $\gamma^{T\prime}$ such that $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$, we can assume without loss of generality that $\gamma^{T\prime}$ adds to γ^T only simple member statements. Also, we only need to consider statements defining $u_i.r_j$, where u_i is a user in γ and r_j is a role in γ. Consider the set of all statements in $\gamma^{T\prime}$ having the form $u_i.r_j \longleftarrow u_k$. For each such statement, we perform the following operation on the RBAC state, starting from γ, have u_i assign u_k to the role r_j. Such an operation may not succeed either because u_i is not in the right administrative role or because u_k does not satisfy the required precondition. We repeat to perform all operations that could be performed. That is, we loop through all such statements and repeat the loop whenever the last loop results in a new successful assignment. Let γ' be the resulting RBAC state. It is not difficult to see that γ' implies the same role memberships as $\gamma^{T\prime}$; using arguments similar to those used above. ■

As we discuss in detail in [71], the above proposition asserts that AATU_Reduce is security (analysis) preserving in the sense that an AATU analysis instance is true if and only if the RT$[\twoheadleftarrow, \cap]$ analysis instance that is the output of AATU_Reduce is true. That is, AATU_Reduce preserves the answer to every security analysis instance. We argue the need for assertion 1 in the proposition by considering the case that there exists a reachable state γ' in the RBAC system, but no corresponding reachable state $\gamma^{T\prime}$ in the RT$[\twoheadleftarrow, \cap]$ system produced by AATU_Reduce. Let the corresponding query be q. If $\gamma' \vdash q$, then let Π be $\exists$, and if $\gamma' \nvdash q$, then let Π be $\forall$. In the former case, the security analysis instance in RBAC is true, but the instance in the RT$[\twoheadleftarrow, \cap]$ system that is the output of AATU_Reduce is false. In the latter case, the analysis instance in RBAC is false, but the instance in RT$[\twoheadleftarrow, \cap]$ is true. Therefore, for AATU_Reduce to preserve the answer to every analysis instance, we need assertion 1.

Similarly, we argue the need for assertion 2 by considering the contrary situation. Let $\gamma^{T\prime}$ be a reachable state in RT$[\twoheadleftarrow, \cap]$ for which there exists no corresponding state in RBAC. Let the corresponding query in RT$[\twoheadleftarrow, \cap]$ be q^T. If $\gamma^{T\prime} \vdash q^T$, then let Π be $\exists$, and let Π be $\forall$ otherwise. Again, AATU_Reduce would not preserve the answer to a security analysis instance, and we would not be able to use the answer to an analysis instance in RT$[\twoheadleftarrow, \cap]$ as the answer to the corresponding instance in RBAC.

Theorem 4.4.2 *An AATU instance $\langle \gamma, q, \psi, \Pi \rangle$ can be solved efficiently, i.e., in time polynomial in the size of the instance, if q is semi-static.*

Proof Let the output of AATU_Reduce corresponding to the input $\langle \gamma, q, \psi \rangle$ be $\langle \gamma^T, q^T, \psi^T \rangle$. If q is semi-static, we observe that q^T is semi-static as well. Furthermore, AATU_Reduce runs in time polynomial in its input. We know from Li et al. [5] that in RT$[\leftarrow, \cap]$, a security analysis instance with a semi-static query can be answered in time polynomial in the size of γ^T. Therefore, in conjunction with Proposition 4.4.1, we can conclude that a security analysis instance with a semi-static query in the RBAC system can be answered in time polynomial in the size of the system (i.e., the size of $\langle \gamma, q, \psi \rangle$). ■

Theorem 4.4.3 *An AATU instance $\langle \gamma, q, \psi, \Pi \rangle$ is* **coNP***-hard.*

Proof We show that the general AATU problem is **coNP**-hard by reducing the monotone 3SAT problem to the complement of the AATU problem. Monotone 3SAT is the problem of determining whether a boolean expression in conjunctive normal form with at most three literals in each clause such that the literals in a clause are either all positive or all negative, is satisfiable. Monotone 3SAT is known to be **NP**-complete [72].

Let ϕ be an instance of monotone 3SAT. Then $\phi = c_1 \wedge \ldots c_l \wedge \overline{c_{l+1}} \wedge \ldots \wedge \overline{c_n}$ where $c_1, \ldots, c_l$ are the clauses with positive literals, and $\overline{c_{l+1}}, \ldots, \overline{c_n}$ are the clauses with negative literals. Let $p_1, \ldots, p_s$ be all the propositional variables in ϕ. For each clause with negative literals $\overline{c_k} = (\neg\, p_{k_1} \vee \neg\, p_{k_2} \vee \neg\, p_{k_3})$, define $d_k = \neg\, \overline{c_k} = (p_{k_1} \wedge p_{k_2} \wedge p_{k_3})$. Then, ϕ is satisfiable if and only if $c_1 \wedge \ldots c_l \wedge \neg\,(d_{l+1} \vee \ldots \vee d_n)$ is satisfiable. Let $\eta = (c_1 \wedge \ldots \wedge c_l) \rightarrow (d_{l+1} \vee \ldots \vee d_n)$ where $\rightarrow$ is logical implication. Then, $c_1 \wedge \ldots c_l \wedge \neg\,(d_{l+1} \vee \ldots \vee d_n) = \neg\, \eta$. Therefore, ϕ is satisfiable if and only if η is not valid. We now construct γ, ψ and q in an AATU instance such that $q = z_1 \sqsupseteq z_2$ is true for user sets z_1 and z_2 in all states reachable from γ if and only if η is valid.

In γ, we have a role a (which is for administrators) and UA contains (A, a) where A is a user (i.e., the role a is not empty in terms of user-membership). With each propositional variable p_i in η, we associate a role r_i. For each r_i, we add $\langle a, true, r_i \rangle$ to can_assign. That is, anyone can be assigned to the role r_i. We let T (the set of trusted users) be empty.

For each j such that $1 \leq j \leq l$, we associate the clause $c_j = (p_{j_1} \vee p_{j_2} \vee p_{j_3})$, with a user set $s_j = (r_{j_1} \cup r_{j_2} \cup r_{j_3})$. For each k such that $(l+1) \leq k \leq n$, we associate the clause $d_k = (p_{k_1} \wedge p_{k_2} \wedge p_{k_3})$, with a user set $s_k = (r_{k_1} \cap r_{k_2} \cap r_{k_3})$. In our query $q = z_1 \sqsupseteq z_2$, we let $z_1 = s_{l+1} \cup \ldots \cup s_n$ and $z_2 = s_1 \cap \ldots \cap s_l$. We now need to show that $z_1 \sqsupseteq z_2$ in every state reachable from γ if and only if η is valid. We show that $z_1 \sqsupseteq z_2$ is *not* true in every state reachable from γ if and only if η is *not* valid.

For the "only if" part, we assume that there exists a state γ' that is reachable from γ such that in γ' there exists a user u that is a member of the user set z_2, but not z_1. Consider a truth-assignment I for the propositional variables in η as follows: if u is a member of the role r_i in γ', then $I(p_i) = true$. Otherwise, $I(p_i) = false$. Under I, η is not true, as $(c_1 \wedge \ldots \wedge c_l)$ is true, but $(d_{l+1} \vee \ldots \vee d_n)$ is false. Therefore, η is not valid.

For the "if" part, we assume that η is not valid. Therefore, there exists a truth-assignment I such that $(c_1 \wedge \ldots \wedge c_l)$ is true, but $(d_{l+1} \vee \ldots \vee d_n)$ is false. Consider a state γ' that has the following members in UA in addition to the ones in γ: for each p_i that is true under I, $(u, r_i) \in UA$. Otherwise, $(u, r_i) \notin UA$. γ' is reachable from γ, and in γ', $z_1 \sqsupseteq z_2$ is not true. ■

We observe from the above proof that the AATU problem remains **coNP**-hard even when every precondition that occurs in can_assign is specified as $true$; the expressive power of the queries is sufficient for reducing the monotone 3SAT problem to the general AATU problem. We infer from our reduction and results from $\mathsf{RT}[\twoheadleftarrow, \cap]$ that an AATU instance is in **PSPACE**.

4.4.2 Reduction for AAR

The reduction algorithm for AAR is given in Figure 4.5. The reduction algorithm includes in the set of principals a principal for every user in U and five special principals: Sys, RSys, HSys, ASys, and BSys. Again, the Sys roles simulate RBAC roles and permissions. In this reduction, we do not distinguish whether a role assignment operation is effected by one user or another, and use only one principal, ASys, to represent every user

```
AAR_Reduce (⟨ γ = ⟨UA, PA, RH⟩,  q = s1 ⊒ s2,
             ψ = ⟨can_assign, can_revoke⟩ ⟩)
{ /* Reduction algorithm for AAR */
  γ^T = ∅;   q^T = QTrans(s1, γ^T) ⊒ QTrans(s2, γ^T);
  foreach (ui, rj) ∈ UA {
    γ^T += HSys.rj ⟵ ui; γ^T += RSys.rj ⟵ ui;
    γ^T += Sys.rj ⟵ RSys.rj; }
  foreach (ri, rj) ∈ RH {
    γ^T += Sys.rj ⟵ Sys.ri; γ^T += HSys.rj ⟵ HSys.ri; }
  foreach (pi, rj) ∈ PA { γ^T += Sys.pi ⟵ Sys.rj; }
  foreach (ai, s, rset) ∈ can_assign {
    if (s==true) {
      foreach r ∈ rset {
        γ^T += HSys.r ⟵ BSys.r; γ^T += Sys.r ⟵ ASys.r; }
    } else { tmpRole = HTrans(s, γ^T);  /* precondition */
      foreach r ∈ rset {
        γ^T += HSys.r ⟵ BSys.r ∩ tmpRole;
        γ^T += Sys.r ⟵ ASys.r ∩ tmpRole; }
  } }
  foreach RT role name x appearing in γ^T {
    G+=Sys.x; S+=Sys.x; G+=HSys.x; S+=HSys.x; G+=RSys.x;
    S+=BSys.x; S+=RSys.x; S+=ASys.x;
  } /* when a can_revoke rule exists for r, ASys.r and
       RSys.r can shrink */
  foreach (ai, rset) ∈ can_revoke {
    foreach r in rset { S-=RSys.r; S-=ASys.r; } }
  return ⟨γ^T, q^T, (G, S)⟩;
} /* End AAR_Reduce */
```

Figure 4.5. AAR_Reduce: the reduction algorithm for AAR

that exercises the user-role assignment operation. The roles of the principal RSys contain all the initial role memberships in UA; these may be revoked in state changes. HSys.r maintains the history of the RBAC role r; its necessity is argued using the following scenario. A user is a member of r_1, which is the precondition for being added to another role r_2. After one assigns the user to r_2 and revokes the user from r_1 the user's membership in r_2 should be maintained, even though the precondition is no longer satisfied (a similar justification for this approach is provided in the context of ARBAC97 [37] as well). BSys is similar to ASys, but it is used to construct the HSys roles. An administrative operation to try to add a user u_i to the role r_j is represented by adding the statement ASys.$r_j \longleftarrow u_i$ and BSys.$r_j \longleftarrow u_i$ to γ^T. An administrative operation to revoke a user u_i from the role r_j is represented by removing the statements RSys.$r_j \longleftarrow u_i$ and ASys.$r_j \longleftarrow u_i$ if either exists in γ^T.

Example 7 Consider the state-change rule ψ we discuss in Example 5, in which can_assign consists of the two tuples $\langle \mathsf{Manager}, \mathsf{Engineer} \wedge \mathsf{FullTime}, \mathsf{ProjectLead} \rangle$ and $\langle \mathsf{HumanResource}, true, \{\mathsf{FullTime}, \mathsf{PartTime}\} \rangle$, and can_revoke consists of the two tuples $\langle \mathsf{Manager}, \{\mathsf{Engineer}, \mathsf{ProjectLead}\} \rangle$ and $\langle \mathsf{HumanResource}, \{\mathsf{FullTime}, \mathsf{PartTime}\} \rangle$. Let γ be the RBAC state shown in Figure 4.1, and let q be the query ProjectLead $\sqsupseteq$ Alice. Then, we represent the output of AATU_Reduce $(\langle \gamma, q, \psi \rangle)$ as $\langle \gamma^T, q^T, \psi^T \rangle$. q^T is Sys.ProjectLead $\sqsupseteq$ {Alice}. The following RT statements in γ^T result from UA:

HSys.Engineer $\longleftarrow$ Alice
RSys.Engineer $\longleftarrow$ Alice
HSys.PartTime $\longleftarrow$ Alice
RSys.PartTime $\longleftarrow$ Alice
HSys.Manager $\longleftarrow$ Bob
RSys.Manager $\longleftarrow$ Bob
HSys.HumanResource $\longleftarrow$ Carol
RSys.HumanResource $\longleftarrow$ Carol
Sys.Engineer $\longleftarrow$ RSys.Engineer
Sys.FullTime $\longleftarrow$ RSys.FullTime
Sys.HumanResource $\longleftarrow$ RSys.HumanResource
Sys.PartTime $\longleftarrow$ RSys.PartTime

The following statements in γ^T result from RH:

Sys.Employee $\longleftarrow$ Sys.Engineer
HSys.Employee $\longleftarrow$ HSys.Engineer

Sys.Employee ⟵ Sys.FullTime HSys.Employee ⟵ HSys.FullTime

Sys.Employee ⟵ Sys.PartTime HSys.Employee ⟵ HSys.PartTime

Sys.Engineer ⟵ Sys.ProjectLead HSys.Engineer ⟵ HSys.ProjectLead

Sys.FullTime ⟵ Sys.Manager HSys.FullTime ⟵ HSys.Manager

The following statements in γ^T result from PA:

Sys.View ⟵ Sys.HumanResource Sys.Access ⟵ Sys.Employee

Sys.Edit ⟵ Sys.Engineer

The following statements in γ^T result from can_assign:

HSys.FullTime ⟵ BSys.FullTime Sys.FullTime ⟵ ASys.FullTime

HSys.PartTime ⟵ BSys.PartTime Sys.PartTime ⟵ ASys.PartTime

$\mathsf{Sys.NewRole}_1 \longleftarrow \mathsf{HSys.Engineer} \cap \mathsf{HSys.FullTime}$

$\mathsf{HSys.ProjectLead} \longleftarrow \mathsf{BSys.ProjectLead} \cap \mathsf{Sys.NewRole}_1$

$\mathsf{Sys.ProjectLead} \longleftarrow \mathsf{ASys.ProjectLead} \cap \mathsf{Sys.NewRole}_1$

$\psi^T = \langle G, S \rangle$, where G is the growth-restricted set of roles, and S is the shrink-restricted set of roles. Unlike can_assign, can_revoke results only in some roles not being added to S. G is comprised of all roles of the form Sys.x, HSys.x and RSys.x (but not BSys.x or ASys.x). S is comprised of all roles of the form Sys.x, HSys.x, RSys.x and ASys.x, except the roles RSys.Manager, ASys.Manager, RSys.Engineer, ASys.Engineer, RSys.FullTime, ASys.FullTime, RSys.PartTime, and ASys.PartTime. This is because those roles appear in can_revoke rules, and therefore may shrink.

There exists a state γ_1^T that is reachable from γ^T that has the following statements in addition to the ones in γ^T.

BSys.FullTime ⟵ Alice ASys.ProjectLead ⟵ Alice

We can now infer that in γ_1^T, HSys.FullTime ⟵ Alice, and therefore, $\mathsf{HSys.NewRole}_1 \longleftarrow$ Alice, and so, Sys.ProjectLead ⟵ Alice. Thus, the security analysis instance

$\langle \gamma^T, q^T, \psi^T, \exists \rangle$ is true. If we consider, instead, the query q_1^T which is Sys.PartTime $\sqsupseteq$ Alice, then as RSys.PartTime is a shrink-unrestricted role, there exists a state γ_2^T that is reachable from γ^T in which the statement RSys.PartTime $\longleftarrow$ Alice is absent. Therefore, we would conclude that Sys.ProjectLead does not include Alice. Consequently, the analysis instance $\langle \psi^T, q_1^T, \gamma^T, \forall \rangle$ is false.

We are able to also demonstrate the need for the roles associated with the principals HSys and BSys. Consider the state, γ_2^T that can be reached from γ_1^T by removing the statement RSys.FullTime $\longleftarrow$ Alice. Now, Sys.FullTime does not include Alice. This is equivalent to Carol revoking the membership of the user Alice to the role FullTime. This affects the precondition that one can be assigned to the role ProjectLead only if one is already a member of the roles Engineer and FullTime. Nonetheless, we observe that $\gamma_2^T \vdash q^T$, as indeed it should. That is, Alice should continue to be a member of ProjectLead even if subsequent to her becoming a member of ProjectLead, her membership is removed from FullTime. We observe that this is the case because the role BSys.FullTime is shrink-restricted, and therefore one cannot remove the statement BSys.FullTime $\longleftarrow$ Alice once it has been added, and consequently, HSys.FullTime $\longleftarrow$ Alice is true, and therefore Alice continues to be a member of the role ProjectLead (i.e., is included in Sys.ProjectLead). Of course, Alice can later have her membership revoked from the role ProjectLead (by Bob), and this is equivalent to the statement ASys.ProjectLead $\longleftarrow$ Alice being removed.

The following proposition asserts that the reduction is sound.

Proposition 4.4.4 *Given an AAR instance* $\langle \gamma, q, \psi, \Pi \rangle$*, let* $\langle \gamma^T, q^T, \psi^T \rangle = \mathsf{AAR\_Reduce}(\langle \gamma, q, \psi \rangle)$*, then:*

- Assertion 1: *For every RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$*, there exists an* RT$[\twoheadleftarrow, \cap]$ *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$ *and* $\gamma' \vdash q$ *if and only if* $\gamma^{T\prime} \vdash q^T$*.*
- Assertion 2: *For every* RT$[\twoheadleftarrow, \cap]$ *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$*, there exists an RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$ *and* $\gamma' \vdash q$ *if and only if* $\gamma^{T\prime} \vdash q^T$*.*

Proof *For Assertion 1:* A state change in AAR occurs when a user assignment or a revocation operation is successfully performed. Given any RBAC state γ' such that $\gamma \overset{*}{\mapsto}_{\psi}$

γ', let $\gamma_0, \gamma_1, \cdots, \gamma_m$ be RBAC states such that $\gamma = \gamma_0 \mapsto_\psi \gamma_1 \mapsto_\psi \cdots \mapsto_\psi \gamma_m = \gamma'$. We construct a sequence of RT$[\twoheadleftarrow, \cap]$ states $\gamma_0^T, \gamma_1^T, \cdots, \gamma_m^T$ as follows: $\gamma_0^T = \gamma^T$; for each $i = [0..m-1]$, consider the operation that changes γ_i to γ_{i+1}. If it is an assignment operation in which a user u_1 adds (u, r) to the user-role assignment relation; the state γ_{i+1}^T is obtained by adding Sys.$r \longleftarrow u$ and BSys.$r \longleftarrow u$ to γ_i^T. For each revocation that revokes a user u from a role r, we remove (if they exist) from the RT$[\twoheadleftarrow, \cap]$ state the statements ASys.$r \longleftarrow u$ and $RSys.r \longleftarrow u$. Let $\gamma^{T\prime}$ be γ_m^T.

Step 1: Prove that if $\gamma' \vdash q$ then $\gamma^{T\prime} \vdash q^T$. *Step 1a:* We prove that in $\gamma^{T\prime}$, HSys.r captures all users that are ever a member of the role r at some time, i.e., for each $i \in [0..m]$, if $u \in \text{users}_{\gamma_i}[r]$, then u is a member of the RT$[\twoheadleftarrow, \cap]$ role HSys.r in γ_m^T $\big(SP(\gamma_m^T) \models m(\text{HSys}, r, u)\big)$. We prove this by induction on i. The basis ($i = 0$) is true, because in γ^T we reproduce UA and RH in the definition of the HSys roles (see lines 60–64 in Figure 4.5); furthermore, the HSys roles never shrink. For the step, we show that if $(u, r) \in UA_{i+1}$, then u is a member of the RT$[\twoheadleftarrow, \cap]$ role HSys.r in γ_m^T. This is sufficient for proving the induction hypothesis because the effect of propagation through role hierarchy is captured by the definition of HSys roles. If $(u, r) \in UA_{i+1}$, then either $(u, r) \in UA$ (in which case HSys.$r \longleftarrow u \in \gamma^{T\prime}$), or there is an assignment operation that assigns u to r (in which case BSys.$r \longleftarrow u \in \gamma^{T\prime}$). Let $(r_a, c, r) \in can\_assign$ be an administrative rule used for this assignment, then in γ_i, the user u satisfies c. By induction hypothesis u's role memberships in γ_i is captured in u's role memberships in HSys.r; therefore u would satisfy the translated precondition $tmpRole$. Therefore u is a member of the role HSys.r in γ_m^T (because of the statement HSys.$u \longleftarrow$ BSys.$r \cap tmpRole$).

Step 1b: We prove that in $\gamma^{T\prime}$ the Sys roles capture all the role memberships in γ'. It is sufficient to prove the following: let UA' be the user assignment relation in γ', if $(u, r) \in UA'$, then u is a member of the role Sys.r in $\gamma^{T\prime}$. If $(u, r) \in UA$, then either $(u, r) \in UA$ and this is never revoked (in which case RSys.$r \longleftarrow u \in \gamma^T$ and this statement is never removed, therefore RSys.$r \longleftarrow u \in \gamma^{T\prime}$); or there is an assignment operation in C, and this assignment is not revoked after it (in which case ASys.$r \longleftarrow u \in \gamma^{T\prime}$).

Step two: Prove that if $\gamma^{T\prime} \vdash q^T$ then $\gamma' \vdash q$. It is sufficient to show that if an $\mathsf{RT}[\twoheadleftarrow, \cap]$ role membership is implied by $\gamma^{T\prime}$, then the corresponding RBAC role membership (or permission possession) is also implied. A detailed proof uses induction on the number of rounds in which a bottom-up datalog evaluation algorithm outputs a ground fact. Here, we only point out the key observation. A $\mathsf{RT}[\twoheadleftarrow, \cap]$ role membership is proved by statements generated on lines 60–65 or 71–74. The first three cases correspond to the UA, RH, PA. For the last case, there must exist a statement $\mathsf{ASys}.r \longleftarrow u$ in $\gamma^{T\prime}$, and it implies that u is a member of the role $\mathsf{Sys}.r$. By the construction of $\gamma^{T\prime}$, the user u has been assigned to the role r during the changes leading to γ' and the assignment is not revoked after that.

Also, we only need to consider statements defining $u_i.r_j$, where u_i is a user in γ and r_j is a role in γ. Consider the set of all statements in $\gamma^{T\prime}$ having the form $u_i.r_j \longleftarrow u_k$. For each such statement, we perform the following operation on the RBAC state, starting from γ, have u_i assign u_k to the role r_j. Such an operation may not succeed either because u_i is not in the right administrative role or because u_k does not satisfy the required precondition. We repeat to perform all operations that could be performed. That is, we loop through all such statements and repeat the loop whenever the last loop results in a new successful assignment. Let γ' be the resulting RBAC state. It is not difficult to see that γ' implies the same role memberships as $\gamma^{T\prime}$; using arguments similar to those used above.

For Assertion 2: Among the $\mathsf{RT}[\twoheadleftarrow, \cap]$ roles, Sys roles and HSys roles are fixed; ASys roles can grow or shrink; RSys roles can shrink but not grow; and BSys roles can grow but not shrink. Given an $\mathsf{RT}[\twoheadleftarrow, \cap]$ state $\gamma^{T\prime}$ such that $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$, we can assume without loss of generality that $\gamma^{T\prime}$ adds to γ^T only simple member statements. Consider the set of all statements in $\gamma^{T\prime}$ defining ASys, BSys, and RSys roles. We construct the RBAC state γ' as follows. (1) For every statement $\mathsf{BSys}.r \longleftarrow u$ in $\gamma^{T\prime}$, assign the user u to the role r. Repeat through all such statements until no new assignment succeeds. Using arguments similar to those used for proving assertion 1, it can be shown that now the RBAC roles have the same memberships as the HSys roles. (2) Do the same thing for all the $\mathsf{ASys}.r \longleftarrow u$ statements. At this point, all the role memberships for the Sys roles in $\gamma^{T\prime}$ are replicated in the RBAC roles, because all the HSys memberships have been added. (3) Remove the

extra role membership in the RBAC state, i.e., those not in the Sys roles. The ability to carry out this step depends upon the requirement (in Definition 4.2.4) that if there is a can_assign rule for a role, then there is also revoke rule for the role. ■

Our comments regarding the need for assertions 1 and 2 to preserve answers to security analysis instances, that we make in the previous section in the context of AATU_Reduce, apply to the above proposition in the context of AAR_Reduce as well. If either of the assertions does not hold, then we cannot use the answer to the RT$[\twoheadleftarrow, \cap]$ analysis instance as the answer to the corresponding RBAC instance.

Theorem 4.4.5 *An AAR instance $\langle \gamma, q, \psi, \Pi \rangle$ can be solved efficiently, i.e., in time polynomial in the size of the instance, if q is semi-static.*

Proof Let the output of AAR_Reduce for the input $\langle \gamma, q, \psi \rangle$ be $\langle \gamma^T, q^T, \psi^T \rangle$. If q is semi-static, so is q^T. As AAR_Reduce runs in time polynomial in its input and q^T can be answered in time polynomial in the size of γ^T (which is shown by Li et al. [5]), q can be answered in time polynomial in the size of the system (i.e., the size of $\langle \gamma, q, \psi \rangle$). Thus, an AAR instance with a semi-static query can be solved efficiently. ■

Theorem 4.4.6 *An AAR instance $\langle \gamma, q, \psi, \Pi \rangle$ is* $\mathbf{coNP}$*-complete.*

Proof We deduce that an AAR instance is in $\mathbf{coNP}$ from the fact that AAR_Reduce runs in time polynomial in the size of the system, and the corresponding security analysis problem in the RT$[\cap]$ system that is the output of AAR_Reduce is $\mathbf{coNP}$-complete. (RT$[\cap]$ is a sub-language of RT$[\twoheadleftarrow, \cap]$ that allows only the first, second and fourth kinds of statements from Figure 4.2.) That is, if q is not true in every state reachable from γ, then we offer as counterproof the algorithm AAR_Reduce and the counterproof in the RT$[\twoheadleftarrow, \cap]$ system that q^T is not true in every state reachable from γ^T.

We can show that the general AAR problem is $\mathbf{coNP}$-hard in almost exactly the same way that we show the result for the AATU problem in the proof for Theorem 4.4.3. The only difference is that for every role r_i that is associated with a propositional variable p_i, apart from a rule in can_assign, we add the rule $\langle a, r_i \rangle$ to can_revoke. We construct the

query q the same way as in that proof, and show in the same way that q is true in every state reachable from γ if and only if η is valid. ■

5 COMPARING THE EXPRESSIVE POWER OF ACCESS CONTROL MODELS

In this chapter, we introduce a theory for comparing access control models based on two notions of reductions that we call state-matching reductions and reductions, together with detailed justifications for the design decisions. We analyze the deficiency of using the implementation paradigm to compare access control models and show that it leads to a weak notion of simulations and cannot be used to differentiate access control models from one another based on expressive power. Also, we apply our theory in four cases. We show that:

- there exists no state-matching reduction from a rather simple trust-management scheme, $\mathsf{RT}[\,]$ [73], to the HRU scheme [2]. This is the first formal evidence of the limited expressive power of the HRU scheme. Li et al. [6] show that, contrary to the undecidability result of safety analysis in the HRU scheme, safety analysis and more sophisticated security analysis in the trust management scheme, $\mathsf{RT}[\twoheadleftarrow, \cap]$, is decidable. Li et al. [6] conjecture that these schemes cannot be encoded in the HRU scheme and that the expressive powers of the HRU scheme and of $\mathsf{RT}[\,]$ are incomparable. In this chapter, we present formal evidence for this assertion.

- there exists a reduction, but no state-matching reduction from a rather simple DAC scheme, Strict DAC with Change of Ownership (SDCO), to RBAC with ARBAC97 [37] as the administrative model. Several authors [11, 55] have argued that RBAC is more expressive than various forms of DAC, including SDCO. This is the first evidence of the limited expressive power of an RBAC scheme in comparison to DAC.

- there exists a state-matching reduction from RBAC with an administrative scheme that is a component of ARBAC97 [37] to $\mathsf{RT}[\cap]$ [69,74], a trust-management scheme.

This shows that state-matching reductions can be constructed for powerful access control schemes in the literature.

- there exists no state-matching reduction from ATAM to TAM, when we permit queries in ATAM that check for both the absence and the presence of a right in a cell. This revisits the issue addressed by Sandhu and Ganta [18] and formalizes the benefit from the ability to check for the absence of rights in addition to the ability to check for the presence of rights.

5.1 Comparisons based on security analysis

A methodology that can be used for comparing two systems is simulation. A requirement used in the literature for simulations is the preservation of simple safety properties. Indeed, this is the only requirement on simulations in [10, 17, 18]. If a simulation of scheme A in scheme B satisfies this requirement, then a system in A reaches an unsafe state if and only if the system's mapping in B reaches an unsafe state. In other words, the result of simple safety analysis is preserved by the simulation.

Simple safety analysis, i.e., determining whether an access control system can reach a state in which an unsafe access is allowed, was first formalized by [2] in the context of the well-known access matrix model [3, 4]. In the HRU scheme [2], a protection system has a finite set of rights and a finite set of commands. A state of a protection system is an access control matrix, with rows corresponding to subjects, and columns corresponding to objects; each cell in the matrix is a set of rights. A command takes the form of "if the given conditions hold in the current state, execute a sequence of primitive operations." Each condition tests whether a right exists in a cell in the matrix. There are six kinds of primitive operations: enter a right into a specific cell in the matrix, delete a right from a cell in the matrix, create a new subject, create a new object, destroy an existing subject, and destroy an existing object. The following is an example command that allows the owner of a file to grant the read right to another user.

```
command grantRead(u1,u2,f)
 if  'own' in (u1,f)
 then enter 'read' into (u2,f)
end
```

In the example, `u1`, `u2` and `f` are formal parameters to the command. They are instantiated by objects (or subjects) when the command is executed. Harrison et al. [2] proves that in the HRU scheme, the safety question is undecidable, by showing that any Turing machine can be simulated by a protection system.

Treating the preservation of simple safety properties as the sole requirement of simulations is based on the implicit assumption that simple safety is the *only* interesting property in access control schemes, an assumption that is not valid. When originally introduced by [2], simple safety was described as only one class of queries one can consider. Li et al. [5, 73] have introduced the notion of security analysis, which generalizes simple safety to other properties such as bounded safety, simple availability, mutual exclusion and containment.

In this section, we present a theory for comparing access control models based on the preservation of security properties. We adopt the definitions of access control schemes and security analysis from the previous chapter. We now introduce a generalized notion of security analysis.

Definition 5.1.1 *(*Compositional Security Analysis*)* Given a scheme $\langle \Gamma, Q, \vdash, \Psi \rangle$, a *compositional security analysis* instance has the form $\langle \gamma, \varphi, \psi, \Pi \rangle$, where γ, ψ, and Π are the same as in a security analysis instance, and φ is a propositional formula over Q, i.e., φ is constructed from queries in Q using propositional logic connectives such as $\wedge$, $\vee$, $\neg$.

For example, the compositional security analysis instance $\langle \gamma, (r_1 \in [s, o_1]) \wedge (r_2 \in [s, o_2]), \psi, \exists \rangle$ asks whether the system (γ, ψ) can reach a state in which s has both the right r_1 over o_1 and the right r_2 over o_2. We allow the formula φ to have infinite size. For example, suppose that $\mathcal{S}$, the set of all subjects, is $\{s_1, s_2, s_3, s_4, \ldots\}$, then the formula

$\neg(r \in [s_2, o] \vee r \in [s_3, o] \vee r \in [s_4, o] \vee \cdots)$ is true when no subject other than s_1 has the right r over object o.

Whether we should use security analysis or compositional security analysis is related to what types of policies we want to represent, and what types of policies we want to use as bases to compare the expressive power of different access control models or schemes. With compositional security analysis, we would be comparing models or schemes based on types of policies that are broader than with security analysis. For instance, if our set of queries Q contains queries related to users' access to files, then with compositional security analysis we can consider policies such as "Bob should never have write access to a particular file so long as his wife, Alice has a user account (and thus has some type of access to some file)."

5.1.1 Two types of reductions

In this section, we introduce the notions of reductions and state-matching reductions that we believe are adequate for comparing the expressive power of access control models. Before we introduce reductions, we discuss mappings between access control schemes.

Definition 5.1.2 *(*Mapping*)* Given two access control schemes $A = \langle \Gamma^A, Q^A, \vdash^A, \Psi^A \rangle$ and $B = \langle \Gamma^B, Q^B, \vdash^B, \Psi^B \rangle$. A *mapping* from A to B is a function σ that maps each pair $\langle \gamma^A, \psi^A \rangle$ in A to a pair $\langle \gamma^B, \psi^B \rangle$ in B and maps each query q^A in A to a query q^B in B. Formally, $\sigma : (\Gamma^A \times \Psi^A) \cup Q^A \rightarrow (\Gamma^B \times \Psi^B) \cup Q^B$.

Definition 5.1.3 *(*Security-Preserving Mapping*)* A mapping σ is said to be *security-preserving* when every security analysis instance in A is true if and only if the *image* of the instance is true. Given a mapping $\sigma : (\Gamma^A \times \Psi^A) \cup Q^A \rightarrow (\Gamma^B \times \Psi^B) \cup Q^B$, the *image* of a security analysis instance $\langle \gamma^A, q^A, \psi^A, \Pi \rangle$ *under* σ is $\langle \gamma^B, q^B, \psi^B, \Pi \rangle$, where $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ and $q^B = \sigma(q^A)$.

The notion of security-preserving mappings captures the intuition that simulations should preserve security properties. Given a security-preserving mapping from A to B

and an algorithm for solving the security analysis problem in B, one can construct an algorithm for solving the security analysis problem in A using the mapping. Also, security analysis in B is at least as hard as security analysis in A, modulo the efficiency of the mapping. If an efficient (polynomial-time) mapping from A to B exists, and security analysis in A is intractable (or undecidable), then security analysis in B is also intractable (undecidable). Security preserving mappings are not powerful enough for comparisons of access control schemes based on compositional security analysis. We need the notion of a strongly security-preserving mapping for that purpose.

Definition 5.1.4 *(*Strongly Security-Preserving Mapping*)* Given a mapping σ from scheme A to scheme B, the image of a compositional analysis instance, $\langle \gamma^A, \varphi^A, \psi^A, \Pi \rangle$, in A is $\langle \gamma^B, \varphi^B, \psi^B, \Pi \rangle$, where $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ and φ^B is obtained by replacing every query q^A in φ^A with $\sigma(q^A)$; we write also that $\varphi^B = \sigma(\varphi^A)$. A mapping σ from A to B is said to be *strongly security-preserving* when every compositional security analysis instance in A is true if and only if the image of the instance is true.

While the notions of security-preserving and strongly security-preserving mappings capture the intuition that simulations should preserve security properties, they are not convenient for us to use directly. Using the definition for either type of mapping to directly prove that the mapping is (strongly) security preserving involves performing security analysis, which is expensive. We now introduce the notions of reductions, which state structural requirements on mappings for them to be security preserving. We start with a form of reduction appropriate for compositional security analysis and then discuss weaker forms.

Definition 5.1.5 *(*State-Matching Reduction*)* Given a mapping from A to B, $\sigma : (\Gamma^A \times \Psi^A) \cup Q^A \rightarrow (\Gamma^B \times \Psi^B) \cup Q^B$, we say that the two states γ^A and γ^B are *equivalent* under the mapping σ when for every $q^A \in Q^A$, $\gamma^A \vdash^A q^A$ if and only if $\gamma^B \vdash^B \sigma(q^A)$. A mapping σ from A to B is said to be a *state-matching reduction* if for every $\gamma^A \in \Gamma^A$ and every $\psi^A \in \Psi^A$, $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ has the following two properties:

1. For every state γ_1^A in scheme A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$, there exists a state γ_1^B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$ and γ_1^A and γ_1^B are equivalent under σ.

2. For every state γ_1^B in scheme B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$, there exists a state γ_1^A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$ and γ_1^A and γ_1^B are equivalent under σ.

Property 1 says that for every state γ_1^A that is reachable from γ^A, there exists a reachable state in scheme B that is equivalent, i.e., answers all queries in the same way. Property 2 says the reverse, for every reachable state in B, there exists an equivalent state in A. The goal of these two properties is to guarantee that compositional security analysis results are preserved across the mapping. With the following theorem, we justify Definition 5.1.5.

Theorem 5.1.1 *Given two schemes A and B, a mapping σ from A to B is strongly security-preserving if and only if σ is a state-matching reduction.*

Proof **The "if" direction.** When σ is a state-matching reduction, given a compositional security analysis instance $\langle \gamma^A, \varphi^A, \psi^A, \Pi \rangle$ in scheme A, let $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ and $\varphi^B = \sigma(\varphi^A)$, we show that $\langle \gamma^A, \varphi^A, \psi^A, \Pi \rangle$ is true if and only if $\langle \gamma^B, \varphi^B, \psi^B, \Pi \rangle$ is true.

First consider the case that the instance $\langle \gamma^A, q^A, \psi^A, \Pi \rangle$ is existential, i.e., Π is $\exists$. If the instance is true, i.e., there exists a reachable state γ_1^A in which φ^A is true. Property 1 in Definition 5.1.5 guarantees that there exists a reachable state γ_1^B that is equivalent to γ_1^A; thus φ^B is true in γ_1^B; therefore, the instance in B, $\langle \gamma^B, \varphi^B, \psi^B, \exists \rangle$, is also true. However, if $\langle \gamma^B, \varphi^B, \psi^B, \exists \rangle$ is true, then there exists a reachable state γ_1^B in which φ^B is true. Property 2 in Definition 5.1.5 guarantees that there exists a state in A in which the analysis instance in A is true.

Now consider the case that the instance $\langle \gamma^A, \varphi^A, \psi^A, \Pi \rangle$ is universal, i.e., Π is $\forall$. If the instance is false, i.e., there exists a reachable state γ_1^A in which φ^A is false. Property 1 guarantees that the instance in B is also false. Similarly, if the instance in B is false, then the instance in A is also false.

The "only if" direction. When σ is not a state-matching reduction, then there exists $\gamma^A \in \Gamma^A$ and $\psi^A \in \Psi^A$ such that $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ violates one of the two properties in Definition 5.1.5.

First consider the case that Property 1 is violated. There exists a reachable state γ_1^A such that no state reachable from γ^B is equivalent to γ_1^A. Construct a formula φ^A as follows: φ^A is a conjunction of queries in Q or their complement. For every query q^A in Q^A, φ^A includes q^A if $\gamma_1^A \vdash^A q^A$ and $\neg q^A$ if $\gamma_1^A \vdash^A q^A$. (Note that the length of φ^A may be infinite, as the total number of queries may be infinite.) Clearly, φ^A is true in γ_1^A, but $\sigma(\varphi^A)$ is false in all states reachable from γ^B. Thus, the existential compositional analysis instance involving φ^A has different answers, and σ is not strongly security preserving.

Then consider the case that Property 2 is violated. There exists a state γ_1^B reachable from γ^B such that no state reachable from γ^A is equivalent to γ_1^B. Construct a formula φ^A as follows: φ^A is a conjunction of queries in Q or their complement. For every query query q^A in Q^A, φ^A includes q^A if $\gamma_1^B \vdash^B \sigma(q^A)$ and $\neg q^A$ if $\gamma_1^B \vdash^B \sigma(q^A)$. Clearly, φ^A is false in in all states reachable from γ^A, but $\sigma(\varphi^A)$ is true in γ_1^B; thus, the existential compositional analysis instance involving φ^A has different answers, and σ is not strongly security preserving. ■

Note that the proof uses a compositional analysis instance that contains a potentially infinite-length formula. If one chooses to restrict the formulas in analysis instances to be finite length, then state-matching reduction may not be necessary for being strongly security-preserving. Also, a state-matching reduction preserves compositional security properties. If we only need queries from Q to represent our policies and not compositions of those queries, then the following weaker notion of reductions is more suitable. However, we believe that the notion of state-matching reductions is quite natural by itself; it is certainly necessary when compositional queries are of interest.

Definition 5.1.6 *(*Reduction*)* Given two access control schemes $A = \langle \Gamma^A, Q^A, \vdash^A, \Psi^A \rangle$ and $B = \langle \Gamma^B, Q^B, \vdash^B, \Psi^B \rangle$. A mapping from A to B, σ, is said to be a *reduction* from A to B if for every $\gamma^A \in \Gamma^A$ and every $\psi^A \in \Psi^A$, $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ has the following two properties:

1. For every state γ_1^A and every query q^A in scheme A, if $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$, then in scheme B there exists a state γ_1^B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$ and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_1^B \vdash^B \sigma(q^A)$.

2. For every state γ_1^B in scheme B and every query q^A in scheme A, if $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$, there exists a state γ_1^A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$ and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_1^B \vdash^B \sigma(q^A)$.

Definition 5.1.5 differs from Definition 5.1.6 in that the former requires that for every reachable state in A (B, resp.) there exist a matching state in B (A, resp.) that gives the same answer for *every query*. Definition 5.1.6 requires the existence of a matching state for every query; however, the matching states may be different for different queries. Property 1 in Definition 5.1.6 says that for every reachable state in A and every query in A, there exists a reachable state in B that gives the same answer to (the image of) the query. Property 2 says the reverse direction. The goal of these two properties is to guarantee that security analysis results are preserved across the mapping. The fact that a reduction, as defined in Definition 5.1.6, is adequate for preserving security analysis results is formally captured by the following theorem.

Theorem 5.1.2 *Given two schemes A and B, a mapping, σ, from A to B is security preserving if and only if σ is a reduction.*

Proof **The "if" direction.** When σ is a reduction, given a security analysis instance $\langle \gamma^A, q^A, \psi^A, \Pi \rangle$ in scheme A, let $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ and $q^B = \sigma(q^A)$, we show that $\langle \gamma^A, q^A, \psi^A, \Pi \rangle$ is true if and only if $\langle \gamma^B, q^B, \psi^B, \Pi \rangle$ is true.

First consider the case that the instance $\langle \gamma^A, q^A, \psi^A, \Pi \rangle$ is existential, i.e., Π is $\exists$. If the instance is true, i.e., there exists a reachable state γ_1^A in which q^A is true. Property 1 in Definition 5.1.6 guarantees that there exists a reachable state γ_1^B in which q^B is true. Therefore, the instance in B, $\langle \gamma^B, q^B, \psi^B, \exists \rangle$, is also true. On the other hand, if $\langle \gamma^B, q^B, \psi^B, \exists \rangle$ is true, then there exists a reachable state γ_1^B in which q^B is true. Property 2 in Definition 5.1.6 guarantees that there exists a state in A in which q^A is true; thus the analysis instance in A is true.

Now consider the case that the instance $\langle \gamma^A, q^A, \psi^A, \Pi \rangle$ is universal, i.e., Π is $\forall$. If the instance is false, i.e., there exists a reachable state γ_1^A in which q^A is false. Property 1 guarantees that the instance in B is also false. Similarly, if the instance in B is false, then the instance in A is also false.

The "only if" direction. When σ is not a reduction, then there exists $\gamma^A \in \Gamma^A$ and $\psi^A \in \Psi^A$ such that $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ violates one of the two properties in Definition 5.1.6.

First consider the case that Property 1 is violated. There exists a reachable state γ_1^A and a query q^A such that for every state reachable from γ^B the answer for the query $\sigma(q^A)$ under the state is different from the answer for q^A under γ_1^A. If $\gamma_1^A \vdash^A q^A$, then this means that q^B is false in every state reachable from γ^B. Thus the security analysis instance $\langle \gamma^A, q^A, \psi^A, \exists \rangle$ is true, but its image under σ is false. Thus, the mapping σ is not security-preserving. If $\gamma_1^A \nvdash^A q^A$, then this means that q^B is true in every state reachable from γ^B. Thus the security analysis instance $\langle \gamma^A, q^A, \psi^A, \forall \rangle$ is false, but its image under σ is true.

Then consider the case that Property 2 is violated. There exists a state γ_1^B reachable from γ^B and a query q^A such that for every state reachable from γ^A the answer for the query q^A under the state is different from the answer for $\sigma(q^A)$ under γ_1^B. If $\gamma_1^B \vdash^B \sigma(q^A)$, then this means that q^A is false in every state reachable from γ^A. Thus the security analysis instance $\langle \gamma^A, q^A, \psi^A, \exists \rangle$ is false, but its image under σ is true. If $\gamma_1^B \nvdash^B q^B$, then this means that q^A is true in every state reachable from γ^A. Thus the security analysis instance $\langle \gamma^A, q^A, \psi^A, \forall \rangle$ is true, but its mapping in B is false. ■

Comparisons of two access control models are based on comparisons among access control schemes in those models.

Definition 5.1.7 *(*Comparing the Expressive Power of Access Control Models*)* Given two access control models $\mathcal{M}$ and $\mathcal{M}'$, we say that $\mathcal{M}'$ is at least as expressive as $\mathcal{M}$ (or $\mathcal{M}'$ has at least as much expressive power as $\mathcal{M}'$) if for every scheme in $\mathcal{M}$ there exists a state-matching reduction (or a reduction) from it to a scheme in $\mathcal{M}'$. In addition, if for every scheme in $\mathcal{M}'$, there exists a state-matching reduction (reduction) from it to a scheme in

$\mathcal{M}$, then we say that M and M' are equivalent in expressive power. If $\mathcal{M}'$ is at least as expressive as the $\mathcal{M}$, and there exists a scheme A in $\mathcal{M}'$ such that for any scheme B in $\mathcal{M}$, no state-matching reduction (reduction) from A to B exists, we say that $\mathcal{M}'$ is strictly more expressive than $\mathcal{M}$.

We compare the expressive power of two schemes based on state-matching reductions when compositional queries are needed to represent the policies of interest. Otherwise, reductions suffice. Observe that we can use the above definition to compare the expressive power of two access control schemes A and B, by viewing each scheme as an access control model consists of just that scheme.

5.1.2 Discussions of alterative definitions for reduction

In this section, we discuss alternative definitions that differ slightly from the ones discussed in the previous section. The first of these definitions is used by [10, 18] for simulations.

Definition 5.1.8 *(*Form-1 Weak Reduction*)* A mapping from A to B, given by $\sigma : (\Gamma^A \times \Psi^A) \cup Q^A \rightarrow (\Gamma^B \times \Psi^B) \cup Q^B$, is a ***form-1 weak reduction*** if for every $\gamma^A \in \Gamma^A$ and every $\psi^A \in \Psi^A$, $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ has the following two properties:

1. For every query q^A, if there exists a state γ_1^A in scheme A such that $\gamma^A \overset{*}{\mapsto}_{\psi^A} \gamma_1^A$ and $\gamma_1^A \vdash^A q^A$, then there exists a state γ_1^B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$ and $\gamma_1^B \vdash^B \sigma(q^A)$.

2. For every query q^A, if there exists γ_1^B in scheme B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$ and $\gamma_1^B \vdash^B \sigma(q^A)$, then there exists a state γ_1^A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$ and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_1^B \vdash^B \sigma(q^A)$.

The intuition underlying Definition 5.1.8, as stated by [10] is, “systems are equivalent if they have equivalent worst case behavior”. Therefore, simulations only need to preserve the worst-case access. Definition 5.1.8 is weaker than Definition 5.1.6 in that it requires the existence of a matching state when a query is true in the state, but does not require so

when the query is false. Therefore, it is possible that a query q^A is true in all states that are reachable from γ^A, but the query $\sigma(q^A)$ is false in some states that are reachable from γ^B (the query $\sigma(q^A)$ needs to be true in at least one state reachable from γ^B). This indicates that Definition 5.1.8 does not preserve answers to universal security analysis instances. Definition 5.1.8 is adequate for the purposes in [10, 18] as only simple safety analysis (which is existential) was considered there.

The decision of defining a mapping to be a function from $(\Gamma^A \times \Psi^A) \cup Q^A$ to $(\Gamma^B \times \Psi^B) \cup Q^B$ also warrants some discussion. One alternative is to define a mapping from A to B to be a function that maps each state in A to a state in B, each state-transition rule in A to a state-transition rule in B, and each query in A to a query in B. Such a function would be denoted as $\sigma : \Gamma^A \cup \Psi^A \cup Q^A \rightarrow \Gamma^B \cup \Psi^B \cup Q^B$. One can verify any such function is also a mapping according to Definition 5.1.2, which gives more flexibility in terms of mapping states and state-transition rules from A to B. By Definition 5.1.2, the state corresponding to a state γ^A may also depends upon the state-transition being considered.

Another alternative is to define a mapping from A to B to be a function $\sigma : \Gamma^A \times \Psi^A \times Q^A \rightarrow \Gamma^B \times \Psi^B \times Q^B$, in other words, the mapping of states, state-transition rules, and queries may depend on each other. This definition will also leads to a weaker notion of reduction:

Definition 5.1.9 *(*Form-2 Weak Reduction*)* A form-2 weak reduction from A to B is a function $\sigma : \Gamma^A \times \Psi^A \times Q^A \rightarrow \Gamma^B \times \Psi^B \times Q^B$ such that for every $\gamma^A \in \Gamma^A$, every $\psi^A \in \Psi^A$, and every $q^A \in Q^A$, $\langle \gamma^B, \psi^B, q^B \rangle = \sigma(\langle \gamma^A, \psi^A, q^A \rangle)$ has the following two properties:

1. For every state γ_1^A in scheme A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$, there exists a state γ_1^B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$ and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_1^B \vdash^B q^B$.

2. For every state γ_1^B in scheme B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$, there exists a state γ_1^A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$ and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_1^B \vdash^B q^B$.

It is not difficult to prove that a Form-2 weak reduction is also security preserving, in the sense that any security analysis instance $\langle \gamma^A, q^A, \psi^A, \Pi \rangle$ in A can be mapped to a

security analysis in B. However, it is not a mapping, as the mapping of states and state-transition rules may depend on the query.

Definition 5.1.9 is used implicitly in Theorems 2 and 3 in [7] (that are reproduced in the previous chapter) for reductions from security analysis in two RBAC schemes to that in the RT Role-based Trust-management framework [5, 69]. As we state in Theorem 5.3.5 in this chapter, a form-2 weak reduction used in [7] for one of the RBAC schemes can be changed to a security-preserving mapping in a straightforward manner.

We choose not to adopt this weaker notion of reduction for the following reason. Under this definition, given an access control system (γ^A, ψ^A), to answer n analysis instances involving different queries, one has to do n translations of states and state-transitions, which are often time consuming. While using Definition 5.1.2 and Definition 5.1.6, one can do the mapping of (γ^A, ψ^A) once and use it to answer all n analysis instances.

A third weak form of reduction is introduced by [17]. That work discusses the expressive power of multi-parent creation when compared to single-parent creation.

Definition 5.1.10 *(*Form-3 Weak Reduction*)* A mapping from A to B, given by $\sigma : (\Gamma^A \times \Psi^A) \cup Q^A \rightarrow (\Gamma^B \times \Psi^B) \cup Q^B$, is a *form-3 weak reduction* if for every $\gamma^A \in \Gamma^A$ and every $\psi^A \in \Psi^A$, $\langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ has the following two properties:

1. For every state γ_1^A and every query q^A in scheme A, if $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$, then in scheme B there exists a state γ_1^B such that $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$ and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_1^B \vdash^B \sigma(q^A)$.

2. For every state γ_1^B in scheme B and every query q^A in scheme A, if $\gamma^B \overset{*}{\mapsto}_{\psi^B} \gamma_1^B$, then either (a) there exists a state γ_1^A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$ and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_1^B \vdash^B \sigma(q^A)$, or (b) there exists a state γ_2^B such that $\gamma_1^B \overset{*}{\mapsto}_{\psi^B} \gamma_2^B$ and a state γ_1^A such that $\gamma^A \overset{*}{\mapsto}_{\psi} \gamma_1^A$, and $\gamma_1^A \vdash^A q^A$ if and only if $\gamma_2^B \vdash^B \sigma(q^A)$.

As pointed out by [17], this form of reduction suffices for preserving simple safety properties in monotonic schemes — those schemes in which once a state is reached in which a query is true, in all reachable states from that state, the query remains true. Therefore, this form of reduction cannot be used to compare schemes when queries can become

false after being true. As with the reduction from Definition 5.1.8, this form of reduction cannot be used for universal queries.

5.2 The implementation paradigm for simulation: an examination

Several authors use the implementation paradigm for simulations, e.g., [11] state that "a positive answer [to the question whether LBAC (lattice-based access control) can be simulated in RBAC] is also practically significant, because it implies that the same Trust Computing Base can be configured to enforce RBAC in general and LBAC in particular." However, in these papers [11, 55, 58], a precise definition for simulations is not given. This makes the significance of such results unclear, at least in terms of comparing the expressive power of different access control models.

In this section, we analyze the implementation paradigm and argue that this does not lead to a notion of simulations that is meaningful for comparing the expressive power of different access control models. More precisely, the notions of simulations derived from this paradigm are so weak that almost all access control schemes are equivalent.

To formalize the implementation paradigm for simulation, a natural goal is to use an implementation of an access control scheme for another scheme. Intuitively, if a scheme A can be simulated in a scheme B, then there exists a *simulator* that, when given access to the interface to (an implementation of) B, can provide an interface that is exactly the same as the interface to (an implementation of) A.

When considering the interface of an access control scheme, we have to consider how state-transitions occur. Intuitively, an access control system changes its state because some actors (subjects, principals, users, etc.) initiate certain actions. An implementation of an access control scheme thus has an interface consisting of at least the following functions:

- $init(\gamma)$: set the current state to γ.
- $query(q)$: ask the query q and receives a yes/no response.

- $apply(a)$: apply the action a on the system, which may result in a state-transition in the system.

- functions providing other capabilities, e.g., traversing the subjects and objects in the system.

A simulator of A in B is thus a program that takes an interface of B and provides an interface of A that is indistinguishable from an implementation for A. In other words, the simulator is a blackbox that when given access to a backbox implementation of B, gives an implementation of A. This intuition seems to make sense if the goal is to use an implementation of B to implement A.

It is tempting to start formalizing the above intuition; however, there are several subtle issues that need to be resolved first.

As can be easily seen, for any two schemes A and B, a trivial simulator exists. The simulator implements all the functionalities of A by itself, without interacting with the implementation of B. Clearly, one would like to rule out these trivial simulators. One natural way to do so is to restrict the amount of space used by the simulator to be sub-linear in the size of the state of the scheme it is simulating. It *seems* to be a reasonable requirement that the simulator takes constant space on its own, i.e., the space used by the simulator does not depend on the size of the state. (The space used by the implementation of B is not considered here.)

Another issue is whether to further restrict a simulator's internal behavior. When the simulator receives a query in the scheme A, it may issue multiple queries to the black-box implementation of B before answering the query; it may even perform some state-transition on B before answering the query. Similarly, the simulator may perform multiple queries and state-transitions on B to simulate one state-transition in A.

If no restriction is placed, then the notion of simulation is too weak to separate different access control models. For example, [58] constructed a simulation of ATAM in RBAC. In Section 5.4, we give a simulation of RBAC in strict DAC, a discretionary model that allows only the owner of an object to grant rights over the object to another subject

and ownership cannot be transferred. According to these results, the simplest DAC (in which security analysis is efficiently decidable) has the same expressive power as ATAM (in which simple safety analysis is undecidable). This illustrates the point that, without precise requirements, simulation is not a useful concept for comparing access control models.

If one places restrictions on the simulator, then the question is what restrictions are reasonable. Our conclusion is that it is difficult to justify such requirements. In the following, we elaborate on this.

One possibility that we now argue to be inadequate is to restrict the internal behavior of the simulator, e.g., to restrict it to issue only one query to B to answer one query in A and to make a bounded number of state-transitions in B to simulate one state-transition in A. Under these restrictions, one can prove that RBAC cannot be simulated in the HRU model. The assignment of a user to a role in RBAC results in the user gaining all the accesses to objects implied by the permissions associated with that role; therefore, it changes the answers to an unbounded number of queries (queries involving those permissions.) One may argue that the assignment of a user to a role is a single "action" in RBAC, and therefore, the acquiring of those permissions by that user is accomplished in a single "action." The corresponding assignment of rights in the HRU access matrix cannot be accomplished by a single command, or a bounded number of commands for that matter, as each command only changes a bounded number of cells in the matrix. Thus, any mapping of the user-assignment in RBAC involves an unbounded number of commands being executed in HRU. Nonetheless, one can argue that this is balanced by the efficiency of checking whether a user has a particular right in the two models. A naive implementation of an RBAC model may involve having to collect all roles to which that user is assigned, and then collecting all permissions associated with those roles, and then checking whether one of those permissions corresponds to the object and access right for which we are checking. The time this process takes depends on the size of the current state and is unbounded. The corresponding check in HRU is simpler: we simply check whether the corresponding access right exists in the cell in the matrix. Thus, we can argue that there is a trade-off be-

tween time-to-update, and time-to-check-access between the two schemes. Therefore, we argue that it does not make sense to restrict the number of steps involved in the simulation.

Another possibility that we now argue to be inadequate is to measure how much time the simulator takes to perform a state-transition and to answer one query in the worst case and require that there cannot be a significant slowdown. This possibility is complicated by the fact that the efficiency of these operations are not predetermined in any access control scheme, the implementation can make trade-offs between time complexity and space complexity and between query answering and state-transitions. Any comparison must involve at least three axes: query time, state-transition time, and space. Furthermore, the best ways to implement an access control scheme are not always known. Finally, these implementation-level details do not seem to belong in the comparison of access control models; as such models by themselves are abstract models to study properties other than efficiency.

In summary, when no restriction is placed on the simulations, the "implementation paradigm" does not separate different access control schemes. However, it seems difficult to justify the restrictions that have been considered in the literature. Therefore, our analysis in this section suggests that the "implementation paradigm" does not seem to yield effective definitions of simulations that are useful to compare access control models. This also suggests that expressive power results proved under this paradigm should be reexamined.

5.3 Applying the theory

In this section, we apply our theory from Section 5.1 to compare the expressive power of different access control schemes. In the following section, we show that the HRU access matrix scheme is not as expressive as a relatively simple trust management scheme, RT[ ]. We then examine two particular results from literature using our theory: (1) that RBAC is at least as expressive as DAC (Sections 5.3.2 and 5.3.3), and (2) that TAM is at least as expressive as ATAM (Section 5.3.5), and in each case, assert the opposite. We show

also that the trust management scheme $\mathsf{RT}[\cap]$ is at least as expressive as an RBAC scheme (Section 5.3.4).

Proof Methodology In this section, we prove the existence of reductions and state-matching reductions as well as the nonexistence of state-matching reductions. To prove that there exists a reduction or state-matching reduction from a scheme A to a scheme B, we constructively give a mapping and show that the mapping satisfies the requirements. To prove that there does not exist a state-matching reduction from a scheme A to a scheme B is more difficult, as we have to show that no mapping satisfies the requirements for a state-matching reduction. Our strategy is to use proof by contradiction. We find in scheme A a state γ^A, a state-transition rule ψ^A, as well as a state γ_1^A that is reachable. Suppose, for the sake of contradiction, that a state-matching reduction exists, then there exist states γ^B and γ_1^B such that γ^B is equivalent to γ^A, γ_1^B is equivalent to γ_1^A, and γ_1^B is reachable from γ^B. We show that among the sequence of states leading from γ^B and γ_1^B, there exists one for which there is no matching state that is reachable in A.

5.3.1 Comparing the HRU scheme to a trust management scheme

The HRU scheme [2] is based on the access matrix model, and has generally been believed to have considerable expressive power, partly because it has been shown that one can simulate a Turing Machine in the HRU scheme. In this section, we show that there does not exist a state-matching reduction from a relatively simple trust management scheme, $\mathsf{RT}[\,]$ [5, 73], to the HRU scheme. That $\mathsf{RT}[\,]$ cannot be encoded in the HRU scheme is informally discussed and conjectured in [5, 73]. Using the theory presented in Section 5.1, we are able to formally prove this. As safety analysis is efficiently decidable in $\mathsf{RT}[\,]$ but undecidable in the HRU scheme, there does not exist a state-matching reduction from the HRU scheme to the $\mathsf{RT}[\,]$ scheme either. This shows that the expressive powers of the HRU scheme and of $\mathsf{RT}[\,]$ are incomparable. This Is the first formal evidence of the limited expressive power of the HRU scheme.

The fact that the HRU scheme can simulate a Turing Machine shows that it can compute any computable function when used as a computation device. When used as an access control scheme, the HRU scheme may nonetheless be limited in expressive power. For example, it cannot encode an access control system where in one state a subject has no right over any object and in the next state the subject obtains rights over a potentially unbounded number of objects.

The HRU Scheme

Γ We assume the existence of countably infinite sets of subjects, $\mathcal{S}$, objects $\mathcal{O}$ and rights $\mathcal{R}$, with $\mathcal{S} \subset \mathcal{O}$. Each state γ is characterized by $\langle S_\gamma, O_\gamma, R_\gamma, M_\gamma[\,]\rangle$ where $S_\gamma \subset \mathcal{S}$ is a finite set of subjects that exist in the state γ, $O_\gamma \subset \mathcal{O}$ is a finite set of objects that exist in the state γ, $R_\gamma \subset \mathcal{R}$ is a finite set of rights that exist in the state γ, and $M_\gamma[\,]$ is the access matrix, i.e., $M_\gamma[s, o] \subseteq R_\gamma$ gives the set of rights $s \in S_\gamma$ has over $o \in O_\gamma$ in the state γ. $M_\gamma[s, o]$ is defined only when $s \in S_\gamma$ and $o \in O_s t$. It may appear that we allow R_γ to differ across states. The definition for state-change rules precludes this possibility.

Ψ A state-change rule, ψ, in the HRU scheme is a command schema, i.e., a set of commands. Each command takes a sequence of parameters, each of which may be instantiated by an object, Each command has also an optional condition, which is a conjunction of clauses. Each clause checks whether a right is in a particular cell of $M_\gamma[\,]$. Following the (optional) conditions in a command is a sequence of primitive operations. The primitive operations are one of the following: (1) create an object; (2) create a subject; (3) enter a right into a cell of the access matrix; (4) remove a right from a cell of the access matrix; (5) destroy a subject; (6) destroy an object. We refer the reader to [2] for more details on the syntax of commands. A state-change is the successful execution of a command.

Q We allow queries of the following two forms: (1) $r \in M[s, o]$, and (2) $r \notin M[s, o]$. In the queries, $r \in \mathcal{R}$, $s \in \mathcal{S}$ and $o \in \mathcal{O}$. These are the only kinds of queries that have been considered in the context of the HRU scheme in the literature. In particular, these are the queries that are pertinent to the safety property [2].

$\vdash$ Let q be the query $r \in M[s, o]$. Then, given a state γ, $\gamma \vdash q$ if and only if $r \in R_\gamma$, $s \in S_\gamma$, $O \in O_\gamma$ and $r \in M_\gamma[s, o]$. Otherwise, $\gamma \nvdash q$, or equivalently $\gamma \vdash \neg q$. Let $\widehat{q}$ be the query $r \notin M[s, o]$. Then $\gamma \vdash \widehat{q}$ if and only if $r \in R_\gamma$, $s \in S_\gamma$, $O \in O_\gamma$ and $r \notin M_\gamma[s, o]$. Otherwise, $\gamma \nvdash \widehat{q}$, or equivalently $\gamma \vdash \neg\widehat{q}$.

Observe that one should view both $r \in M_\gamma[s, o]$ and $r \notin M_\gamma[s, o]$ as atomic queries. In particular $\neg(r \in M_\gamma[s, o])$ is not equivalent to $r \notin M_\gamma[s, o]$. It is possible that $\gamma \nvdash r \in M_\gamma[s, o]$ and $\gamma \nvdash r \notin M_\gamma[s, o]$; this happens when either s or o does not exist in γ. Even though it is not possible that $\gamma \vdash ((r \in M_\gamma[s, o]) \wedge (r \notin M_\gamma[s, o]))$.

The RT[ ] Scheme

Γ We assume the existence of countably infinite sets of principals (e.g., A, B, C) and role names (e.g., r, s, t, u). A role is formed by a principal and a role name, separated by a dot (e.g., $A.r, X.u$). An RT[ ] state consists of statements which are assertions made by principals about membership in their roles. Two types of assertions are supported. These are simple member (e.g., $A.r \longleftarrow B$) and simple inclusion (e.g., $A.r \longleftarrow B.r_1$). One reads the $\longleftarrow$ symbol as "includes". The example for the first kind of statement asserts that B is a member of A's r role. The example for the second kind of statement asserts that every member of $B.r_1$ is a member of $A.r$. The portion of a statement that appears to the left of the $\longleftarrow$ symbol is called its head, and the portion that appears to the right is called the body. We refer the reader to [69] for more details on the syntax and semantics of RT[ ] statements.

Ψ A state-change rule in a system based on the RT[ ] scheme consists of two sets, G and S. Both consist of RT[ ] roles. G is the set of growth-restricted roles, i.e., if $A.r \in G$, then statements with $A.r$ at the head cannot be added in future states. S is the set of shrink-restricted roles, i.e., if $A.r \in S$, then roles with $A.r$ at the head cannot be removed in future states. We refer the reader to [5, 73] for more details on the two sets, and the intuition behind them.

Q [5] define three kinds of queries in RT[ ]. (1) $\{B_1, \ldots, B_n\} \sqsupseteq A.r$ – this kind of query asks whether the role $A.r$ is bounded by the set of pricipals $\{B_1, \ldots, B_n\}$; (2)

$A.r \sqsupseteq \{B_1, \ldots, B_n\}$ – this kind of query asks whether each principal $B_1, \ldots, B_n$ is a member of $A.r$; (3) $X.u \sqsupseteq A.r$ – this kind of query asks whether the set of member of $A.r$ is included in the set of members of $X.u$.

$\vdash$ Given a state, we check if a query is entailed by first evaluating the set of members of each RT[ ] role in the query. This is done using credential chain discovery [70]. We then compare the two sets and check if the set to the left includes the set to the right. The first two kinds of queries are called semi-static queries as one of the sides in the query is a set of users that is independent of the state, and needs no further evaluation. We refer the reader to [70] for more details on query-entailment in RT[ ].

Theorem 5.3.1 *There exists no state-matching reduction from the* RT[ ] *scheme to the HRU scheme.*

Proof By contradiction. Assume that there exists a state-matching reduction, σ, from the RT[ ] scheme to the HRU scheme. We denote components of a RT[ ] system with the superscript R and the HRU scheme with the superscript H. We now consider a system based on the RT[ ] scheme. Let γ^R be the start-state in our RT[ ] system such that γ^R has no statements. The state-change rule in our RT[ ] system is $G = S = \emptyset$. We now consider the start-state in the corresponding HRU system $\sigma(\gamma^R) = \gamma^H$ and the state-change rule $\sigma(\psi^R) = \psi^H$. Let k be the number of objects in γ^H, i.e., $k = |O_{\gamma^H}|$. Let l be the maximum number of primitive operations of the form "enter right" in any of the commands in ψ^H. Let m be the maximum number of primitive operations of the form "remove right" in any of the commands in ψ^H.

Choose some $n > (k^2 + l + m) + 1$. Our choice of n is such that for any γ_1^H such that $\gamma^H \mapsto \gamma_1^H$, fewer than $n - 1$ queries that are true in γ^H (i.e., are entailed by γ^H) are false in γ_1^H (i.e., are not entailed by γ_1^H). The reason is that: (1) as γ^H has at most k objects (some or all of which may be subjects), a command may contain statements to destroy all these objects. Consequently, these statements can cause up to k^2 queries of the form $r \notin M[s, o]$ to be false in γ_1^H when they are true in γ^H; (2) as a command in ψ^H has at most l statements to enter rights in to cells, these statements can cause up to l queries of

the form $r \notin M[s, o]$ to be false in γ_1^H when they are true in γ^H; (3) as a command in ψ^H has at most m statements to remove rights from cells, these statements can cause up to m queries of the form $r \in M[s, o]$ to be false in γ_1^H when they are true in γ^H. We emphasize that these are the only possibilities for queries to become false in a state-change from γ^H; the number of queries that are entailed by γ^H, but not γ_1^H is fewer than $n - 1$.

Consider queries q_i^R for each integer i such that $1 \leq i \leq n$ in the RT[ ] system where q_i^R is of the form $\{B_i\} \sqsupseteq A.r$ for some principals $A, B_1, \ldots, B_n$ and some role $A.r$. We make two observations about these queries. The first is that $\gamma^R \vdash q_1^R \wedge \ldots \wedge q_n^R$. The reason is that $A.r$ is empty in γ^R and therefore is a subset of every set of the form $\{B_i\}$. The second observation is that in all states reachable from γ^R, either all queries of the form q_i^R such that $1 \leq i \leq n$ are entailed, or at most one of those queries is entailed. The reason is that for the set of users in the role $A.r$ to be a subset of $\{B_i\}$ for a particular i, it must be either empty, or contain exactly one element, B_i. Now consider the state γ_t^R such that $\gamma^R \overset{*}{\mapsto}_\psi \gamma_t^R$ and $\gamma_t^R \vdash q_1^R \wedge \neg q_2^R \wedge \ldots \wedge \neg q_n^R$. That is, q_1^R is true in γ_t^R, but none of the other queries of the form q_i^R is true. We use the subscript t only to demarcate the state and not as a count of the number of state-changes needed to reach it. In fact, γ_t^R can be reached from γ^R with a single state-change: we simply add the statement $A.r \longleftarrow B_1$ to our RT[ ] system.

Now consider the corresponding states and queries in the HRU system produced as output by σ. Let $\gamma^H = \sigma(\gamma^R)$, $\gamma_t^H = \sigma(\gamma_t^R)$, and $q_i^H = \sigma(\gamma_i^R)$ for $1 \leq i \leq n$. As we assume that σ is a state-matching reduction, $\gamma^H \vdash q_1^H \wedge \ldots \wedge q_n^H$, and there exists γ_t^H such that $\gamma^H \overset{*}{\mapsto}_\psi \gamma_t^H$ and $\gamma_t^H \vdash q_1^H \wedge \neg q_2^H \wedge \ldots \wedge \neg q_n^H$. Consider any sequence of state-changes from γ^H to γ_t^H. Pick the first state in the sequence γ_c^H in which at least one of the queries q_i^H is false. Consider the state γ_{c-1}^H immediately preceding it. Then, $\gamma_{c-1}^H \vdash q_1^H \wedge \ldots \wedge q_n^H$. Because one step of change cannot make $n-1$ queries to go from true to false, in γ_c^H, some queries $q_1, q_2, q_3, \cdots, q_n$ are false but at least 2 queries in them are true. As we argued in the previous paragraph, there cannot exist a matching state in A for γ_c^H. We now have the desired contradiction to the existence of a state-matching reduction from the RT[ ] scheme to the HRU scheme. ■

5.3.2 Examining comparisons of RBAC and DAC

Munawer and Sandhu [58] presents a simulation of ATAM in RBAC and conclude that RBAC is at least as expressive as ATAM. [11, 55, 75] give simulations of various MAC and DAC schemes in RBAC. The main conclusion of [11, 55, 75] is that as MAC and DAC can be simulated in RBAC, a Trusted Computing Based (TCB) needs to include an implementation of RBAC only, and DAC and MAC policies can be successfully represented and enforced by the TCB.

In the simulations used in [11, 55, 58, 75], the preservation of safety (or other security) properties is not identified as an objective. From the above conclusion in [11, 55, 75], it seems that they follow the implementation paradigm. As discussed in Section 5.2, this paradigm leads to a weak notion of simulations, as exemplified by the simulation of RBAC in strict DAC in Section 5.4.

We observe also that the problem of comparing RBAC with DAC as stated by [11, 55] is ill-defined (or at least not clearly defined). RBAC by itself only specifies the structures to store access control information, but not how to manipulate these structures, which are specified by administrative models. In other words, only the set Γ of states is precisely defined, the set Ψ of state-transition rules is not. The counterpart of RBAC is the access matrix model, instead of DAC (or MAC). In DAC, we specify that access control information is stored in a matrix, and we also specify rules on how to change the access matrix. The statement that RBAC is at least as expressive as DAC (or MAC) is similar to saying that the access matrix model is at least as expressive as DAC or MAC. Comparing the RBAC model with the access matrix model is not fruitful either, as both models can include arbitrary state-transition rules.

5.3.3 Comparing ARBAC97 with a form of DAC

To compare any RBAC-based model with DAC, one needs to specify the administrative model (state-transition rules) for RBAC. In existing comparisons of RBAC and DAC [11, 55, 58], new and rather complicated administrative models are introduced “on

the fly" to simulate the effects in DAC. In this section, we compare the expressive power of RBAC with ARBAC97 [37] as the administrative model to that of SDCO, a rather simple form of DAC. We first present precise characterizations of SDCO and the ARBAC97 scheme. We then assert that while there does exist a reduction, there does not exist a state-matching reduction from SDCO to the ARBAC97 scheme, given a natural query set for each scheme.

This result is significant as it shows that we cannot assert that RBAC is more expressive than DAC without qualifying the assertion; a strongly security-preserving mapping does not exist from SDCO to ARBAC97. Our conclusion provides the first evidence that the expressive power of RBAC (or at least some reasonable incarnation of it) is limited.

The SDCO Scheme

Γ SDCO is a scheme based on the access matrix model and is a special case of the HRU scheme. Each state $\gamma \in \Gamma$ is $\langle S_\gamma, O_\gamma, M_\gamma[\,], R_\gamma \rangle$ where S_γ, O_γ and R_γ are finite, strict subsets of the countably infinite sets $\mathcal{S}$ (subjects), $\mathcal{O}$ (objects) and $\mathcal{R}$ (rights) respectively. The set of rights for the scheme is $R_\gamma = \{own, r_1, \ldots, r_n\}$, where own is the distinguished right indicating ownership of the object. $M_\gamma[\,]$ is the access matrix.

Ψ The state-transition rules are the commands $createObject$, $destroyObject$ and $grantOwn$, and for each $r_i \in R_\gamma - \{own\}$, a command $grant\_r_i$.

$command\ \ createObject(s, o)$
  $create\ object\ o$
  $enter\ own\ into\ [s, o]$

$command\ \ destroyObject(s, o)$
  $if\ own \in [s, o]$
    $destroy\ o$

$command\ \ grantOwn(s, s', o)$
  $if\ own \in [s, o]$
    $enter\ own\ into\ [s', o]$
    $remove\ own\ from\ [s, o]$

$command\ \ grant\_r_i(s, s', o)$
  $if\ own \in [s, o]$
    $enter\ r_i\ into\ [s', o]$

Q Each query is of one the following forms: (1) Is $s \in S$?; (2) Is $o \in O$?; and (3) Is $r \in M[s, o]$?

$\vdash$ The entailment relation is defined as follows for each type of query from above. In each of the following, $\gamma \in \Gamma$ is a state. (1) $\gamma \vdash s \in S$ if and only if $s \in S_\gamma$; (2) $\gamma \vdash o \in O$ if and only if $o \in O_\gamma$; (3) $\gamma \vdash r \in M[s, o]$ if and only if $r \in R_\gamma \wedge s \in S_\gamma \wedge o \in O_\gamma \wedge r \in M_\gamma[s, o]$.

The ARBAC97 Scheme

Γ We assume the existence of the countably infinite sets $\mathcal{U}$ (users), $\mathcal{P}$ (permissions) and $\mathcal{R}$ (roles). An ARBAC97 state is $\langle UA, PA, RH, AR \rangle$ where UA is the user-role assignment relation that contains a pair $\langle u, r \rangle$ for every user $u \in \mathcal{U}$ that is assigned to a role $r \in \mathcal{R}$. PA is the permissions-role assignment relation that contains a pair $\langle p, r \rangle$ for every permission $p \in \mathcal{P}$ that is assigned to the role $r \in \mathcal{R}$. RH is the role-hierarchy, and for $r_1, r_2 \in \mathcal{R}$, $r_1 \succeq r_2 \in RH$ means that all users that are members of r_1 are also members of r_2, and all permissions that are assigned to r_2 are authorized to users that are members of r_1. $AR \subset \mathcal{R}$ is a set of administrative roles. In ARBAC97 [37], changes to AR may be made only by a central System Security Officer (SSO) who is trusted not to leave the system in an undesirable state; if the SSO effects a state-transition, then she does security analysis to ensure that the resulting state is acceptable. Therefore, in our analysis, we assume that AR does not change.

Ψ State-transitions in the ARBAC97 scheme are predicated on the relations that are part of the $URA97$ (user-roles assignment), $PRA97$ (permission-role assignment) and $RRA97$ (role-role assignment) components. We introduce the notion of a role range that is used in the definition of the state-transitions. A role range, ξ is written as (r_1, r_2), where r_1 and r_2 are roles, and every role r that satisfies $r_1 \succeq r \wedge r \succeq r_2 \wedge r \neq r_1 \wedge r \neq r_2$ is in the role range ξ. We write $r \in \xi$ when r is in the role range ξ. We represent as Ξ the set of

all role ranges. Role ranges in ARBAC97 satisfy some other properties, and we refer the reader to [37] for those. Those properties are not relevant to our discussion here.

$$URA97 \begin{cases} can\_assign \subseteq AR \times CR \times \Xi \\ can\_revoke \subseteq AR \times \Xi \end{cases} \qquad PRA97 \begin{cases} can\_assignp \subseteq AR \times CR \times \Xi \\ can\_revokep \subseteq AR \times \Xi \end{cases}$$

$$RRA97 \begin{cases} can\_modify \subseteq AR \times \Xi \end{cases}$$

CR is a set of pre-requisite conditions. A pre-requisite condition is a propositional logic formula over regular roles. For instance, $c = r_1 \wedge \overline{r_2}$ is a pre-requisite condition that indicates: "role r_1 and not role r_2," where $r_1, r_2 \in R$.

We postulate that a state-transition is the successful execution one of the following operations.

assignUser(a, u, r)
  if $\exists \langle ar, c, \xi \rangle \in$ *can_assign such that*
  a is a member of $ar \wedge u$ *satisfies* $c \wedge$
  $r \in \xi$ *then*
    add $\langle u, r \rangle$ *to UA*

revokeUser(a, u, r)
  if $\exists \langle ar, \xi \rangle \in$ *can_revoke such*
  that a is a member of $ar \wedge$
  $r \in \xi$ *then*
    remove $\langle u, r \rangle$ *from UA*

assignPermission(a, p, r)
  if $\exists \langle ar, c, \xi \rangle \in$ *can_assignp such that*
  a is a member of $ar \wedge p$ *satisfies* $c \wedge$
  $r \in \xi$ *then*
    add $\langle p, r \rangle$ *to PA*

revokePermission(a, p, r)
  if $\exists \langle ar, \xi \rangle \in$ *can_revokep such*
  that a is a member of $ar \wedge$
  $r \in \xi$ *then*
    remove $\langle p, r \rangle$ *from PA*

addToRange(a, ξ, r)
  if $\exists \langle ar, \xi \rangle \in$ *can_modify such that*
  a is a member of ar then
    add $r_1 \succeq r$ *to RH*
    add $r \succeq r_2$ *to RH*
    where $\xi = (r_1, r_2) \wedge r \neq r_1 \wedge r \neq r_2$

removeFromRange(a, ξ, r)
  if $\exists \langle ar, \xi \rangle \in$ *can_modify such that*
  a is a member of ar then
    remove $r_1 \succeq r$ *from RH*
    remove $r \succeq r_2$ *from RH*
    where $\xi = (r_1, r_2) \wedge r \neq r_1 \wedge r \neq r_2$

$addAsSenior(a, r, s)$
  if $\exists\langle ar, \xi\rangle \in$ *can_modify such that*
  a *is a member of* $ar \wedge r, s \in \xi$ *then*
    add $r \succeq s$ *to* RH

$removeAsSenior(a, r, s)$
  if $\exists\langle ar, \xi\rangle \in$ *can_modify such that*
  a *is a member of* $ar \wedge r, s \in \xi$ *then*
    remove $r \succeq s$ *from* RH

$Q, \vdash$ We allow queries of the following forms that are all natural for the ARBAC97 scheme: (1) given a role r, does there exist a user u such that $\langle u, r\rangle \in UA$?, (2) given user u, does there exist a role r such that $\langle u, r\rangle \in UA$?, (3) given user u and role r, is $\langle u, r\rangle \in UA$?, (4) given a permission p, does there exist a role r such that $\langle p, r\rangle \in PA$? (5) given permission p, does there exist a role r such that $\langle p, r\rangle \in PA$?, (6) given permission p and role r, is $\langle p, r\rangle \in PA$?, (7) given roles r_1, r_2, is $r_1 \succeq r_2 \in RH$?, and (8) give user u and permission p, is u authorized to have the permission p? That is, do there exist roles r_1, r_2 such that $\langle u, r_1\rangle \in UA \wedge \langle p, r_2\rangle \in PA \wedge r_1 \succeq r_2 \in RH$? The entailment relation, $\vdash$ is based simply on whether the conditions checked in a query hold in the given state.

Theorem 5.3.2 *There exists a reduction from SDCO to ARBAC97.*

Proof By construction. We present the mappings `Reduce` and `ReduceQuery` and show that they satisfy the properties for a reduction from SDCO to ARBAC97. `Reduce` takes as input the start-state and state-transition rules of an SDCO system and produces as output the start-state and state-transition rules of an ARBAC97 system. `ReduceQuery` takes as input a query in the SDCO system and produces as output a query in the ARBAC97 system. We assume, without loss of generality, that there is a one-to-one correspondence between the set of users $\mathcal{U}$ in ARBAC97 and the set of subjects $\mathcal{S}$ in SDCO, and between the set of roles $\mathcal{R}$ in ARBAC97 and the set $(\mathcal{O} \times R_\gamma) \cup \{subjectExists, A, top, bottom\}$, where $\mathcal{O}$ is the set of objects, and R_γ is the set of rights in the SDCO system, and $subjectExists$, A, top and $bottom$ are specific roles that are used in the mapping.

```
1 Subroutine Reduce(γ,ψ)
2   /* inputs: γ - an SDCO state, ψ - SDCO state-transition rules */
3   /* outputs: γ^A - an ARBAC97 state,
4               ψ^A - ARBAC97 state-transition rules */
5   initialize γ^A,ψ^A as follows:
6   UA = PA = RH = AR = can_assign = can_revoke = can_assignp = can_revokep = can_modify = ∅
7   add top ≽ bottom to RH; let ξ be the role range (top, bottom)
8   let the set of administrative roles AR = A; add (a, A) to UA
9   can_assign = {⟨A, true, ξ⟩}, can_revoke = can_modify = {⟨A, ξ⟩}
10  execute addToRange(a, ξ, subjectExists) where subjectExists is a role
11  foreach s ∈ S_γ execute assignUser(a, s, subjectExists)
12  execute removeFromRange(a, ξ, subjectExists)
13  foreach ⟨o,r⟩ ∈ O_γ × R_γ execute addToRange(a, ξ, o_r)
14  foreach r ∈ M_γ[s,o] execute assignUser(a, s, o_r)
15  return γ^A,ψ^A
16
17 Subroutine ReduceQuery(q)
18  /* input: q - an SDCO query */
19  /* output: q^A - an ARBAC97 query */
20  if q == s ∈ S then q^A = ⟨s, subjectExists⟩ ∈ UA
21  if q == o ∈ O then q^A = ∃ u such that ⟨u, o_own⟩ ∈ UA
22  if q == r ∈ M[s,o] then q^A = ⟨s, o_r⟩ ∈ UA
23  return q^A
```

We now show that property (1) for a reduction is satisfied by the above mapping. Let γ_0 be a start-state in SDCO. We produce the corresponding start-state γ_0^A in ARBAC97 using the Reduce subroutine above. Given a state γ_k and query q such that $\gamma_0 \overset{*}{\mapsto}_{\psi} \gamma_k$, we show that there exists γ_k^A and query q^A such that $\gamma_0^A \overset{*}{\mapsto}_{\psi^A} \gamma_k^A$ where $\gamma_k^A \vdash q^A$ if and only if $\gamma_k \vdash q$. If $\gamma_k = \gamma_0$, then $\gamma_k^A = \gamma_0^A$. If q is $s \in S$, then q^A is $\langle s, subjectExists \rangle \in UA$. By line 11 in Reduce q^A is true if and only if q is true. If q is $o \in O$, then q^A is $\exists\, u$ such that $\langle u, o_{own} \rangle \in UA$. By line 14 in Reduce, and the property that every object that exists in SDCO has an owner associated with it (that is, $own \in M[s,o]$ for some subject s), q^A is

true if and only if q is true. And if q is $r \in M[s, o]$, q^A is $\langle s, o_r \rangle \in UA$, and by line 14 of `Reduce`, q is true if and only if q^A is true.

Consider some γ_k reachable from γ_0 and a query q. We show the existence of γ_k^A that is reachable from γ_0^A and that answers q^A the same way by construction. If q is of type $s \in S$, we let $\gamma_k^A = \gamma_0^A$. if q is of type $o \in O$ or $r \in M[s, o]$, we do the following. We consider each state-transition in SDCO $\gamma_0 \mapsto_\psi \gamma_1 \mapsto \ldots \mapsto \gamma_k$. If the state-transition is the execution of $createObject(s, o)$, we execute $addToRange\,(a, \xi, o_{own})$ and $assignUser\,(a, s, o_{own})$. If the state-transition in SDCO is the execution of $destroyObject\,(s, o)$, we execute $revokeUser\,(a, u, o_r)$ for every $\langle u, o_r \rangle \in UA$ for every r, and $removeFromRange\,(a, \xi, o_{own})$. If the state-transition in SDCO is the execution of $grantOwn\,(s, s', o)$, we execute $revokeUser\,(a, s, o_{own})$ and $assignUser\,(a, s', o_{own})$. If the state-transition in SDCO is the execution of $grant\_r_i\,(s, s', o)$, we execute $assignUser\,(a, s', o_{r_i})$. Now, consider each possible query q. If q is $s \in S$, then $\gamma_k^A = \gamma_0^A$. In our SDCO scheme, the subjects are fixed at the start and never change. So $\gamma_k^A \vdash q^A$ if and only if $\gamma_0 \vdash q$. If q is $o \in O$, then $\gamma_k \vdash q$ if and only if o exists in the state γ_k. This is the case if and only if some subject s has the own right over o. This is the case if and only if we have the role o_{own} in the range ξ and the user corresponding to s is a member of that role. Therefore, $\gamma_k \vdash q$ if and only if $\gamma_k^A \vdash q^A$. And finally, if q is $r \in M[s, o]$, then $\gamma_k \vdash q$ if and only if r has been granted to s by the owner of o. This is true if and only if we have assigned the user corresponding to s to the role o_r. Thus, again, $\gamma_k \vdash q$ if and only if $\gamma_k^A \vdash q^A$.

We prove that property (2) for a reduction is satisfied by our mapping also by construction. Let γ_0^A be the start-state in ARBAC97 corresponding to γ_0, the start-state in SDCO. Then, if γ_k^A is a state reachable from γ_0^A and q^A is a query in ARBAC97 whose corresponding query in SDCO is q, we construct γ_k, a state in SDCO reachable from γ_0 as follows. If q is $s \in S$, we let $\gamma_k = \gamma_0$. Otherwise, for each role o_{own} that has a member s, we execute $createObject\,(s, o)$. For each role o_r that has a member s', if the role o_{own} has a member s, we execute $grant\_r\,(s, s', o)$. If q is $s \in S$, then q^A is $\langle s, subjectExists \rangle \in UA$, and clearly $\gamma_k^A \vdash q^A$ if and only if $\gamma_k \vdash q$, as the subjects that exist do not change from the

start-state in SDCO, and the members of $subjectExists$ do not change from the start-state in ARBAC97. If q is $o \in O$, $\gamma_k^A \vdash q^A$ if and only if $\exists\ s$ such that $\langle s, o_{own} \rangle \in UA$. And if q^A is true, we would have added the own right to $M_{\gamma_k}[s, o]$, which means that $\gamma_k \vdash q$ if and only if $\gamma_k^A \vdash q^A$. And finally, if q is $r \in M[s, o]$, $\gamma_k^A \vdash q^A$ if and only if $\langle s, o_r \rangle \in UA$. The condition that q^A is true is the only one in which we would have added the right r to $M_{\gamma_k}[s, o]$, and therefore $\gamma_k \vdash q$ if and only if $\gamma_k^A \vdash q^A$. ■

Before we introduce Theorem 5.3.4, we introduce the following lemma as an intermediate result on the state-change rules in ARBAC97. The intermediate result aids in the proof of the theorem.

Lemma 5.3.3 *Let ψ be a state-transition rule, and γ and γ' be states in the $ARBAC97$ scheme. Then, for any two queries q_1 and q_2, there exists no γ' such that $\gamma' \vdash (\neg q_1 \wedge q_2)$ when $\gamma \vdash (q_1 \wedge \neg q_2)$ and $\gamma \mapsto \gamma'$.*

Proof We observe that the operations $assignUser$, $assignPermission$, $addToRange$ and $addAsSenior$ can cause queries to become only true, and not false. Similarly, the operations $revokeUser$, $revokePermission$, $removeFromRange$ and $removeAsSenior$ cannot cause a query to become true. Therefore, given a state-transition in the ARBAC97 scheme, it cannot cause a query that is true to become false and another query that is false to become true in the new state. ■

Theorem 5.3.4 *There exists no state-matching reduction from SDCO to ARBAC97.*

Proof By contradiction. Assume that there exists a state-matching reduction from SDCO to ARBAC97. Let $\mathcal{S} = \{s_1, s_2, s_3, \ldots\}$. In SDCO, adopt as γ a state with the following properties. Let $s_1 \in S_\gamma$, $o \in O_\gamma$ and $own \in M[s_1, o]$. Let q_i be the query "$own \in [s_i, o]$" for each $i = 1, 2, \ldots$, and q_o be the query "$o \in O_\gamma$". These queries are mapped to q_i^A and q_o^A respectively in the ARBAC97 scheme. We observe that $\gamma \vdash (q_1 \wedge \neg q_2 \wedge \neg q_3 \wedge \ldots \wedge q_o)$. There exists a state $\tilde{\gamma}$ reachable from γ such that $\tilde{\gamma} \vdash (\neg q_1 \wedge q_2 \wedge \neg q_3 \wedge \ldots \wedge q_o)$. And, there exists no reachable state $\hat{\gamma}$ such that $\hat{\gamma} \vdash (q_1 \wedge \neg q_2 \wedge \ldots \wedge q_j \wedge \ldots \wedge q_o)$ or $\hat{\gamma} \vdash$

$(\neg q_1 \wedge \neg q_2 \wedge \ldots \wedge \neg q_j \wedge \ldots \wedge q_o)$ for any $j \neq 1$. (if $o \in O_\gamma$, then there must be exactly one subject that owns o). Consider the state γ^A in ARBAC97 that corresponds to γ (if there does not exist one, then we have the desired contradiction). We know that $\gamma^A \vdash \left(q_1^A \wedge \neg q_2^A \wedge \neg q_3^A \wedge \ldots \wedge q_o^A\right)$. There must also exist a reachable state $\tilde{\gamma}^A$ that corresponds to $\tilde{\gamma}$ (if there does not exist one, then we have the desired contradiction). By Lemma 5.3.3, we know that $\tilde{\gamma}^A$ is not reachable from γ^A is a single state-transition. Therefore, there must exist some state $\hat{\gamma}^A$ that is reachable from γ^A such that $\hat{\gamma}^A \vdash \left(q_1^A \wedge \neg q_2^A \wedge \ldots \wedge q_j \wedge \ldots \wedge q_o^A\right)$ or $\hat{\gamma}^A \vdash \left(\neg q_1^A \wedge \neg q_2^A \wedge \ldots \wedge \neg q_j^A \wedge \ldots \wedge q_o^A\right)$ for at least one $j \neq 1$. As there exists no corresponding state in the SDCO scheme that is reachable from γ, we have a contradiction to the assumption that there exists a state-matching reduction from SDCO to ARBAC97. ■

One may ask whether there are other schemes based on RBAC for which there is indeed a state-matching reduction from SDCO. An approach may be to adopt a different query set for ARBAC97. We observe that for certain other query sets as well, the non-existence of a state-matching reduction holds. As an example, suppose we map the query for the presence of a right in SDCO to a query for the absence of a permission in RBAC. In this case as well, there exists no state-matching reduction from SDCO. Whether there exists a meaningful set of state-transition rules (an administrative model) for RBAC for which there is a state-matching reduction from SDCO is an open problem.

5.3.4 Comparing an RBAC scheme with a trust management language

In this section, we compare a particular RBAC scheme to the trust management scheme, RT$[\cap]$. The RBAC scheme we consider is called Assignment And Revocation (AAR) [7]. In AAR, the state is an RBAC state, and state-transition rules are those from the URA97 component of the ARBAC97 [37]; users may be assigned to and revoked from roles.

RT$[\cap]$ is a trust management scheme in which a state is a set of credentials issued by the principals involved in the system. A credential denotes membership in a principal's role. A credential is one of three types: (1) A principal is asserted to be a member of

another principal's role, (2) All the principals that are members of a principal's role are asserted to also be members of another principal's role, and (3) All the principals that are members of two roles (the intersection of the members of the roles) are also members of another principal's role.

We first present precise characterizations of the AAR scheme and RT$[\cap]$. [7] present a form-2 weak reduction (see Definition 5.1.9) from AAR to RT$[\cap]$. We assert with the following theorem that the result can be made stronger.

The AAR Scheme

Γ In AAR, a state is the RBAC state $\langle UA, PA, RH \rangle$, as discussed in the previous section for ARBAC97.

Ψ The state-transitions allowed are the operations $assignUser$ and $revokeUser$ from the previous section, with the exception that negation is not allowed in pre-requisite conditions. In addition, in AAR, we require that for every role for which there is a can_assign entry, there is also a can_revoke entry. That is, if $\exists \langle ar, c, \xi \rangle \in can\_assign$ such that ar has at least one member and c may evaluate to $true$, then $\forall\, r \in \xi, \exists \langle ar', \xi' \rangle \in can\_revoke$ such that $r \in \xi'$ and ar' has at least one member.

$Q, \vdash$ Queries are of the form $s_1 \sqsupseteq s_2$, where s_1 and s_2 are *user-sets*. A user-set is an expression that evaluates to a set of users. A set of roles, a set of permissions and a set of users are user-sets, as are unions and intersections of user-sets. We refer the reader to [7] for more details on user-sets. Entailment involves evaluating the user-sets s_1 and s_2 to the sets of users S_1 and S_2 respectively, and determining whether $S_1 \supseteq S_2$. Several interesting queries related to safety, availability, liveness and mutual-exclusion can be posed as comparisons of user-sets.

The RT$[\cap]$ Scheme

Γ An RT$[\cap]$ state is a set of credentials, each of which is one of the following types: (1) $A.r \longleftarrow U$, (2) $A.r \longleftarrow B.r_1$, and (3) $A.r \longleftarrow B.r_1 \cap C.r_2$. Each of A, B, C, U is a principal, r, r_1, r_2 is a role name, and $A.r, B.r_1, C.r_2$ is a role. The symbol $\longleftarrow$ is read as

"includes". Statement (1) asserts that U is a member of A's r role. Statement (2) asserts that all members of the role $B.r_1$ are members of the role $A.r$. Statement (3) asserts that anyone that is a member of both $B.r_1$ and $C.r_2$ is a member of $A.r$.

Ψ A state-transition in $\mathsf{RT}[\cap]$ is either the removal of a credential, or the addition of one. State-transitions are controlled by *growth* and *shrink*-restricted sets of roles — G and S respectively. A role that is in the growth-restricted set may not have any assertions added with that role at the head of the assertion, and a role that is in the shrink-restricted may not have any assertions removed. Thus, the state-transition rules are represented as $\langle G, S \rangle$.

$Q, \vdash$ We allow queries of the form $c_1 \sqsupseteq c_2$ where each c_1 and c_2 is either an $\mathsf{RT}[\cap]$ role, a credential, or credentials joined by union, $\cup$ or intersection, $\cap$. We observe that this is slightly different from the definition for queries in [7]. The reason is that in that work, only a form-2 weak reduction (see Definition 5.1.9) is presented, and therefore queries are processed in conjunction with each state and state-transition rule in the mapping. We seek to map queries independently of states and state-transition rules. Entailment in $\mathsf{RT}[\cap]$ is done using credential chain discovery [70]: we find a chain of credentials that proves a (portion of a) query, if one exists.

Theorem 5.3.5 *There exists a state-matching reduction from the AAR scheme to* $\mathsf{RT}[\cap]$.

Proof By construction. We show that the mapping from [7] from AAR to $\mathsf{RT}[\cap]$ is a state-matching reduction. We consider each assertion from Definition 5.1.5 in turn. Each role r in AAR is associated with the role $Sys.r$ in $\mathsf{RT}[\cap]$. We show that after a series of state-transitions, the role-memberships in AAR match the role-memberships in the corresponding state of $\mathsf{RT}[\cap]$.

Assertion 1: Let γ be the given AAR state, and $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$. Then, $\gamma = \gamma_0 \mapsto_{\psi} \gamma_1 \ldots \mapsto_{\psi} \gamma_m = \gamma'$. Each state-transition is either the assignment of a user to a role using $assignUser$ or revocation of a user's membership in a role using $revokeUser$. Let the corresponding states in $\mathsf{RT}[\cap]$ be $\gamma^T = \gamma_0^T, \gamma_1^T, \ldots \gamma_m^T = \gamma^{T\prime}$. The users that are members of any role r in γ are the same as the users that are members of the corresponding role $Sys.r$ in γ^T. If the state-transition from γ_i to γ_{i+1} is the result of the assignment of the user u to the role

r, then we effect the following changes to transition from the state γ_i^T to γ_{i+1}^T: we add the two statements $\mathsf{ASys}.r \longleftarrow u$ and $\mathsf{BSys}.r \longleftarrow u$. If the state-transition is the result of the revocation of the user u from the role r, then we remove all statements that exist of the following two forms: $\mathsf{ASys}.r \longleftarrow u$ and $\mathsf{RSys}.r \longleftarrow u$. We observe that in $\gamma^{T'}$, any $\mathsf{HSys}.r$ has as members all users that were ever members of the role r. Consequently, in $\gamma^{T'}$, each $\mathsf{Sys}.r$ has as members those users that are members of r in γ'. Therefore, we can assert that $\gamma' \vdash q$ iff $\gamma^{T'} \vdash q^T$.

Assertion 2: In $\mathsf{RT}[\cap]$, the only roles that can grow are the ASys and BSys roles. The only roles that can shrink are the ASys and RSys roles. Given $\gamma^T = \sigma(\gamma)$ where γ is a given AAR state and $\gamma^{T'}$ is the corresponding $\mathsf{RT}[\cap]$ state, let $\gamma^T \overset{*}{\mapsto}_{\psi} \gamma^{T'}$. We construct the AAR state γ' that corresponds to $\gamma^{T'}$ as follows. For each statement of the form $\mathsf{BSys}.r \longleftarrow u$ or of the form $\mathsf{ASys}.r \longleftarrow u$, we assign the user u to the role r. Now, we compare the user-role memberships of each user to the roles r and $\mathsf{Sys}.r$. There cannot be any users in $\mathsf{Sys}.r$ that are not in r: the reason is that we have not revoked any user membership in r (starting from the user-role membership in the state γ). There may be users in r that are not in $\mathsf{Sys}.r$. Given the requirement that every role for which there is a can_assign, we also have a can_revoke, the only way for these extra users to be in r and not $\mathsf{Sys}.r$ is that there exists a can_assign that permits those users to be assigned to r (starting at the state γ). We revoke such users' membership from r using the relevant can_revoke entries. Now, the memberships in r and $\mathsf{Sys}.r$ are identical, and we can assert that for all queries q, $\gamma^{T'} \vdash \sigma(q)$ iff $\gamma' \vdash q$. ■

5.3.5 Comparing ATAM with TAM

TAM is a scheme based on the access matrix model and is similar to the HRU scheme [2]. Every object is typed, and the type cannot change once the object is created. State-transitions occur via the execution of commands that are similar to HRU commands. We specify a type for every parameter to a command. ATAM is the same as TAM, except that in a condition in an ATAM command, the absence of a right in a cell of the access matrix

may be checked (and not just the presence of a right). Below, we present characterizations of the two schemes.

Sandhu and Ganta [18] presents a mapping from the ATAM to TAM. Based on the mapping, one may conclude that TAM is at least as expressive as ATAM. As the converse is trivially true (TAM is a special case of ATAM), one may conclude that ATAM and TAM have the same expressive power; we gain nothing from the ability to check for the absence of rights in the condition of an ATAM command. Sandhu and Ganta [18] makes the observation that the simulation of a command in ATAM may require the execution of an unbounded number of commands in TAM, and concludes with the following comment: "...practically testing for the absence of rights appears to be useful. It is an open question whether this claim can be formalized..." In this section, we formalize this claim by asserting that there is no state-matching reduction from ATAM to TAM.

The TAM Scheme

Γ TAM is similar to the HRU scheme. Each state $\gamma \in \Gamma$ is $\langle S_\gamma, O_\gamma, M_\gamma[\,], R_\gamma, T_\gamma, typeOf \rangle$ where S_γ, O_γ, R_γ and T_γ are finite, strict subsets of the countably infinite sets $\mathcal{S}$ (subjects), $\mathcal{O}$ (objects), $\mathcal{R}$ (rights) and $\mathcal{T}$ (types of objects and subjects) respectively. The function $typeOf$: $(S_\gamma \cup O_\gamma) \rightarrow T_\gamma$, maps each subject and object to a type that cannot change once the subject or object is created. $M_\gamma[\,]$ is the access matrix.

Ψ A state-transition rule is a set of commands. Each command has an optional list of conditions that are joined by conjunction. A command then consists of primitive operations. Each parameter to the command is associated with a type. Each condition may check only for the presence of a right in a cell.

$Q, \vdash$ We allow queries of the form "is $r \in M[s, o]$?" Entailment is defined as follows. Given a state $\gamma \in \Gamma$, $\gamma \vdash r \in M[s, o]$ if and only if $s \in S_\gamma \wedge o \in O_\gamma \wedge r \in R_\gamma \wedge r \in M_\gamma[s, o]$.

The ATAM Scheme

$\Gamma, \Psi, Q, \vdash$ An ATAM state is the same as a TAM state. State-transition rules are the same as for TAM, except that a condition in a command may check for the absence of a right (as opposed to only the presence of a right). In ATAM, we allow Q to contain

queries of the following two forms: (1) Is $r \in M[s, o]$?, and (2) Is $r \notin M[s, o]$? This is consistent with the intent of [18] to determine whether the ability to check for the absence of rights does indeed add more expressive power. $\vdash$ is defined the same as in TAM for a query of type (1). For a query of the type (2), $\vdash$ is defined as follows. Given a state $\gamma \in \Gamma$, $\gamma \vdash r \notin M[s, o]$ if and only if $s \in S_\gamma \wedge o \in O_\gamma \wedge r \in R_\gamma \wedge r \notin M_\gamma[s, o]$.

Theorem 5.3.6 *There exists no state-matching reduction from ATAM to TAM.*

Proof By contradiction. Assume that there exists a state-matching reduction σ from ATAM to TAM. Consider an ATAM scheme in which ψ (the state-transition rule) consists of the following commands.

$$\begin{array}{cc} command\ createSubject(X\colon t) & command\ addRight(Y\colon t, Z\colon t) \\ create\ subject\ X\ of\ type\ t & enter\ r\ into\ [Y, Z] \end{array}$$

Adopt as γ_0 (the start state) in ATAM a state with no subjects or objects. (that is, $S_{\gamma_0} = O_{\gamma_0} = \emptyset$). The set of rights, $R_{\gamma_0} = \{r\}$, and there is a single type t for all subjects (no objects other than subjects exist or can be created in our ATAM system). We denote components of the TAM system under the mapping σ with a superscript T. For example, $\sigma(\gamma_0) = \gamma_0^T$ and $\sigma(\psi) = \psi^T$.

We assume that the countably infinite set of subjects $\mathcal{S} = \{s_1, s_2, \ldots\}$. In the ATAM system, we wish to consider queries of the form $q_{i,j} = r \in M[s_i, s_j]$ and $\widehat{q_{i,j}} = r \notin M[s_i, s_j]$ for some $s_i, s_j \in \mathcal{S}$. First, we make the observation that any two distinct queries $p, q \in \{q_{i,j} | s_i, s_j \in \mathcal{S}\} \cup \{\widehat{q_{i,j}} | s_i, s_j \in \mathcal{S}\}$ are mapped to distinct queries in TAM. That is, $p \neq q \Rightarrow p^T \neq q^T$. Otherwise, pick a pair p, q such that $p \neq q$ but $p^T = q^T$. For any two such queries p and q, there exists a state γ in ATAM such that $\gamma_0 \overset{*}{\mapsto}_\psi \gamma$ and $\gamma \vdash p \wedge \neg q$. Clearly, a corresponding reachable state (that answers the queries p and q the same way) does not exist in TAM, which gives us the desired contradiction. We observe also that by the definition of a state-matching reduction, queries are mapped independent of the start state and the state-change rules.

Consider ψ^T, the command schema in TAM. As a query in TAM is of the form $r \in M[s, o]$, we can determine an upper bound, m, for the number of queries a command in the TAM system can change from false to true when executed. These are queries of both types $q_{i,j}^T$ and $\widehat{q_{i,j}}^T$. One way to determine a value for m is to count the number of "$enter\ right$" primitive operations in each command and take the maximum (even though this maximum may not be a tight upper bound). m is constant, and may be dependant on γ and ψ, but not the set of queries. Choose some $n > m$.

Now, consider the state in ATAM γ_k such that $\gamma_0 \overset{*}{\mapsto}_\psi \gamma_k$ and $\gamma_k \vdash \neg q_{1,1} \wedge \widehat{q_{1,1}} \wedge \neg q_{1,2} \wedge \widehat{q_{1,2}} \wedge \ldots \wedge \neg q_{n,n} \wedge \widehat{q_{n,n}}$ (we use the subscript k only to distinguish the state, and not as a count of the number of state-changes needed to reach it). That is, γ_k does not entail any of the queries of the type $q_{i,j}$ and entails all queries of the type $\widehat{q_{i,j}}$ for all integers i, j such that $1 \leq i, j \leq n$. The state γ_k corresponds to $S_{\gamma_k} = \{s_1, \ldots, s_n\}$ with no right r in any of the cells. One way to reach this state from γ_0 is to execute the command $createSubject$ n times with the parameter instantiated to s_i in the i^{th} execution.

We assume that as σ, a state-matching reduction exists, there exists a corresponding rechable state γ_k^T in TAM that answers the (mapped) queries the same way. Consider any sequence $\gamma_0^T \mapsto_{\psi^T} \gamma_1^T \mapsto_{\psi^T} \ldots \mapsto_{\psi^T} \gamma_k^T$. Pick the first state, γ_c^T in the sequence that satisfies the following condition: $\gamma_c^T \vdash q_{i,j}^T \vee \widehat{q_{i,j}}^T$ for all integers i, j such that $1 \leq i, j \leq n$. Such a state exists: γ_k^T is such a state, and may be the only state in the sequence that meets the condition. We observe also that γ_0^T does not satisfy the condition, thereby implying that the sequence has at least one state-change.

Consider the state γ_{c-1}^T in the sequence just before γ_c^T. γ_{c-1}^T has the following property: there exist integers v, w with $1 \leq v, w \leq n$, such that $\gamma_{c-1}^T \vdash \neg \left(q_{v,w}^T \vee \widehat{q_{v,w}}^T \right) \Rightarrow \gamma_{c-1}^T \vdash \neg q_{v,w} \wedge \neg \widehat{q_{v,w}}^T$. For every state in the ATAM system that entails the corresponding formula of queries $\neg q_{v,w} \wedge \neg \widehat{q_{v,w}}$, the state also entails at least one of the following two formulae of queries: (1) Q_1 = $\neg q_{v,1} \wedge \neg \widehat{q_{v,1}} \wedge \neg q_{v,2} \wedge \neg \widehat{q_{v,2}} \wedge \ldots \wedge \neg q_{v,n} \wedge \neg \widehat{q_{v,n}} \wedge \neg q_{1,v} \wedge \neg \widehat{q_{1,v}} \wedge \ldots \wedge \neg q_{n,v} \wedge \neg \widehat{q_{n,v}}$, or, (2) Q_2 = $\neg q_{w,1} \wedge \neg \widehat{q_{w,1}} \wedge \neg q_{w,2} \wedge \neg \widehat{q_{w,2}} \wedge \ldots \wedge \neg q_{w,n} \wedge \neg \widehat{q_{w,n}} \wedge \neg q_{1,w} \wedge \neg \widehat{q_{1,w}} \wedge \ldots \wedge \neg q_{n,w} \wedge \neg \widehat{q_{n,w}}$.

The reason is that a state in ATAM that entails $\neg q_{v,w} \wedge \neg\widehat{q_{v,w}}$ is one in which either the subject s_v or s_w, or both do not exist ($v = w$ is allowed, and does not affect our arguments). None of the queries of either type $q_{i,j}$ or $\widehat{q_{i,j}}$ corresponding to a subject that does not exist in a state is entailed by the state. Therefore, in TAM, $\gamma_{c-1}^T \vdash Q_1^T \vee Q_2^T$ (where Q_1^T and Q_2^T are obtained from Q_1 and Q_2 respectively by adding the superscript T to each query in the formula).

Consider the state-change in TAM from γ_{c-1}^T to γ_c^T. It must change (at least) n queries that appear in Q_1^T or Q_2^T from false to true. This is not possible, as each state-change can change at most $m < n$ queries from false to true. We have the desired contradiction to the existence of a state-matching reduction from the ATAM scheme to the TAM scheme. ■

Thus, the notion of state-matching reductions formalizes the difference in expressive power between ATAM and TAM. One may ask whether there exists a reduction from ATAM to TAM. One may also ask whether reductions or state-matching reductions exist from ATAM to TAM when we allow TAM to contain queries of the type "is $r \notin M_\gamma[s, o]$?" as well (but a command only allows checking for the presence of a right in a cell in the condition). These are open questions.

5.4 A "simulation" of RBAC in strict DAC

We now informally describe a simulation of RBAC in strict DAC, the simplest form of DAC. The point of this simulation is to show that if precise requirements are not specified on simulations, then anything is possible.

The state of a strict DAC model is represented by an access matrix, which has one subject for each user and each role and one object for each permission. There is also one special subject admin, who is the creator and owner of every object in the system. All subjects are also objects. We use three rights, 'own', 'dc', and 'c'. We assume that the implementation of the strict DAC model provides the following functionality, it internally sorts all the objects and can return the first object, given an object o, it return the object next to o. The commands implemented in the strict DAC are as follows:

```
command create(s, o)
  create o;
  enter own into (s,o);
end;
command delete(s, o)
  if own ∈ (s,o)
  destroy o;
end;
command grant-dc(s1, s2, o)
  if own ∈ (s1,o)
  enter dc into (s2,o);
  enter c into (s2,o);
end;
command grant-c(s1, s2, o)
  if own ∈ (s1,o)
  enter c into (s2,o);
end;
command revoke-dc(s1, s2, o)
  if own ∈ (s1,o)
  remove c from (s2,o);
end;
command revoke-c(s1, s2, o)
  if own ∈ (s1,o)
  remove c from (s2,o);
end;
```

The addition of new users, roles, and permissions are carried out by the simulator in the straightforward way, i.e., have admin executes a creation command; admin then becomes the owner of these objects. When a new user-role assignment, (u, r), is added,

the following procedure is executed, observe that only constant space is needed for the simulation.

```
addUR(u,r) {
 run command grant-dc(admin , u, r);
 while (propagate());
}
propagate() {
 repeat = false;
 for every s,o1,o2 in the matrix {
  if c ∉(s,o2) && c ∈(s,o1) && c ∈(o1,o2) {
   run command grant-c(admin , s, o2);
   repeat = true;
 }}
 return repeat;
}
```

The procedures for adding a role-permission assignment and a role-role inheritance relationship is similar.

Whenever a user-role assignment is removed, the simulator executes the following procedure, which first clear all the propagated rights and redo the propagation.

```
removeUR(u,r) {
 if (dc ∈ (u,r)) {
  run command revoke-dc(admin , u, r);
  clear();
  while (propagate());
 }
}
clear() {
 for every s,o in the matrix {
```

```
  if c ∈(s,o) {
    run command revoke-c(admin , s, o2);
 }}
}
```

6 CONCLUSION

We have proved our thesis that there exists a theory based on reductions that preserve results of security analysis for comparing access control models. We have demonstrated the effectiveness of the theory by applying it in several cases. We have established new results regarding safety analysis in DAC and security analysis in RBAC that are related to our thesis.

Our theory is a significant advancement in access control. Our theory is not only sound in its own right, but applications of it have lead to results that provide considerable insight into the power of access control schemes. One of the applications solves an open problem in access control. Another counters what appeared to be persuasive claims about the expressive power of a particular access control model. A third confirms a conjecture that could not be proven without a sound theory like ours. And a fourth application disassociates the undecidability of safety in an access control scheme from its expressive power; the two are often coupled in existing literature. From a broader standpoint, our theory and its applications demonstrate that formal methods can continue to play an important role in computer security.

A question that arises is whether our theory can be applied in broader contexts of security than computer security. The answer is “yes”; we point out that provided a scheme can be represented as a four-tuple of states, state-change rules, queries and entailment, our theory can be applied.

There is considerable scope for future work on the issue of expressive power in the context of access control. We propose to use our theory to compare more models with each other. For instance, we would like to compare various versions of DAC and “layer” these versions based on their relative expressive power. Also, while our theory is based on capturing the notion of policies that can be represented and verified in an access control system, we do not believe that reductions and state-matching reductions capture all the

types of policies we would want to consider. For instance, a reasonable question to ask during a security audit may be: “Did Alice get her write access to a sensitive file only after her husband, Bob was given privileged access to the system?” This can be perceived as a policy issue, and we may want to express this as some expression involving queries.

Neither reductions nor state-matching reductions capture such query expressions. As part of our future work, we propose to expand our theory to include such policies. A related question regards the limits to extending our theory to consider more kinds of policies. This remains an open question.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[2] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[3] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM Operating Systems Review, 8(1):18-24, Jan 1974.

[4] G. Scott Graham and Peter J. Denning. Protection — Principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429. AFIPS Press, May 16–18 1972.

[5] Ninghui Li, William H. Winsborough, and John C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 123–139. IEEE Computer Society Press, May 2003.

[6] Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM*, 52(3):474–514, May 2005. Preliminary version appeared in *Proceedings of 2003 IEEE Symposium on Security and Privacy*.

[7] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 126–135, June 2004.

[8] Anita K. Jones. Protection mechanism models: Their usefulness. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 237–252. Academic Press, Inc., 1978.

[9] Ravi S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating systems. *Journal of the ACM*, 35(2):404–432, 1988.

[10] Ravi S. Sandhu. Expressive power of the schematic protection model. *Journal of Computer Security*, 1(1):59–98, 1992.

[11] Sylvia Osborn, Ravi S. Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, May 2000.

[12] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, April 2003.

[13] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[14] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, Mitre Corporation, November 1973.

[15] Leonard J. LaPadula and D. Elliott Bell. Secure computer systems: A mathematical model. Technical Report ESD-TR-73-278, Mitre Corporation, November 1973.

[16] National Computer Security Center. A guide to understanding discretionary access control in trusted systems, September 1987. NCSC-TG-003.

[17] Paul Ammann, Richard Lipton, and Ravi S. Sandhu. The expressive power of multi-parent creation in monotonic access control models. *Journal of Computer Security*, 4(2-3):149–165, January 1996.

[18] Ravi S. Sandhu and Srinivas Ganta. On testing for absence of rights in access control models. In *Proceedings of the Sixth Computer Security Foundations Workshop*, pages 109–118. IEEE Computer Society Press, June 1993.

[19] R.E. Ladner, N.A. Lynch, and A.L. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[20] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.

[21] Deborah D. Downs, Jerzy R. Rub, Kenneth C. Kung, and Carole S. Jordan. Issues in discretionary access control. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 208–218, April 1985.

[22] Patricia P. Griffiths and Bradford W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, 1976.

[23] Jon A. Solworth and Robert H. Sloan. A layered design of discretionary access controls with decidable safety properties. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 2004.

[24] Anita K. Jones, Richard J. Lipton, and Lawrence Snyder. A linear time algorithm for deciding security. In *17th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 33–41, October 1976.

[25] Richard J. Lipton and Lawrence Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[26] Paul Ammann and Ravi S. Sandhu. The extended schematic protection model. *Journal of Computer Security*, 1(3-4):335–383, 1992.

[27] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 122–136. IEEE Computer Society Press, May 1992.

[28] Masakazu Soshi. Safety analysis of the dynamic-typed access matrix model. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 106–121. Springer, October 2000.

[29] Masakazu Soshi, Mamoru Maekawa, and Eiji Okamoto. The dynamic-typed access matrix model and decidability of the safety problem. *IEICE Transactions on Fundamentals*, E87-A(1):190–203, January 2004.

[30] David F. Ferraiolo and D. Richard Kuhn. Role-based access control. In *Proceedings of the 15th National Information Systems Security Conference*, 1992.

[31] David F. Ferraiolo, Janet A. Cuigini, and D. Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th Annual Computer Security Applications Conference (ACSAC'95)*, December 1995.

[32] Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. A formal model for role-based access control using graph transformation. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 122–139, October 2000.

[33] Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. A graph-based formalism for RBAC. *ACM Transactions on Information and System Security*, 5(3):332–365, August 2002.

[34] Matunda Nyanchama and Sylvia Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3–33, February 1999.

[35] Ravi S. Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: preliminary description and outline. In *Proceedings of the Second ACM workshop on Role-based access control (RBAC 1997)*, pages 41–50, November 1997.

[36] Ravi S. Sandhu and Venkata Bhamidipati. Role-based administration of user-role assignment: The URA97 model and its Oracle implementation. *Journal of Computer Security*, 7, 1999.

[37] Ravi S. Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based aministration of roles. *ACM Transactions on Information and Systems Security*, 2(1):105–135, February 1999.

[38] Jason Crampton and George Loizou. Administrative scope and role hierarchy operations. In *Proceedings of Seventh ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 145–154, June 2002.

[39] Jason Crampton and George Loizou. SARBAC: A new model for role-based administration. Technical Report BBKCS-02-09, Birbeck College, University of London, UK, March 2002.

[40] Jason Crampton and George Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security*, 6(2):201–231, May 2003.

[41] Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. Decidability of safety in graph-based models for access control. In *Proceedings of the Seventh European Symposium on Research in Computer Security (ESORICS 2002)*, pages 229–243. Springer, October 2002.

[42] He Wang and Sylvia L. Osborn. An administrative model for role graphs. In *Proceedings of the 17th Annual IFIP WG11.3 Working Conference on Database Security*, August 2003.

[43] D. Elliott Bell. Secure computer systems: A refinement of the mathematical model. Technical Report ESD-TR-73-278, Mitre Corporation, April 1974.

[44] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, Mitre Corporation, March 1976.

[45] Sushil Jajodia and Ravi Sandhu. Towards a multilevel secure data model. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 50–59, Denver, CO, USA, 1991. ACM Press.

[46] Michael A. Harrison and Walter L. Ruzzo. Monotonic protection systems. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 461–471. Academic Press, Inc., 1978.

[47] Paul Ammann and Ravi S. Sandhu. Safety analysis for the extended schematic protection model. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 87–97, May 1991.

[48] T. Budd. Safety in grammatical protection systems. *International Journal of Computer and Information Sciences*, 12(6):413–430, 1983.

[49] Naftaly H. Minsky. Selective and locally controlled transport of privileges. *ACM Transactions on Programming Languages and Systems*, 6(4):573–602, October 1984.

[50] Rajeev Motwani, Rina Panigrahy, Vijay A. Saraswat, and Suresh Ventkatasubramanian. On the decidability of accessibility problems (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 306–315. ACM Press, May 2000.

[51] Ravi S. Sandhu. Undecidability of the safety problem for the schematic protection model with cyclic creates. *Journal of Computer and System Sciences*, 44(1):141–159, February 1992.

[52] William H. Winsborough and Ninghui Li. Safety in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 147–160, May 2004.

[53] Ajay Chander, Drew Dean, and John C. Mitchell. A state-transition model of trust management and access control. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 27–43. IEEE Computer Society Press, June 2001.

[54] Srinivas Ganta. *Expressive Power of Access Control Models Based on Propagation of Rights*. PhD thesis, George Mason University, 1996.

[55] Ravi S. Sandhu and Qamar Munawer. How to do discretionary access control using roles. In *Proceedings of the Third ACM Workshop on Role-Based Access Control (RBAC 1998)*, pages 47–54, October 1998.

[56] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1):71–127, February 2003.

[57] Ninghui Li and Mahesh V. Tripunitara. On Safety in Discretionary Access Control. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

[58] Qamar Munawer and Ravi S. Sandhu. Simulation of the augmented typed access matrix model (ATAM) using roles. In *Proceedings of INFOSECU99 International Conference on Information and Security*, 1999.

[59] Teresa Lunt. Access control policies: Some unanswered questions. In *Proceedings of the 2nd IEEE Computer Security Foundations Workshop*, pages 227–245. IEEE Computer Society Press, June 1988.

[60] Pierangela Samarati and Sabrina de Capitani di Vimercati. Access control: Policies, models, and mechanisms. In Ricardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer, 2001.

[61] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, May 1994.

[62] Andreas Schaad, Jonathan Moffett, and Jeremy Jacob. The role-based access control system of a European bank: A case study and discussion. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 3–9. ACM Press, 2001.

[63] Sejong Oh and Ravi S. Sandhu. A model for role admininstration using organization structure. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, June 2002.

[64] David F. Ferraiolo, Ravi S. Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, August 2001.

[65] Gail-Joon Ahn and Ravi S. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4):207–226, November 2000.

[66] Trent Jaeger and Jonathon E. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information and System Security*, 4(2):158–190, May 2001.

[67] Jason Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 43–50. ACM Press, June 2003.

[68] David F. Ferraiolo, R. Chandramouli, Gail-Joon Ahn, and Serban Gavrila. The role control center: Features and case studies. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, June 2003.

[69] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[70] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, February 2003.

[71] Mahesh V. Tripunitara and Ninghui Li. Comparing the expressive power of access control models. In *Proceedings of 11th ACM Conference on Computer and Communications Security (CCS-11)*, pages 62–71. ACM Press, October 2004.

[72] Michael R. Garey and David J. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[73] Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management, 2004. Accepted to appear in *Journal of the ACM*.

[74] Ninghui Li and John C. Mitchell. RT: A role-based trust-management framework. In *The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*. IEEE Computer Society Press, April 2003.

[75] Sylvia Osborn. Mandatory access control and role-based access control revisited. In *Proceedings of the Second ACM Workshop on Role-Based Access Control (RBAC'97)*, pages 31–40, November 1997.

VITA

VITA

Mahesh's undergraduate education was completed at Dalhousie University, Halifax, Canada, from where he received a Bachelor of Science (Honors) degree in computer science in 1993. He received a Master of Science degree in computer science from Purdue University, West Lafayette, Indiana in 1995. He worked in the software development and consulting industries for almost eight years before returning to Purdue in fall 2003 to complete a Ph.D. in computer science.