2015 IEEE 28th Computer Security Foundations Symposium

Decomposing, Comparing, and Synthesizing Access Control Expressiveness Simulations

William C. Garrison III
Department of Computer Science
University of Pittsburgh
bill@cs.pitt.edu

Adam J. Lee
Department of Computer Science
University of Pittsburgh
adamlee@cs.pitt.edu

***Abstract*—Access control is fundamental to computer security, and has thus been the subject of extensive formal study. In particular, *relative expressiveness analysis* techniques have used formal mappings called *simulations* to explore whether one access control system is capable of emulating another, thereby comparing the expressive power of these systems. Unfortunately, the notions of expressiveness simulation that have been explored vary widely, which makes it difficult to compare results in the literature, and even leads to apparent contradictions between results. Furthermore, some notions of expressiveness simulation make use of non-determinism, and thus cannot be used to define mappings between access control systems that are useful in practical scenarios. In this work, we define the minimum set of properties for an *implementable* access control simulation; i.e., a deterministic "recipe" for using one system in place of another. We then define a wide range of properties spread across several dimensions that can be enforced on top of this minimum definition. These properties define a taxonomy that can be used to separate and compare existing notions of access control simulation, many of which were previously incomparable. We position existing notions of simulation within our properties lattice by formally proving each simulation's equivalence to a corresponding set of properties. Lastly, we take steps towards bridging the gap between theory and practice by exploring the systems implications of points within our properties lattice. This shows that relative expressive analysis is more than just a theoretical tool, and can also guide the choice of the most suitable access control system for a specific application or scenario.**

I. Introduction

Access control is foundational to computer security and, as such, has been the topic of extensive formal study. Much of this work has focused on comparing different techniques for representing and enforcing access control, deemed access control *models*, *systems*, or *schemes*. By far the most common type of comparative study in access control techniques is the expressiveness simulation (e.g., [1]–[14]). A simulation is a formal mapping from, say, system $\mathcal{S}$ to system $\mathcal{T}$ that proves $\mathcal{T}$ is at least as expressive as $\mathcal{S}$: that is, $\mathcal{T}$ possesses the raw capability to be used in operating environments in place of $\mathcal{S}$.

However, the formal definitions of the various simulations used in the literature vary widely. Different simulations have been used to prove various types of results, ranging from very specific properties about whole ranges of models (e.g., monotonic access control models with multi-parent creation cannot be simulated by monotonic models with only single-parent creation [6]) to the ability to replace certain specific models with others in practice (e.g., role-based access control can be configured to enforce mandatory and discretionary policies [9]). However, this disparity in the goals of these works has led to many different definitions of access control simulation, often tailored to the particular result sought. It has been shown that these different simulations prove wildly different notions of expressiveness, often not preserving any particular security properties [13].

Furthermore, not all of these notions of simulation are practically useful. For instance, some make use of non-determinism, manipulating the policy differently depending on what future queries will be asked. While this may allow a theorist to show that system $\mathcal{T}$ is capable of doing all the things $\mathcal{S}$ is, if a practitioner wants to use system $\mathcal{T}$ in place of system $\mathcal{S}$, she needs a deterministic procedure for doing so.

In this work, we build a taxonomy for expressiveness simulations based on the simulation properties that they satisfy. We determine the minimum requirements for a mapping to be *implementable*, or applicable toward using one system in place of another in practice. We use these requirements to construct a general definition of implementable simulation, and provide a taxonomy of additional restrictions on this definition for simulations that enforce more stringent properties. We then position existing simulations from the literature within this lattice, providing the first such comparison in the literature.

To this end, we make the following contributions.

Definition of implementable access control simulation We propose a general definition of an *implementable* access control mapping that is broad enough to encompass much of the wide range of existing access control simulations, yet precise enough to guarantee implementability. Intuitively, an *implementable* simulation of $\mathcal{S}$ in $\mathcal{T}$ shows that $\mathcal{T}$ can accomplish everything $\mathcal{S}$ can, and deterministically shows *how* (Section III).

Lattice of simulation properties We decompose and expand upon the properties enforced by various access control simulations from the literature, forming a lattice relating the range of access control simulations to one another. This lattice allows us to formally compare the guarantees offered by existing notions of access control simulation (many of which were not formerly known to be comparable) and points to unexplored combinations of properties that can yield different expressiveness results (Section IV).

DOI 10.1109/CSF.2015.9

IEEE computer society

Positioning of existing simulations We construct formal proofs positioning existing notions of access control simulation within our lattice of simulation properties, including a comparative discussion of simulations that previously seemed incomparable. We thus systematize the formal relationships between previously-published simulations, allowing reconciliation of previously disparate expressiveness knowledge (Section V).

Selecting simulation properties We observe that many of the dimensions upon which our simulation property lattice is built have implications for the use of simulations for satisfying real-world requirements using existing access control systems (e.g., required storage, whether data structures must be locked for concurrent usage). Thus, in addition to positioning existing notions of simulation within our lattice of properties, we assist in creating new notions of simulation by selecting the properties that should be enforced in an expressiveness analysis based upon the scenario in which an eventual access control deployment will occur. To this end, we discuss in detail various interactions between simulation properties, the results of enforcing different properties, and how a specific deployment scenario dictates which properties are relevant (Section VI).

We begin by providing background on the goals of and techniques used in relative expressiveness analysis.

II. Relative Expressiveness Analysis

In this section, we describe how relative expressiveness analysis is conducted, survey the history of the technique, and point out the wide variety in existing access control expressiveness simulations.

A. Motivating Examples

An access control system's *expressiveness* (or expressive power) is a measure of the range of policies that it can represent and the transformations it can make to those policies. Statements of *relative* expressiveness state that one system is capable of replacing another (that is, it can represent all the same policies and transform them in equivalent ways). Assume, for instance, that an organization is considering transitioning from one access control solution to another, in order to accommodate evolving requirements. The organization may have specific desired features for this new access control system, but it certainly must be able to represent all of the policies that the existing system can, or it would not be a suitable replacement. Thus, this organization is searching for a new system that is *at least as expressive as* its old system.

Another use of relative expressiveness is in suitability analysis. Prior work has noted that *practically* evaluating an access control system must take into account the application in which the system is to be used, as well as additional cost metrics (e.g., computation, ease of use). This analysis problem has been identified as a system's *suitability* to a particular application [15], [16]. Suitability analysis formalizes an application's access control requirements (a *workload*), and uses expressiveness to prove that an access control system can satisfy those requirements. Assume, in this case, that the aforementioned organization is choosing an initial access control system for a new collection of data. Comparing the candidates' relative expressiveness is not particularly enlightening, since the most expressive system may not be the most suitable; the organization should instead formalize their access control workload and use relative expressiveness analysis to identify which of the candidates are *expressive enough* to satisfy this workload. Thus, while work in suitability analysis has shown that expressive power alone is insufficient for evaluating an access control system, expressiveness is a fundamentally important component of a more general suitability analysis workflow: one cannot determine which access control system is *best* for a particular use case without first determining which are *capable of satisfying* that use case.

B. Prior Work

Relative expressiveness analysis generally starts by formalizing a pair of access control systems as state machines. These state machines include, at a minimum: a set of states, each of which encapsulates a snapshot of the access control system's data structures; a procedure describing how to interpret the states' data structures to determine which authorization requests are granted; and a set of commands, used to manipulate the data structures and thus transition between states. Some formalisms for access control systems also include additional queries beyond access requests [13], [14]. A *simulation*, then, is a structure that proves $\mathcal{T}$ is at least as expressive as $\mathcal{S}$—or, that $\mathcal{T}$ can be used in place of $\mathcal{S}$. The term *simulation* is rather vague, here, and for good reason: various notions of simulation in the literature have meant very different things (e.g., What type of behavior must be simulated? How closely must $\mathcal{T}$ represent the information in $\mathcal{S}$?), and as a result have implied very different types of expressiveness results.

The works of Sandhu, Ganta, Munawer, and Osborn [2]–[4], [7]–[9] include some of the earliest access control simulations. In these works, a simulation of $\mathcal{S}$ in $\mathcal{T}$ must show that a permission can be granted in $\mathcal{S}$ if and only if it can also be granted in $\mathcal{T}$. No other formal properties are enforced, though in some cases additional properties become part of the *de facto* definition of simulation. For instance, while there is no requirement for $\mathcal{T}$ to have a state equivalent to each $\mathcal{S}$ state (merely for $\mathcal{T}$ to be able to grant each access that $\mathcal{S}$ does, in some state), the example simulations all include methods for mapping each $\mathcal{S}$ state to a $\mathcal{T}$ state (as this is the simplest way to show the required property). In addition, although the definition does not prohibit the use of an unbounded number of $\mathcal{T}$ commands to simulate a single $\mathcal{S}$ command, Sandhu and Munawer [7] only use simulations in which an $\mathcal{S}$ command is simulated using a constant number of $\mathcal{T}$ commands.

Ganta's PhD dissertation [5] attempts to formalize a more rigorous notion of expressiveness simulation. In his simulation, the state correspondence is explicit, requiring that each state in $\mathcal{S}$ have a corresponding state in $\mathcal{T}$ that grants all the same accesses (at least, all those that exist in $\mathcal{S}$—those that exist in $\mathcal{T}$ but not in $\mathcal{S}$ are unconstrained). In addition, to ensure that $\mathcal{T}$ cannot grant accesses that $\mathcal{S}$ cannot, any state that can be entered in $\mathcal{T}$ must also have a corresponding reachable state in

$\mathcal{S}$. Finally, to ensure accesses in $\mathcal{T}$ cannot be combined in ways that cannot occur in $\mathcal{S}$, the following restriction is made: when simulating a $\mathcal{T}$ command in $\mathcal{S}$, multiple commands may be used, but each state along the way must allow *either* a subset of the accesses of the start state or a subset of the accesses of the end state. Thus, no two accesses can be allowed in the same state in $\mathcal{T}$ that are not allowed in a single state in $\mathcal{S}$.

Ammann, Lipton, and Sandhu [1], [6] took a different (and much more strict) approach to more rigorously defining a simulation. First, they describe a strict state correspondence that requires $\mathcal{T}$ to represent its states with the same sets and relations as $\mathcal{S}$, and for these sets to have identical contents in corresponding $\mathcal{T}$ and $\mathcal{S}$ states. In other words, $\mathcal{T}$ cannot include additional elements in any sets that $\mathcal{S}$ uses (although additional, distinct sets may be stored). For example, one could simulate the state $\{U = \{a, b\}, V = \{c\}\}$ with state $\{U = \{a, b\}, V = \{c\}, W = \{\langle a, d\rangle, \langle b, d\rangle\}\}$, but not with $\{U = \{a, b\}, V = \{c, d\}\}$. Given this notion of state correspondence, a simulation then shows that $\mathcal{T}$ can reach a state corresponding to each reachable $\mathcal{S}$ state, and cannot reach any state that does not have a reachable corresponding state in $\mathcal{S}$. This strict notion of simulation is used to show that monotonic, multi-parent systems are more expressive than monotonic, single-parent systems (e.g., there are monotonic multi-parent systems that cannot be simulated by any monotonic single-parent system).

Chander, Dean, and Mitchell [10] restrict the definition of simulation in a different way. Rather than force a more strict state correspondence (the static portion of the simulation), they more tightly restrict the way the simulation handles the system as it executes (i.e., the command mapping). In these simulations, the state correspondence is comparatively lax: to simulate an $\mathcal{S}$ state, a $\mathcal{T}$ state must allow and deny all the same authorization requests as its corresponding $\mathcal{S}$ state. Additional requests can exist in $\mathcal{T}$ and are unconstrained, but all requests corresponding to those in $\mathcal{S}$ must have the same value in corresponding states. However, the process for simulating an $\mathcal{S}$ command using $\mathcal{T}$ commands must be independent of the state: it cannot execute a $\mathcal{T}$ command for each user, or otherwise inspect the state when determining what commands should be executed. In addition, in the strong form of simulation, each $\mathcal{S}$ command must be simulated with a single $\mathcal{T}$ command. They then compare the expressiveness of access control lists, trust management, and two forms of capability systems (all systems studied in forms with and without revocation and delegation).

Tripunitara and Li [12], [13] noted that the existing notions of simulation did not correspond directly to any particular safety analysis questions, and thus a simulation of any of these types does not make any particular safety guarantees. They formalize *compositional security analysis* (intuitively, determining whether a certain set of access control queries will always, never, or sometimes become true in any reachable state), which is a generalization of simple safety analysis [17]. They then present a notion of simulation tailor-made to preserve these types of analysis questions.

Their simulation, called the state-matching reduction, considers a broader range of queries than only authorization requests, placing the strictness of its state correspondence somewhere between the work of Ammann, Lipton, and Sandhu and that of Chander, Dean, and Mitchell. The state-matching reduction maps each query $q^{\mathcal{S}}$ in $\mathcal{S}$ to a single query $q^{\mathcal{T}}$ in $\mathcal{T}$, and the simulation must determine the value of $q^{\mathcal{S}}$ in any state in $\mathcal{T}$ by checking the value of $q^{\mathcal{T}}$. Finally, reachability constraints ensure that $\mathcal{T}$ can reach a state corresponding to each reachable $\mathcal{S}$ state, and cannot reach any state that does not have a reachable corresponding state in $\mathcal{S}$. Tripunitara and Li prove that this notion of simulation preserves compositional security analysis instances: that is, if there exists a state-matching reduction from $\mathcal{S}$ to $\mathcal{T}$, then any compositional security analysis instance has the same truth value in both systems. Tripunitara and Li's reductions have since been used to analyze role-based access control [18] and prove that newly-proposed systems are more expressive than certain existing systems [19].

Work by Hinrichs et al. [14] recognizes the value of the state-matching reduction but claims that, in practice, not all scenarios require the preservation of all possible compositional security analysis instances (nor are these the only types of safety properties that are ever relevant). They present *parameterized expressiveness*, which defines a baseline set of simulation properties, and provides several additional properties that can be enforced atop the baseline to provide additional guarantees. The base simulation uses the same query-based state correspondence as Tripunitara and Li, but relaxes the query mapping to allow it to consult multiple $\mathcal{T}$ queries to determine the value of an $\mathcal{S}$ query during simulation. Further properties enforced above this baseline include using the identity query mapping for authorization requests (to ensure that $\mathcal{T}$'s authorization questions are the queries being used to simulate $\mathcal{S}$'s authorization requests), forbidding string manipulations (to prohibit the state mapping from using arbitrary encodings to store information in the contents of strings such as user names), and restricting the command mapping from mapping non-administrative commands in $\mathcal{S}$ to administrative commands in $\mathcal{T}$. This framework has since been used to evaluate the suitability of certain general-purpose access control systems for various unique, application-specific requirements [15], [16].

C. Usage and Implications

Unfortunately, there are several indications that research on expressiveness analysis is being held back by the inability to reconcile the vastly different notions of expressiveness simulations and the disconnect between the properties preserved by a simulation and those that are important to a practical deployment. Several works have demonstrated scenarios in which static notions of expressiveness indicate two systems are equally capable of satisfying a set of operational requirements, but in practice they are better-suited to very different deployment scenarios [15], [20]. Bourdier et al. point out the existence of several competing techniques for expressiveness analysis, none of which consider the deployment. They approach one facet of this problem by proposing a formalism for access control systems that can more easily be transformed into implementations using rewrite-based tools [21]. Several others simply express a desire to use expressiveness analysis, but never do so, presumably due to the complexities of selecting and using the right notion of simulation [22], [23].

A group at the National Institute of Standards and Technology has developed Policy Machine, an attempt at a universal access control system (one that can represent any policy via only configuration changes) [24]. However, in evaluating Policy Machine's success, they avoid formally proving its expressiveness and instead show informal mappings that demonstrate how one might use Policy Machine to represent several existing access control systems' policies [25]. Soon after, the group published a report bemoaning the lack of quality metrics for evaluating access control systems, noting that, in access control, "one size does not fit all," and thus said metrics must consider the deployment scenario [26].

This overview illustrates that while each notion of expressiveness simulation has been used to prove various results, the body of knowledge is troublesome to interpret and utilize due to the wide variation in the properties required by each simulation. In this work, we fill this void in the literature by (1) proposing a minimal definition of simulation that satisfies properties guaranteeing that its results are practically useful; (2) presenting a set of additional properties that more strict simulations can enforce; and (3) categorizing the above notions of simulation based on the properties that they enforce. We make the additional contribution of (4) discussing relationships between attributes of a deployment scenario and the practical effects of enforcing simulation properties, thus assisting analysts in selecting the most relevant properties (and therefore conducting the most relevant form of expressiveness analysis) for the environment in which an access control system will be deployed.

III. Implementable Expressiveness Simulations

In this section, we give requirements for a simulation to be *implementable* and define our general formulation of relative expressiveness analysis through the lens of implementability.

A. Implementability Requirements

In this work, we aim to consider expressiveness simulations that are *implementable*: i.e., practically useful for making decisions about which system is most suitable for a particular deployment. Implementability enforces the following intuition: if a system $\mathcal{T}$ is at least as expressive as $\mathcal{S}$, then one should be able to determine a general way to use $\mathcal{T}$ in place of $\mathcal{S}$. Thus, we define a minimal set of properties for an expressiveness mapping to be considered implementable.

State mapping In order to use $\mathcal{T}$ in place of $\mathcal{S}$, it must be possible to (uniquely) determine which $\mathcal{T}$ state to use in place of a particular $\mathcal{S}$ state. Thus, the state mapping must be a function from the simulated system states to the simulating system states.[1]

Command mapping To use $\mathcal{T}$ in place of $\mathcal{S}$, it must be possible to execute commands in $\mathcal{T}$ that are equivalent to the commands in $\mathcal{S}$. It is not necessarily the case that each $\mathcal{S}$ command can be simulated using a single $\mathcal{T}$ command, so we require a function from $\mathcal{S}$ commands to *sequences of* $\mathcal{T}$ commands.[2] Finally, it may be necessary to map an $\mathcal{S}$ command differently depending upon the state in which it is intended to be executed. Since using $\mathcal{T}$ in place of $\mathcal{S}$ means we only have a $\mathcal{T}$ state to inspect during execution, this function should map an $\mathcal{S}$ command and a $\mathcal{T}$ state to a sequence of $\mathcal{T}$ commands.

Query decider For some simulations of $\mathcal{S}$ in $\mathcal{T}$, we may only care that $\mathcal{T}$ allows the same set of accesses that $\mathcal{S}$ would. However some types of simulations may allow the overriding of $\mathcal{T}$'s default method of deciding granted permissions (e.g., adding the additional requirement that the requesting user is a member of the `REAL_USERS` group, to distinguish from other data stored in the user-set). While some types of simulations do not allow this, to remain general we simply require a function that maps each $\mathcal{S}$ query and $\mathcal{T}$ state to either true or false. In some formalisms, this only includes the queries requesting access, while in other cases other types of queries are allowed (e.g., "Is user u a member of role r?").

We use these requirements to motivate our definition of the general case of implementable relative expressiveness.

B. Expressiveness Mappings

To define relative expressiveness mappings, we must first define the state machines that represent access control systems. Since we aim to compare existing expressiveness simulations, we use a formalism for these structures that remains similar to existing work, e.g., [10], [12], [13].

An access control system is formalized as a state machine belonging to a particular access control *model*. An access control model formalizes the way in which the access control system will store and interpret information to make access control decisions. Its data structures are formalized as a set of access control *states*, and its methods for determining whether to allow or deny inquiries as a set of *authorization requests*. The value of all requests in a state (whether they are allowed or denied) defines the access control policy, or *theory*, to be enforced in that state.

Definition 1 *An* access control model *is defined as* $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$*, where* Γ *is the set of states and* $\mathcal{R}$ *is the set of authorization requests, where each request* $r \in \mathcal{R}$ *is a function* $\Gamma \rightarrow \{\text{TRUE}, \text{FALSE}\}$. *The entailment* ($\vdash$) *of a request is defined as* $\gamma \vdash r \triangleq r(\gamma) = \text{TRUE}$. ◊

For example, consider a simple role-based access control model whose states are defined over sets U of users, P of permissions, and R of roles, as well as the user assignment $UR \subseteq U \times R$ and permission assignment $PA \subseteq R \times P$. The requests in this model are of the form "Is u authorized for p?," which is TRUE if $\exists r : \langle u, r \rangle \in UR \wedge \langle r, p \rangle \in PA$.

[1] It is possible that multiple states in $\mathcal{S}$ can be represented using the same state in $\mathcal{T}$. Thus, we do not require the state mapping to be an injection. Furthermore, there may be states in $\mathcal{T}$ that are not used to simulate $\mathcal{S}$, and thus the state mapping need not be an surjection.

[2] Not *sets* of $\mathcal{T}$ commands, as commands may appear multiple times; and not *bags* of $\mathcal{T}$ commands, as order matters.

When we refer to the *size* of a state, we are referring to the size of its decomposition into primitive objects (e.g., users and roles) and tuples (e.g., entries in a user assignment relation).

Definition 2 *Given an access control model $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$ and a state $\gamma \in \Gamma$, the* set decomposition *of γ is denoted $[\gamma]$, and refers to the "set of sets" forming γ, in which γ is represented as being comprised of primitive sets and relations.* ◊

Thus, the size of an access control state γ is defined as $|\gamma| = \sum_{S \in [\gamma]} |S|$. For example, if $[\gamma] = \{U = \{u_1\}, R = \{r_1, r_2\}, UR = \{\langle u_1, r_1 \rangle, \langle u_1, r_2 \rangle\}\}$, then $|\gamma| = |U| + |R| + |UR| = 5$.

An access control *system* expands on a model by providing methods of transforming the current state and additional methods of querying the states. These additional queries allow the user to ask additional boolean queries of the system, but a value of TRUE does not indicate an authorization was granted.

Definition 3 *Given access control model $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$, an* access control system *within $\mathcal{M}$ is a state transition system, $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, where Ψ is the set of commands, where each command $\psi \in \Psi$ is a function $\Gamma \to \Gamma$, and $Q \supseteq \mathcal{R}$ is the set of queries, where each query $q \in Q$ is a function $\Gamma \to \{\text{TRUE}, \text{FALSE}\}$.* ◊

We use the notation *next*(γ, ψ) to denote the state resulting from executing ψ in γ (that is, $\psi(\gamma)$), and $terminal(\gamma, \psi_1 \circ \psi_n)$ to denote the final state produced by repeatedly applying *next* to the commands $\psi_1, \ldots, \psi_n$ starting from state γ: *next*$(\ldots$ *next*$(\gamma, \psi_1), \ldots, \psi_n)$.

A system based on the example role-based model must define commands to transform the state: e.g., to assign roles to users, and assign permissions to roles. Additional queries beyond the model's requests may include those of the form "Is user u a member of role r?"

Next, we define an access control mapping, which maps one system to another but does not enforce any simulation properties. We define a mapping as motivated in Section III-A so that it can represent any implementable expressiveness simulation.

Definition 4 *Given two access control systems, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$ and $\mathcal{T} = \langle \Gamma^{\mathcal{T}}, \Psi^{\mathcal{T}}, Q^{\mathcal{T}} \rangle$, a* mapping *from $\mathcal{S}$ to $\mathcal{T}$ is a triple of functions $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, where:*

- *$\sigma_\Gamma : \Gamma^{\mathcal{S}} \to \Gamma^{\mathcal{T}}$ is the state mapping*
- *$\sigma_\Psi : \Psi^{\mathcal{S}} \times \Gamma^{\mathcal{T}} \to (\Psi^{\mathcal{T}})^*$ is the command mapping*
- *$\sigma_Q = Q^{\mathcal{S}} \times \Gamma^{\mathcal{T}} \to \{\text{TRUE}, \text{FALSE}\}$ is the query decider*

This definition is demonstrated in Fig. 1. Each function takes its most general form that satisfies the requirements from Section III-A. Thus, the definition remains general (it does not enforce any specific security requirements yet), while ensuring that any such mappings can generate implementable procedures for using the simulating system in place of the simulated system.

To demonstrate Definition 4, consider mapping a simple access control list system to the role-based system described throughout this section. The state mapping can map each ACL state to a role-based state in which each user u has a unique role r_u, and each user's role is assigned the permissions from the ACL state. The command mapping can map, e.g., "grant u access to o" to "assign o to role r_u." The query mapping would then map "Can u access o?" to "Is u authorized for o?"

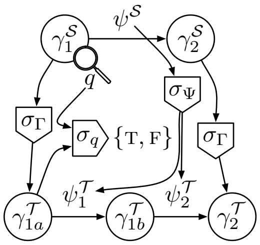
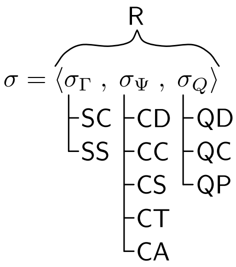


Fig. 1: The general form of an implementable expressiveness mapping.


R
$\sigma = \langle \sigma_\Gamma \ , \ \sigma_\Psi \ , \ \sigma_Q \rangle$
SC
SS
CD
CC
CS
CT
CA
QD
QC
QP

Symbol	Description
SC	State correspondence
SS	State storage
CD	Command mapping dependence
CC	Command mapping complexity
CS	Command mapping stuttering
CT	Trace structure
CA	Actor preservation
QD	Query decider dependence
QC	Query decider complexity
QP	Query preservation
R	Reachability

Fig. 2: An overview of the dimensions of expressiveness simulation properties

IV. Expressiveness Simulation Properties

In this section, we describe the lattice of properties that we use to taxonomize access control expressiveness simulations.

A. Overview of dimensions of properties

In order for a mapping to be considered a simulation, it must enforce additional properties over Definition 4. We restrict this definition in a variety of ways. Although no set of restrictions can be shown to be *the* full, correct set for all conceivable simulations, there are naturally three categories of restrictions to consider for simulations, given their structure (a set of three functions): i.e., refinements to each of the state correspondence, command mapping, and query decider. We also consider restrictions to the reachability constraints required (a cross-cutting dimension describing how these functions must relate to one another). A summary of these dimensions is depicted in Fig. 2.

Our state correspondence σ_Γ can be based on any of a handful of structural definitions, defined by SC (i.e., what elements do we inspect to determine whether two states correspond?). Further, SS can limit the amount of storage the state correspondence uses (e.g., $\mathcal{T}$ must simulate $\mathcal{S}$ using only a linear amount of additional storage).

The command mapping σ_Ψ can be restricted by CD in what state elements it can use to map commands (e.g., whether it can inspect arbitrary state elements or only those that are exposed via queries). CC considers limiting the time-complexity of the command mapping routine. Since the command mapping returns a *sequence* of commands, CS can limit the number of commands it can return (e.g., only one, or constant in the size of the state). We identify CT, a dimension of concurrency-related trace structure restrictions, as well as CA, requiring the simulation to map $\mathcal{S}$ commands executed by certain types of users only to $\mathcal{T}$ commands executed by that category of users.

The query decider σ_Q can also be restricted in a number of ways. Like the command mapping, we may limit what elements of the state the decider can inspect when deciding how to answer queries within a specific state (QD), or the time-complexity of the routine (QC). In some cases a simulation of $\mathcal{S}$ in $\mathcal{T}$ may be required to map certain $\mathcal{S}$ queries to specific related queries in $\mathcal{T}$, most notably authorization requests (e.g., to answer whether user u should have permission p in $\mathcal{S}$, $\mathcal{T}$ should simply check whether user u has permission p in its current $\mathcal{T}$ state); this type of restriction is handled in QP.

Finally, our reachability restrictions R define how these three functions relate, by allowing us to parameterize whether we require one-way reachability ($\mathcal{T}$ must be able to transition to states corresponding to all reachable $\mathcal{S}$ states) or bidirectional reachability ($\mathcal{T}$ also cannot transition to states that do not correspond to reachable $\mathcal{S}$ states).

The bare minimum set of these simulation properties that must be enforced for a mapping to be considered a simulation is a notion of state correspondence and a reachability relation. We present the definition of implementable expressiveness simulation, which refines the mapping by enforcing these properties.

Definition 5 *Given two access control systems, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$ and $\mathcal{T} = \langle \Gamma^{\mathcal{T}}, \Psi^{\mathcal{T}}, Q^{\mathcal{T}} \rangle$ and a mapping $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$ from $\mathcal{S}$ to $\mathcal{T}$, an* implementable expressiveness simulation *of $\mathcal{S}$ in $\mathcal{T}$ based on σ is defined as $\sigma' = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q, \sim, R \rangle$, where:*

- *$\sim \subseteq \Gamma^{\mathcal{S}} \times \Gamma^{\mathcal{T}}$ is the state correspondence, and $\forall \gamma \in \Gamma^{\mathcal{S}}, \gamma \sim \sigma_\Gamma(\gamma)$*
- *R is a reachability restriction*

We define all properties over the expressiveness simulation $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q, \sim, R \rangle$. Unless otherwise noted, properties within a dimension are totally ordered from most to least strict.

B. State correspondence properties

As discussed in Section III-B, the state correspondence of an implementable simulation of $\mathcal{S}$ in $\mathcal{T}$ is a function, $\sigma_\Gamma : \Gamma^{\mathcal{S}} \to \Gamma^{\mathcal{T}}$ mapping each state in $\mathcal{S}$ to a state in $\mathcal{T}$. There are several ways in which we can restrict this mapping.

Dimension SC: State correspondence structure

This dimension of properties restricts the way in which corresponding states are structurally similar. All properties within this dimension were inspired by state correspondence relations from prior expressiveness simulations; other application-specific state correspondence relations are conceivable.

SCs: Structure-correspondent
$\gamma^{\mathcal{S}} \overset{\mathsf{s}}{\sim} \gamma^{\mathcal{T}} \triangleq \forall S_i \in \left[\gamma^{\mathcal{S}}\right].(S_i \in \left[\gamma^{\mathcal{T}}\right])$

SCq: Query-correspondent
$\gamma^{\mathcal{S}} \overset{\mathsf{q}}{\sim} \gamma^{\mathcal{T}} \triangleq \forall q \in Q^{\mathcal{S}}.(\gamma^{\mathcal{S}} \vdash q \iff \sigma_Q(q, \gamma^{\mathcal{T}}) = \text{TRUE})$

SCa: Authorization-correspondent
$\gamma^{\mathcal{S}} \overset{\mathsf{a}}{\sim} \gamma^{\mathcal{T}} \triangleq \forall r \in \mathcal{R}^{\mathcal{S}}.(\gamma^{\mathcal{S}} \vdash r \iff \sigma_Q(r, \gamma^{\mathcal{T}}) = \text{TRUE})$

Authorization-correspondent simulations enforce that every $\gamma^{\mathcal{S}}$ maps to a $\gamma^{\mathcal{T}}$ that agrees on all authorization requests: any permission granted/denied in $\gamma^{\mathcal{S}}$ must also be granted/denied in $\gamma^{\mathcal{T}}$. Requests that exist in $\mathcal{T}$ but not in $\mathcal{S}$ are not restricted. This type of correspondence is used in [2], [3], [7]–[10]. *Query-correspondence* requires that $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ agree on *all* queries, not just authorization requests. This type of correspondence is used in the expressiveness simulations of [13], [14].

Finally, *structure-correspondent* simulations require all corresponding state elements to be identical. If $\gamma^{\mathcal{S}}$ structure-corresponds to $\gamma^{\mathcal{T}}$, then every set in $\gamma^{\mathcal{S}}$ exists in $\gamma^{\mathcal{T}}$, and contains all the same elements ($\gamma^{\mathcal{T}}$ may contain additional sets or relations). Thus, if $\gamma^{\mathcal{S}}$ contains sets of users and permissions, and a relation between them (a subset of users × permissions) specifying accesses, $\gamma^{\mathcal{T}}$ must contain identical sets of users and permissions, and an identical set of $\langle \text{user}, \text{permission} \rangle$ pairs. This notion of state correspondence is used in [6].

The type of state correspondence used is a central characteristic of a type of simulation. Enforcing a state correspondence that is too weak can allow the simulating system to diverge from the simulated system in unexpected ways, while a state correspondence that is too strong will cause the simulating system to track the simulated system more closely than necessary (e.g., by constraining the values of queries that the deployment never needs to ask). Thus, choosing a particular state correspondence is choosing how closely the simulating system must stay to the simulated system.

Dimension SS: State storage

An orthogonal class of restrictions that can be placed on the state correspondence relation involve its allowed storage. Here, we restrict the size of $\gamma^{\mathcal{T}} = \sigma_\Gamma(\gamma^{\mathcal{S}})$ with respect to $\gamma^{\mathcal{S}}$.

SSl: Linear storage
$\exists c \in \mathbb{R}^+, s \in \mathbb{Z}^+ : \forall \gamma \in \Gamma^{\mathcal{S}} : |\gamma| \geq s \Rightarrow |\sigma_\Gamma(\gamma)| \leq c|\gamma|$

SSp: Polynomial storage
$\exists k \in \mathbb{R}^+, s \in \mathbb{Z}^+ : \forall \gamma \in \Gamma^{\mathcal{S}} : |\gamma| \geq s \Rightarrow |\sigma_\Gamma(\gamma)| \leq |\gamma|^k$

SS∞: Unbounded storage *No restriction.*

A *linear storage* simulation says that $\gamma^{\mathcal{T}}$ can grow at most linearly with $\gamma^{\mathcal{S}}$, while in a *polynomial storage* simulation, the size of $\gamma^{\mathcal{T}}$ is bounded by a polynomial in the size of $\gamma^{\mathcal{S}}$. The most obvious result of enforcing properties within SS is limited trusted storage, but it can also limit iteration over the resulting state (e.g., if an action must be taken for each document in the simulating system, SSI ensures that this sequence of actions is linear in the size of the simulated state).

C. Command mapping properties

Recall that the command mapping for an implementable simulation (Definition 5) is a function $\sigma_\Psi : \Psi^{\mathcal{S}} \times \Gamma^{\mathcal{T}} \to (\Psi^{\mathcal{T}})^*$ that returns the sequence of $\mathcal{T}$ commands needed to simulate an $\mathcal{S}$ command starting from a particular $\mathcal{T}$ state. Thus, it allows us to simulate $\mathcal{S}$ commands in an active simulation using $\mathcal{T}$. We now discuss the ways in which we can restrict this mapping.

Dimension CD: Command mapping dependence

While Definition 5 maps each $\mathcal{S}$ command and $\mathcal{T}$ state to a sequence of $\mathcal{T}$ commands, some previous works use more strict command mappings, mapping each $\mathcal{S}$ command to a sequence of $\mathcal{T}$ commands without considering the state [10]. In between these options, we may map each $\mathcal{S}$ command and $\mathcal{T}$ *theory*, calculating the sequence of $\mathcal{T}$ commands by observing only the queriable portions of the $\mathcal{T}$ state. *Command mapping dependence* thus restricts the information that the command mapping can consider about a $\mathcal{T}$ state when calculating the trace of $\mathcal{T}$ commands to execute.

CDi: Independent command mapping $\exists \sigma' : \Psi^{\mathcal{S}} \to (\Psi^{\mathcal{T}})^*.(\sigma_\Psi(\psi, \gamma) \equiv \sigma'(\psi))$

CDt: Theory-dependent command mapping $\exists \sigma' : \Psi^{\mathcal{S}} \times \mathit{Th}(\mathcal{T}) \to (\Psi^{\mathcal{T}})^*.(\sigma_\Psi(\psi, \gamma) \equiv \sigma'(\psi, \mathit{Th}(\gamma)))$

CDs: State-dependent command mapping *No restriction.*

With *independent command* simulations, $\mathcal{S}$ commands must be precompiled to $\mathcal{T}$ commands which will work in any reachable $\mathcal{T}$ state. This is a restriction placed by [10]. *Theory dependent* command mappings allow limited inspection of the $\mathcal{T}$ state; this restriction allows the sequence of $\mathcal{T}$ commands to be determined based only on the *theory* of the $\mathcal{T}$ state: the values of all $\mathcal{T}$ queries in the state. If two $\mathcal{T}$ states answer all queries the same way, the same $\mathcal{T}$ commands would be used in both to simulate an $\mathcal{S}$ command. With this restriction, the monitor that transforms $\mathcal{S}$ inputs into $\mathcal{T}$ procedures need not be more privileged than users of the access control system, since queries are the user's only API to observe the state.

Finally, *state-dependent* command mappings can arbitrarily observe the state. This requires a monitor that is privileged enough to observe elements of the state that are not queriable, and two states that answer all queries identically may simulate commands differently depending on unobservable state.

Dimension CC: Command mapping complexity

Having considered the inputs available to the command mapping, we now consider the time complexity of this mapping. Note that this is measured as the increase in time as the *state* grows and thus is meaningless for independent command.

CCc: Constant command mapping $\forall \psi \in \Psi^{\mathcal{S}}$, *the algorithm for* $\sigma_\psi(\gamma) = \sigma_\Psi(\psi, \gamma)$ *has time complexity* $T(n) \in \mathcal{O}(1)$

CCl: Linear command mapping $\forall \psi \in \Psi^{\mathcal{S}}$, *the algorithm for* $\sigma_\psi(\gamma) = \sigma_\Psi(\psi, \gamma)$ *has time complexity* $T(n) \in \mathcal{O}(n)$

CC∞: Unbounded command mapping *No restriction.*

Constant command simulations do not allow more processing time for bigger states. Thus, the command mapping cannot loop over sets within the state. With *linear command*, the command mapping can take time linear in the size of the state, e.g., looping over sets in the state, but cannot contain double loops over sets, sort sets, etc. Finally, *unbounded command* simulations put no limit on the complexity of the command mapping (though we may expect it to have to be tractable, e.g., poly-time).

Dimension CS: Command mapping stuttering

Since the command mapping maps an $\mathcal{S}$ state to a sequence of $\mathcal{T}$ states, we may restrict the number of commands that can be used to simulate a single $\mathcal{S}$ command.

CS1: Lock-step $\forall \psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}} : |\sigma_\Psi(\psi, \gamma)| \leq 1$

CSc: Constant step $\exists c : \forall \psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}} : |\sigma_\Psi(\psi, \gamma)| \leq c$

CS∞: Unbounded step *No restriction.*

A *lock-step* simulation allows at most one $\mathcal{T}$ command for each simulated $\mathcal{S}$ command. This mitigates concurrency issues for multiuser systems, since the system does not pass through potentially inconsistent states between command executions. *Constant step* simulations allow multiple commands to be used, but only a number constant in the size of the state. Thus, multiple actions can be taken, but not, e.g., a command for each user in the system. Finally, *unbounded step* does not restrict how many $\mathcal{T}$ commands can be executed per $\mathcal{S}$ command.

Dimension CT: Trace structure

This class of properties enforces structural constraints on the traces of commands returned by the command mapping. This can address the potentially inconsistent states between start and end states in traces generated by the command mapping. Here, we present several examples of trace restrictions, using the notation $terminal(\gamma, \psi_1, \cdots, \psi_j)$ to denote the end state resulting from executing the sequence of commands $\psi_1, \cdots, \psi_j$, starting from the state γ. Note that this dimension of properties is not totally ordered.

CT1: Semantic lock-step

$$\begin{aligned}
&\forall \psi \in \Psi^{\mathcal{S}}, \gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\
&\quad \exists \overline{\psi} = \langle \psi_1, \psi_2, \ldots, \psi_m \rangle \in (\Psi^{\mathcal{T}})^*, i \in (1, m].(\\
&\quad\quad \sigma_\Psi(\psi, \gamma^{\mathcal{T}}) = \overline{\psi} \wedge \\
&\quad\quad \forall j \in [1, i).(\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}} \Rightarrow \\
&\quad\quad\quad \gamma^{\mathcal{S}} \sim terminal(\gamma^{\mathcal{T}}, \psi_1 \cdots \psi_j)) \wedge \\
&\quad\quad \forall j \in [i, m].(\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}} \Rightarrow \\
&\quad\quad\quad \mathit{next}(\gamma^{\mathcal{S}}, \psi) \sim terminal(\gamma^{\mathcal{T}}, \psi_1 \cdots \psi_j))))
\end{aligned}$$

First, a *semantic lock-step* simulation can appear to be lock-step (i.e., it does not enter any inconsistent states), because even though it is allowed to execute multiple $\mathcal{T}$ commands to simulate a single $\mathcal{S}$ command, only one of those commands

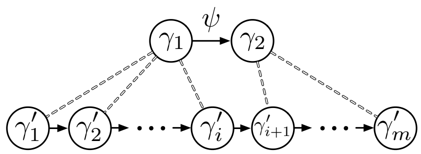


Fig. 3: A graphical representation of semantic lock-step

is allowed to make correspondence-related changes. That is, consider the sequence of $\mathcal{T}$ states constructed by executing the sequence of commands $\sigma_\Psi(\psi^{\mathcal{S}}, \gamma^{\mathcal{T}})$. In semantic lock-step, all of these states must correspond to the either the start state in $\mathcal{S}$ or the end state in $\mathcal{S}$, and once the transition from start state to end state is made, the remaining states must all be equivalent to the end state. Thus, from the point of view of a user who can ask any combination of queries, the simulation appears to be lock-step. This restriction is depicted in Fig. 3.

CTq: Query monotonic

$\forall\psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, q \in Q^{\mathcal{T}}.monotonic(\psi, \gamma, q)$, *where:*

$$\begin{aligned}
&monotonic(\psi, \gamma, q) \triangleq \exists\overline{\psi} = \langle\psi_1, \psi_2, \ldots, \psi_m\rangle \in (\Psi^{\mathcal{T}})^*.(\\
&\quad \sigma_\Psi(\psi, \gamma) = \overline{\psi} \wedge \\
&\quad \forall i \in (1, m).(\\
&\qquad terminal(\gamma, \psi_1 \cdots \psi_i) \vdash q \Rightarrow \\
&\qquad\quad (terminal(\gamma, \psi_1 \cdots \psi_{i-1}) \vdash q \vee terminal(\gamma, \psi_1 \cdots \psi_m) \vdash q) \wedge \\
&\qquad terminal(\gamma, \psi_1 \cdots \psi_i) \nvdash q \Rightarrow \\
&\qquad\quad (terminal(\gamma, \psi_1 \cdots \psi_{i-1}) \nvdash q \vee terminal(\gamma, \psi_1 \cdots \psi_m) \nvdash q)))
\end{aligned}$$

Consider the start and end states of a trace in $\mathcal{T}$, γ and γ', respectively. Let Q^+ be the set of queries that become true in γ' that were false in γ, and Q^- be the set of queries that become false in γ' that were true in γ. During the trace from γ to γ', *query monotonicity* enforces that no queries are made true except Q^+, and no queries are made false except Q^-. Thus, from the point of view of a user who can ask only single queries, the simulation appears to be lock-step.

CTa: Access monotonic

$\forall\psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, r \in \mathcal{R}^{\mathcal{T}}.monotonic(\psi, \gamma, r)$

Access monotonicity is similar to query monotonicity but considering only authorization requests. Let $\mathcal{R}^+$ be the set of requests that become allowed in γ' that were denied in γ, and $\mathcal{R}^-$ be the set of requests that become denied in γ' that were allowed in γ. During the trace from γ to γ', access monotonicity enforces that no requests are granted except $\mathcal{R}^+$, and no requests are revoked except $\mathcal{R}^-$.

CTs: Non-contaminating

$$\begin{aligned}
&\forall\psi \in \Psi^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\
&\quad \exists\overline{\psi} = \langle\psi_1, \psi_2, \ldots, \psi_m\rangle \in (\Psi^{\mathcal{T}})^*.(\\
&\qquad \sigma_\Psi(\psi, \gamma^{\mathcal{T}}) = \overline{\psi} \wedge \\
&\qquad \forall\gamma_i^{\mathcal{T}} \in \left\{\gamma_i^{\mathcal{T}} \mid \exists\psi_i \in \overline{\psi} : \gamma_i^{\mathcal{T}} = terminal(\gamma^{\mathcal{T}}, \psi_1 \cdots \psi_i)\right\}.(\\
&\qquad\quad Allowed(\gamma_i^{\mathcal{T}}) \subseteq Allowed(\gamma^{\mathcal{T}}) \vee \\
&\qquad\quad Allowed(\gamma_i^{\mathcal{T}}) \subseteq Allowed(terminal(\gamma^{\mathcal{T}}, \overline{\psi})))))
\end{aligned}$$

The *non-contaminating* trace property ensures that no two accesses are allowed in the same state that are not both allowed in either the start or end state. This prevents, e.g., an intermediate state where a file can be accessed by two users simultaneously when simulating a command intended to *switch* which user can access the file. This definition uses the $Allowed(\gamma)$ notation, indicating the set of all permissions p allowed in state γ (i.e., such that $\gamma \vdash p$).

Dimension CA: Actor preservation

Actor preservation properties restrict which users can be invoked in $\mathcal{T}$ to handle $\mathcal{S}$ commands. Here, we assume that $\alpha(\psi)$ denotes the actor executing the command ψ. Note that this requires system support (e.g., the executing actor being an implicit argument passed to a command) in order for a simulation to be executable.

CA⊤: Self-execution $\forall\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, \forall\psi^{\mathcal{T}} \in \sigma_\Psi(\psi^{\mathcal{S}}, \gamma), \alpha(\psi^{\mathcal{S}}) = \alpha(\psi^{\mathcal{T}})$

CAa: Administration-preservation *Let A be the administrative subset of executing entities in the system.* $\forall\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, \forall\psi^{\mathcal{T}} \in \sigma_\Psi(\psi^{\mathcal{S}}, \gamma), \alpha(\psi^{\mathcal{T}}) \in A \Rightarrow \alpha(\psi^{\mathcal{S}}) \in A$

Self-execution says that any command in $\mathcal{S}$ executed by any user u must be mapped to a sequence of commands in $\mathcal{T}$, all of which are executed by u. *Administration-preservation* prevents the invocation of administrators in $\mathcal{T}$ where they were not needed in $\mathcal{S}$. In an administration-preserving simulation, any command in $\mathcal{S}$ executed by a non-administrative user is mapped to a sequence of commands in $\mathcal{T}$, none of which is executed by an administrator. Other forms of actor preservation, as well as defining the set of administrators, are application-specific.

D. Query decider properties

We defer the bulk of the technical discussion of the query decider restrictions to the technical report accompanying this paper [27], as they are largely similar to the command mapping restrictions. Query decider dependence (QD), like command mapping dependence (CD), restricts the information that the query decider can consider about a $\mathcal{T}$ state when deciding the truth value of an $\mathcal{S}$ query in that state. Query decider complexity (QC) restricts the runtime of the routine.

Query preservation (QP) indicates which queries need to stay the same as they are mapped from system $\mathcal{S}$ to system $\mathcal{T}$. A particular application may require any given set of queries to be preserved; the most common property in this dimension is *authorization preservation*, which enforces that the query decider maps each $\mathcal{S}$ request to the value of the identical request in the $\mathcal{T}$ state. This can be seen as ensuring that $\mathcal{T}$ is using its model "as intended" (i.e., forcing it to answer simulated requests as it would its own native requests).

E. Reachability

Dimension R: Reachability

The last dimension of properties we consider ties the mappings together to ensure the simulation is indeed what one could consider a simulation in the classic sense. A state correspondence, query decider, and command mapping do not automatically define a simulation without reachability constraints. Here, we define forward and bidirectional reachability, two variants of this type of constraint (note that these properties are presented in *increasing* strictness since the latter builds upon the former).

R$\rightarrow$: Forward reachability

$$\begin{aligned}&\forall \gamma_0^{\mathcal{S}}, \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\ &\quad \gamma_0^{\mathcal{S}} \sim \gamma_0^{\mathcal{T}} \wedge \gamma_0^{\mathcal{S}} \mapsto \gamma_1^{\mathcal{S}} \Rightarrow \exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\ &\qquad \gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}))\end{aligned}$$

R$\leftrightarrow$: Bidirectional reachability *Forward reachability, and:*

$$\begin{aligned}&\forall \gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\ &\quad \gamma_0^{\mathcal{S}} \sim \gamma_0^{\mathcal{T}} \wedge \gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}} \Rightarrow \exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(\\ &\qquad \gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}))\end{aligned}$$

In *forward reachability*, any transition made in $\mathcal{S}$ must be possible in $\mathcal{T}$. If $\gamma_0^{\mathcal{S}}$ corresponds to $\gamma_0^{\mathcal{T}}$, and $\gamma_1^{\mathcal{S}}$ can be reached from $\gamma_0^{\mathcal{S}}$ via the commands of $\mathcal{S}$, then $\gamma_1^{\mathcal{S}}$ must correspond to a state $\gamma_1^{\mathcal{T}}$ in $\mathcal{T}$ that is reachable from $\gamma_0^{\mathcal{T}}$. The notion of state correspondence is determined by the property chosen in dimension SC.

Bidirectional reachability (or bi-reachability), also requries that $\mathcal{T}$ cannot enter a state that does not correspond to a reachable state in $\mathcal{S}$. If $\gamma_0^{\mathcal{S}}$ corresponds to $\gamma_0^{\mathcal{T}}$, and $\gamma_1^{\mathcal{T}}$ is reachable from $\gamma_0^{\mathcal{T}}$ by executing a command, then there must exist an $\mathcal{S}$ state $\gamma_1^{\mathcal{S}}$ that corresponds to $\gamma_1^{\mathcal{T}}$ and that is reachable from $\gamma_0^{\mathcal{S}}$ by executing one or more commands. This process may make use of multiple steps, since the procedure for finding the corresponding $\mathcal{S}$ states does not need to be constructed, these states must simply exist. The operational advantage of enforcing R$\leftrightarrow$ is that, even if the simulating system's native operations are exposed to users, the system can never enter a state that does not have an equivalent in the simulated system.

V. Positioning Existing Simulations

As mentioned in Section IV-A, no set of properties can be proven to describe all conceivable simulations. In this section, we support the set of properties defined in this work by showing that it can *precisely* describe the wide range of existing expressiveness simulations.

A. Expressiveness using Simulation Properties

We will now draw the formal distinction between a simulation and expressiveness. Here, we use $\mathcal{T} \underset{\mathcal{X}}{sim} \mathcal{S}$ to denote, "$\mathcal{T}$ can admit a simulation of type $\mathcal{X}$ of $\mathcal{S}$," and $\mathcal{S} \leq^{\mathcal{X}} \mathcal{T}$ to denote, "$\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to simulations of type $\mathcal{X}$."

While previous work considers the expressiveness result to be equivalent to a simulation (i.e., $\mathcal{T} \underset{\mathcal{X}}{sim} \mathcal{S} \equiv \mathcal{S} \leq^{\mathcal{X}} \mathcal{T}$), expressiveness in a practical sense is subject to a subtle distinction. Since we mean for expressiveness to be *implementable* (i.e., if $\mathcal{T}$ is as expressive as $\mathcal{S}$, then $\mathcal{T}$ can be used in place of $\mathcal{S}$), expressiveness within the domain of simulation properties should mean the following: if $\mathcal{T}$ is as expressive as $\mathcal{S}$, then $\mathcal{T}$ can simulate any system that $\mathcal{S}$ can simulate. Thus, we define *expressiveness* in the context of a set of simulation properties.

Definition 6 (Expressiveness) *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and a set of simulation properties $\mathcal{P}$, we say that $\mathcal{T}$ is* at least as expressive *as $\mathcal{S}$ with respect to $\mathcal{P}$ (denoted $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$) to mean that, for every system $\mathcal{U}$, if $\mathcal{S}$ can simulate $\mathcal{U}$ while enforcing $\mathcal{P}$, then $\mathcal{T}$ can simulate $\mathcal{U}$ while enforcing $\mathcal{P}$ ($\forall \mathcal{U} : \mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{sim} \mathcal{U}$).* $\diamondsuit$

We first point out that this definition of expressiveness is strictly more general than the more traditional (often implied) notion. Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \leq^{\mathcal{X}} \mathcal{T}$ implies $\mathcal{T} \underset{\mathcal{X}}{sim} \mathcal{S}$. The additional generalization can be viewed from a formal standpoint as dropping the (incorrect) assumption that all types of simulation are transitive (i.e., that $\mathcal{T}\ sim\ \mathcal{S}$ and $\mathcal{S}\ sim\ \mathcal{U}$ imply $\mathcal{T}\ sim\ \mathcal{U}$). For instance, assume that $\mathcal{T}$ can simulate $\mathcal{S}$ and $\mathcal{S}$ can simulate $\mathcal{U}$, each with a quadratic increase in state storage. While $\mathcal{T}$ may be able to simulate $\mathcal{U}$, this simulation may require greater than quadratic storage.

From a more intuitive standpoint, we point out that, except in the case of custom-built access control solutions, any deployment is a simulation of a workload (i.e., ideal operation) using an existing system. That is, unless $\mathcal{S}$ is custom-made to exactly satisfy the desired workload, replacing it with $\mathcal{T}$ is not a matter of whether $\mathcal{T}$ can simulate $\mathcal{S}$, but whether $\mathcal{T}$ can admit an equally good simulation of the (perhaps not formally specified) workload that $\mathcal{S}$ is known to simulate. This concept is discussed by Kane and Browne [28], who point out that an access control implementation is often only an approximation of the desired policy. In particular, as policy languages get more complex, deployments often make use of approximations that are easier to analyze and more efficient to enforce than the overly-expressive policy language.

B. Decomposing Expressiveness Simulations to Properties

In order to use the set of expressiveness simulation properties detailed in Section IV to systematically compare previously proposed notions of simulation, we present our formal way of stating that a notion of simulation and a set of simulation properties are equivalent. We call this correspondence *simulation decomposition*: when a notion of simulation $\mathcal{X}$ can be *decomposed to* a set of simulation properties $\mathcal{P}$, then analyses using $\mathcal{X}$ and $\mathcal{P}$ yield equivalent expressiveness results.

Definition 7 (Simulation Decomposition) *Given a notion of access control simulation $\mathcal{X}$ and a set of simulation properties $\mathcal{P}$, $\mathcal{X}$ can be* decomposed *to $\mathcal{P}$ (denoted $\mathcal{X} \doteq \mathcal{P}$) if and only if, for all systems $\mathcal{S}$ and $\mathcal{T}$, $\mathcal{T} \underset{\mathcal{X}}{sim} \mathcal{S} \iff \mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. That is, $\mathcal{T}$ admits an $\mathcal{X}$ simulation of $\mathcal{S}$ if and only if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$.* $\diamondsuit$

Recall from Definition 6 that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ says that any system that can be simulated by $\mathcal{S}$ while preserving properties $\mathcal{P}$ can can also simulated by $\mathcal{T}$ while preserving $\mathcal{P}$. In light of this, we will position an existing notion of simulation, $\mathcal{X}$, within the lattice formed by our simulation properties (i.e., prove $\mathcal{X} \doteq \mathcal{P}$) by proving the following for the set of properties $\mathcal{P}$:

1) (*Only-if direction*) $\mathcal{T} \underset{\mathcal{X}}{sim} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{sim} \mathcal{U}$

2) (*If direction*) $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{\mathcal{X}}{sim} \mathcal{S}$

We give an example of such a proof in the following section.

C. Example Decomposition

To demonstrate how simulation decomposition proofs are written, we now consider the Ammann-Lipton-Sandhu simulation [6]. The ALS simulation considers access control states as graphs: sets of primitive objects are node types, and sets of relations are edge types. The set of node types and edge types in the states of system $\mathcal{S}$ are denoted $NT(\mathcal{S})$ and $ET(\mathcal{S})$, respectively. The ALS state correspondence is then defined as follows (reworded slightly from [6]).

Definition 8 *A state in system $\mathcal{S}$, a simulated system, and a state in system $\mathcal{T}$, a simulating system,* correspond *iff the graph defining the state in $\mathcal{S}$ is identical to the subgraph obtained by taking the state in $\mathcal{T}$ and discarding all nodes (edges) not in $NT(\mathcal{S})$ $(ET(\mathcal{S}))$.* ◊

The ALS simulation is defined with respect to this state correspondence.

Definition 9 *Under the definition of correspondence in Definition 8, system $\mathcal{T}$* simulates *system $\mathcal{S}$ iff the following conditions hold:*

1) *If system $\mathcal{S}$ can reach a given state, system $\mathcal{T}$ can reach a corresponding state.*
2) *If system $\mathcal{T}$ can reach a given state, system $\mathcal{S}$ can reach a corresponding state.*

We will now demonstrate the two-step simulation decomposition proof technique described in Section V-B for the ALS simulation. For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$. Recall that SCs is structure state correspondence, which says that the simulating state must include all of the unaltered sets from the simulated state; QPa is authorization preservation, which says that each authorization request must be mapped identically from simulated to simulating system (and thus the simulating system must support the same set of requests as the simulated system); and R↔ is bireachability, which says that the simulating system can reach a state which corresponds to each reachable simulated state, and cannot reach a state which does not correspond to a reachable state in the simulated system.

We will demonstrate the two steps of the proof technique by proving two requesite lemmas. First, step 1 (only-if direction):

Lemma 1 *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{ALS}{sim} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{sim} \mathcal{U}$$

That is, if $\mathcal{T}$ admits an ALS simulation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$.

Proof *To prove this lemma, we let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be access control systems such that $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U}$ but are otherwise arbitrary, and we show that $\mathcal{T} \underset{\mathcal{P}}{sim} \mathcal{U}$.*

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let next$(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(terminal(\gamma_0^{\mathcal{S}}, \sigma_\Psi(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}})) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{s}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, there exists a sequence of $\mathcal{T}$ commands $\Psi_0^{\mathcal{T}}$ such that $terminal(\gamma_0^{\mathcal{T}}, \Psi_0^{\mathcal{T}}) = \gamma_1^{\mathcal{T}}$. Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \rightarrow (\Psi^{\mathcal{T}})^*$ such that it returns $\Psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that next$(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(terminal(\gamma_0^{\mathcal{T}}, \sigma_\Psi(\psi, \gamma_0^{\mathcal{T}})) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCs},\mathsf{R}\rightarrow\}}{sim} \mathcal{U}$. Next, we show QPa.

Choose some arbitrary request $r_0^{\mathcal{U}} \in \mathcal{R}^{\mathcal{U}}$ and state $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$. Since $\mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U}$,

$$\forall r^{\mathcal{U}} \in \mathcal{R}^{\mathcal{U}}, \gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \sigma_Q(r^{\mathcal{U}}, \gamma^{\mathcal{S}}) = \gamma^{\mathcal{S}} \vdash r^{\mathcal{U}}$$

Thus, we know that $\mathcal{S}$ supports all $\mathcal{U}$ requests, and corresponding $\mathcal{S}$ and $\mathcal{U}$ states will answer $\mathcal{U}$ requests identically. Therefore, $r_0^{\mathcal{U}} \in \mathcal{R}^{\mathcal{S}}$. Since $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$,

$$\forall r^{\mathcal{S}} \in \mathcal{R}^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \sigma_Q(r^{\mathcal{S}}, \gamma^{\mathcal{T}}) = \gamma^{\mathcal{T}} \vdash r^{\mathcal{S}}$$

Thus, $\sigma_Q(r_0^{\mathcal{U}}, \gamma^{\mathcal{T}}) = \gamma^{\mathcal{T}} \vdash r_0^{\mathcal{U}}$.

Hence, $\mathcal{T} \underset{\{\mathsf{SCs},\mathsf{QPa},\mathsf{R}\rightarrow\}}{sim} \mathcal{U}$. Next, we show R↔.

Choose some arbitrary states $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{S}}.(\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Let $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{s}{\sim} \gamma_1^{\mathcal{S}})$$

Thus, given $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$ and $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\mathcal{P}}{sim} \mathcal{U}$. □

Next, we demonstrate step 2 (if direction):

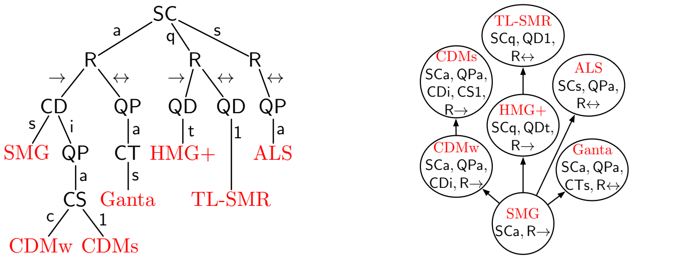
Lemma 2 *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{ALS}{sim} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits an ALS simulation of $\mathcal{S}$.*

Proof *To prove this lemma, we let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, and we show that $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$.*

Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{sim} \mathcal{U}$.

Simulation	Decomposition
ALS	SCs QPa R↔
CDMw	SCa QPa CDi R→
CDMs	SCa QPa CDi CS1 R→
Ganta	SCa QPa CTs R↔
HMG+	SCq QDt R→
HMG+a	QPa
HMG+s	CTa
HMG+p	CAa
SMG	SCa R→
TL-SMR	SCq QD1 R↔

(a) Decompositions of surveyed simulations



(b) Taxonomy of simulations

(c) Partial lattice of simulations

Fig. 4

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{sim} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{sim} \mathcal{S}$.

Thus, given $\gamma_0^{\mathcal{S}}, \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, by forward reachability, if $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{S}} \mapsto \gamma_1^{\mathcal{S}}$, then

$$\exists \gamma_1^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Since SCs *and* QPa *satisfy the ALS definition of state correspondence, this means we have satisfied the first property of the ALS simulation.*

1) *If $\mathcal{S}$ can reach a given state, $\mathcal{T}$ can reach a corresponding state.*

And by bidirectional reachability, given $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, if $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$, then

$$\exists \gamma_1^{\mathcal{S}}.(\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

And therefore, we have satisfied the second property of the ALS simulation:

2) *If $\mathcal{T}$ can reach a given state, $\mathcal{S}$ can reach a corresponding state.*

These properties satisfy the definition for ALS simulation, and hence $\mathcal{T}$ admits an ALS simulation of $\mathcal{S}$ ($\mathcal{T} \underset{ALS}{sim} \mathcal{S}$). □

Therefore, we have proved the decomposition of the ALS simulation:

Theorem 3 *ALS $\doteq$ {SCs, QPa, R↔}; that is, the ALS simulation decomposes to structure correspondence, authorization preservation, and bidirectional reachability.*

Proof *By Lemma 1, if $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 2, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{ALS}{sim} \mathcal{S}$, and thus the ALS simulation decomposes to* {SCs, QPa, R↔}. □

In the interest of space, all other decomposition proofs can be found in the technical report accompanying this paper [27].

D. Results

Now, we present the results of decomposing the simulations from the series of previous works discussed in Section II into sets of simulation properties from Section IV. First, a chart of our results is shown in Fig. 4a, which states the decomposition of the SMG simulation [2]–[4], [7]–[9], the Ganta simulation [5], the ALS simulation [1], [6], the CDM weak and strong simulations [10], the TL state-matching reduction [12], [13], and HMG+ parameterized expressiveness (along with several parameterized expressiveness properties) [14]. Properties are omitted if they are not explicitly required by the simulation's definition but are implied by other, explicit properties (e.g., CDMs decomposes to a set including CDi, which also implies CCc). Section VI-A discusses which properties imply others.

In Fig. 4b, we arrange this data as a taxonomy, with each split representing a dimension, with weaker properties positioned to the left and stronger properties to the right. We split first on the state correspondence, which is perhaps the biggest difference among the surveyed simulations. This separates simulations that preserve only the answers to authorization requests (SCa) from those that preserve all queries (SCq) and those that preserve full state structure (SCs). We note that the ALS simulation is alone in its decomposition including SCs; all other surveyed simulations allowed the simulating system to store information in a different organization than the simulated system, so long as the required queriable information (requests or queries) can be recovered. We also note that the predominant difference between the SMG simulation and the CDM simulations is the command dependence: in SMG, a command can be mapped completely differently if it is to be executed in different states, while in CDM, each command must be mapped without knowing the state in which it will be executed. The Ganta simulation is unique in enforcing the non-contamination trace restriction. HMG+ and TL-SMR use the same state correspondence, but HMG+ enforces a more lax query dependence and does not require bireachability. Simulations that are positioned farther apart are the most dissimilar. Most starkly different are SMG and ALS, positioned far left and far right, which share no simulation properties except in dimensions in which both enforce only minimum properties, despite their similar publication times.

In Fig. 4c, we position the surveyed simulations within a lattice. Higher simulations decompose to more strict properties, and an arrow from simulation $\mathcal{X}$ to simulation $\mathcal{X}'$ indicates that $\mathcal{X}'$ decomposes to strictly stronger properties than $\mathcal{X}$. Here we can see that the SMG simulation is strictly weakest, which supports previous claims to this effect [5], [13]. Several orthogonal directions were taken in defining other simulations

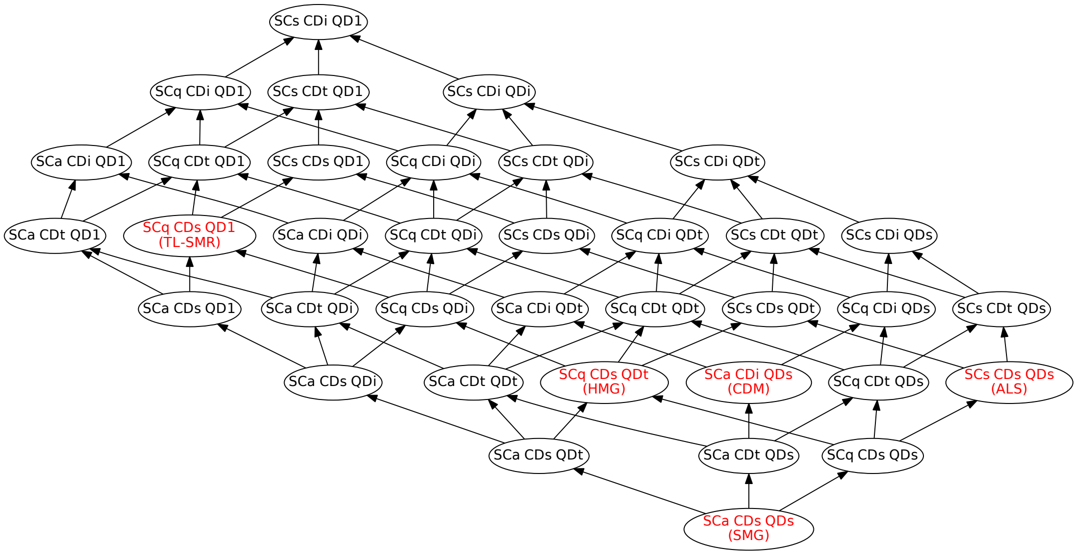


Fig. 5: Lattice of state correspondence, command dependence, and query dependence with positioned surveyed simulations

to enforce stronger properties. The CDM simulations, as noted above, restrict the command dependence. The Ganta simulation requires non-contamination and bireachability. The TL state-matching reduction and HMG+ parameterized expressiveness consider queries, and thus strengthen the state correspondence. The ALS simulation enforces an even more strict state correspondence, requiring the structure of a simulated system's state to be preserved in the simulating system. Interestingly, we note that while all are stronger than SMG, most pairs are incomparable due to being stronger in orthogonal ways. In particular, while TL-SMR is considered to be a relatively strong notion of simulation, this is not substantiated by the lattice, which shows TL-SMR to be stronger than HMG+ and SMG, but incomparable to the CDM, ALS, and Ganta simulations.

Figure 5 presents a lattice of state correspondence, command dependence, and query dependence, with the surveyed simulations positioned within it (in this space, the Ganta simulation is at the same point as the SMG simulation). This figure makes evident the wide range of points between existing simulations that have not been explored. In this figure, we omit several dimensions for readability, namely reachability (which further separates Ganta, ALS, and TL-SMR from SMG, CDM and HMG+) and stuttering (which would break CDM into its weak and strong counterparts). Perhaps the most interesting points to explore within this lattice are those that exist between two surveyed simulations. For example, $\{\mathsf{SCq}, \mathsf{CDs}, \mathsf{QDs}\}$ adds to SMG the preservation of queries beyond requests, but stops short of HMG+ by not restricting the query decider to consider only the theory of the state while mapping queries. Similarly, $\{\mathsf{SCa}, \mathsf{CDt}, \mathsf{QDs}\}$ takes away some of SMG's freedom to inspect the state mapping commands, but rather than go all the way to the independent command mapping of CDM, it still allows it to inspect the state's responses to queries. We also point out $\{\mathsf{SCq}, \mathsf{CDs}, \mathsf{QDi}\}$, which differs from HMG+ by enforcing query decider independence (mapping queries cannot consider the state or theory), but can map each simulated query to a boolean expression over simulating queries.

VI. Selecting New Sets of Properties

In Section V, we positioned the simulations used in previous works within a comparative lattice, allowing them to be formally compared for the first time. In this section, we enable a second use of our lattice of expressiveness simulation properties: crafting new notions of expressiveness by choosing the properties that most closely correspond to the scenario in which an access control system will be deployed. We first discuss interactions between dimensions; this discussion should act as a warning against choosing individual properties in isolation. We then interpret the impact each identified dimension has on the simulation, and identify properties of a deployment scenario that may dictate particular choices in each dimension. Finally, we discuss the potential impact these techniques could have on future expressiveness analysis.

A. Interactions Between Dimensions

We noted in Section V that some simulations decompose to sets of properties that include *implied* properties, or properties that are redundant given the others in the set. For instance, command independence (CDi) implies constant-time command mapping (CCc); if the command mapping does not depend on the state, then its procedure must be constant-time in the size of the state. Further, CCc implies constant step (CSc), since a constant-time procedure must have constant-size output.

An additional type of interaction is between basic properties and those properties whose definition relies on the basic properties in the abstract. For example, the definition of forward reachability (R$\rightarrow$) refers to sequences of commands output by σ^{Ψ}, the length of which may be limited by command mapping stuttering (CS). Further, the definitions of both reachability properties (R) and trace structure properties (CT) refer to corresponding states. Here, the details of what makes states correspond is left to the state correspondence structure (SC).

These dependencies show that the proof of a property in one dimension may rely on the properties chosen in another. Thus, e.g., changing to a stronger state correspondence requires re-proving a simulation's results for reachability and trace structure, since these are dependent on state correspondence.

Several property dimensions are defined over the size of the simulated state: command mapping complexity (CC), command mapping stuttering (CS), and query decider time-complexity (QC). Thus, these dimensions can be altered with respect to the original, simulated state by the state storage size (SS). For example, enforcing polynomial storage (SSp) and linear-time command mapping (CCl) will guarantee a command mapping that is linear-time with respect to the *simulating state*, which is a polynomial expansion over the original *simulated state*.

B. Interpreting the Dimensions

We now discuss the practical impacts each identified dimensions, and what types of environments may cause one to prefer a particular property in these dimensions over others.

SC: State correspondence structure allows one to change what needs to be preserved about the state during a simulation. If the deployment scenario in question assumes only that the simulation allows the proper authorization requests, SCa should suffice. For scenarios that require the access control system to support (and provide correct answers to) additional queries such as, *"Is user u a member of role r?"*, SCq is more appropriate. Finally, in scenarios that make use of additional code that has access to (and assumes a particular arrangement of) the access control system's internal data structures, SCs is the best choice.

SS: State storage limits the size of the simulated state with respect to the original state (i.e., the state of the system being simulated). This can be restricted for several reasons. The most obvious is storage space: if trusted storage for representing access control state is limited, we may restrict the simulation from mapping states in a way that increases storage by more than a linear factor (SSl) or a polynomial factor (SSp). However, the more interesting reason comes from an interaction described in Section VI-A. Since other dimensions place restrictions (e.g., on the number of commands executed) based on the size of the simulating state, we may restrict the state expansion to linear (SSl) in order, e.g., to restrict the command mapping procedure to be linear-time in the size of the *original, simulated state*. If state storage is polynomial (SSp), then even if we enforce a command mapping that is linear in the simulating state (CCl), this only restricts it to being polynomial-time with respect to the simulated state. Thus, even when trusted storage space is unbounded in the deployment scenario, one may desire to limit state size to limit later iteration over this state.

CD: Command mapping dependence allows one to require that the command mapping be computable without full knowledge and inspection of the state in which a command will be executed in. Independent command (CDi) requires that each command is mapped independent of the state, and is useful in deployment scenarios in which the agent calculating the simulating commands is completely unprivileged, and cannot inspect the state. It is also useful when commands must be precompiled, thus adding no computation at runtime beyond that of the simulating commands themselves. Theory-dependent command mapping (CDt) allows the command mapping to inspect the *theory* of the state (i.e., the answers to all queries). This property is useful in deployment scenarios in which the simulation agent is no more privileged than normal users—calculating the mapped commands requires only information available by asking queries. Finally, state-dependent command mapping (CDs) allows the command mapping to arbitrarily inspect the state, requiring a powerful simulation agent.

CC: Command mapping complexity restricts the time-complexity of the command mapping with respect to the size of the simulating state. Constant command mapping (CCc) can restrict the command mapping from taking any longer for larger states, and is thus appropriate when states can be large but mapping commands must always remain fast. Linear command mapping (CCl) prevents expensive nested loops over access control state as well as operations such as sorting, while still allowing more processing for larger states.

CS: Command stuttering restricts the number of simulating commands executed for each simulated command. Lock-step (CS1) simulations must execute no more than one simulating command per simulated command, and thus ensure there is no intermediate state exposed to users. In deployment scenarios without the ability to force atomic execution of a sequence of commands (or without built-in data structure locking), this property is crucial to preventing the inspection of intermediate (potentially inconsistent) states. Constant step (CSc) simulations are allowed a constant number of commands for each simulated command, and are thus appropriate when the state can grow to be large but the deployment scenario requires that the number of steps for any simulated action remain bounded (e.g., to prevent starvation due to locked structures).

CT: Trace structure properties restrict the path that the simulating system can take during the simulation of a single command. Semantic lock-step (CT1, depicted in Fig. 3) provides the benefits of a lock-step simulation in a slightly relaxed way: a "setup" phase prepares for the transition by changing only internal data (i.e., while remaining equivalent to the start state), then the transition occurs to a state equivalent to the end state, and then the "cleanup" phase cleans up any unnecessary leftover data (again, while remaining equivalent to the same end state). This is particularly useful when lock-step is too strict, but the deployment scenario is sensitive to the exposure of intermediate states (since, in CT1, no states are exposed except those equivalent to the start and end states).

Query monotonicity (CTq) ensures that no query changes its truth value except those that are required to change between the start and end state. This allows multiple steps, but ensures that intermediate states, while not corresponding with the start or end state, never answer any query in a way that neither the start nor end state would. This is useful in scenarios where intermediate states are undesirable, but users are not expected to execute more than a single query between "valid" states (and will thus never detect the inconsistency). Access monotonicity (CTa) is similar, but applies only to authorization requests, and is useful in scenarios where inconsistent states are not a danger as long as they do not wrongly allow or forbid a request. Finally, non-contamination (CTs) ensures that no two accesses are allowed in an intermediate state that are not *both* allowed in either the start or end state. Thus, the simulating system is restricted not only from allowing accesses forbidden in the simulated system, but also combinations of individually-allowed accesses that are never combined in the simulated system. This restriction is particularly useful in environments with operations that "swap" accesses from one subject or object to another, or where separation of privilege is utilized.

CA: Actor preservation restricts which users can be invoked to simulate commands. Self-execution (CA$\top$) requires each simulating command be executed by the same user as the original, simulated command. This allows the simulating agent to be completely unprivileged, mapping commands as a service to the user, but without executing them with any privilege beyond the user's own. Administration-preservation (CAa) requires any non-administrative simulated command be mapped to a sequence of non-administrative commands (i.e., a command that does not invoke administrative privileges cannot be simulated by an administrative command). This corresponds to scenarios in which users will be expected to operate largely without administrative intervention. No restriction in this dimension means that the command mapping can return commands to be executed by any other user. This is most appropriate when the simulating agent is trusted to execute administrative actions on behalf of untrusted users, or when the commands returned can then be delegated to other users to be approved and executed.

Finally, **R: reachability** specifies whether the simulating system should be restricted from entering a state that does not correspond to a simulated state. If the simulation agent is users' only interface to the deployed access control system, forward reachability (R$\rightarrow$) is sufficient. However, if users can access the simulating system's native commands, bireachability (R$\leftrightarrow$) ensures that the system cannot transition to a state that is inconsistent with the simulated system.

C. Studying Canonical Usages

Next, we use the above interpretation of our expressiveness simulation properties to guide a discussion about how each of the notions of simulation that we studied in Section V is used by its creators. In many cases, the definition for a particular notion of simulation is underconstrained, and the simulations written within the framework actually satisfy stronger properties than the defined lower bound. We refer to the set of properties that the authors seem to intend for a simulation to uphold as its *canonical usage*. In the case of Sandhu's simulation, the author recognizes that the given constructions are stronger than the definition, noting that formalizing the definition of the stronger simulation is beyond the scope of the work [2]. Here, we make conjectures regarding the decomposition of the canonical usage of these simulations. A lattice view of these conjectures is shown in Fig. 6, where $\overline{\mathcal{X}}$ indicates the canonical usage of simulation type $\mathcal{X}$. For example, $\overline{\text{SM}}$ refers to the form of the SMG simulation used in [7], [8].

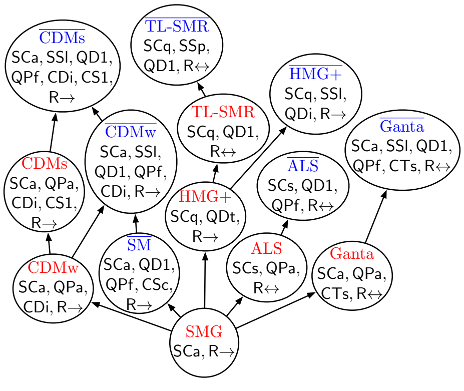


Fig. 6: Partial lattice of canonical usage

It is interesting to note that the relationships between notions of simulation are not necessarily preserved in the canonical usage. While SMG by definition is the weakest simulation, the canonical usage $\overline{\text{SM}}$ is incomparable to any simulation's definition and positioned strictly weaker than the canonical usage of the CDM simulations. While, by definition, the TL state-matching reduction is more strict than HMG+ parameterized expressiveness, their canonical usages are incomparable due to $\overline{\text{TL-SMR}}$ enforcing bireachability (R$\leftrightarrow$) and using polynomial state size (SSp), compared to $\overline{\text{HMG+}}$ enforcing forward reachability (R$\rightarrow$) and using linear state size (SSl). Finally, we note that all of CDMs, CDMw, SMG, and ALS simulations are canonically used in such a way that enforces full query preservation (QPf); that is, all of the constructed mappings of these types use the identity mapping for all supported queries, despite the fact that none of them specifically require this by definition. This trend of a notion of simulation's usage being consistently more strict than its definition reveals the difficulty in fully specifying the set of properties that a notion of expressiveness simulation is intended to enforce. The discussion in this section, aimed at helping analysts choose a reasonable set of properties for a deployment, can also help ensure that newer notions of simulation are fully specified, and best match their intended usages.

VII. Conclusion and Future Work

In this paper, we organize the existing knowledge of expressiveness simulations by formalizing a granular, property-based representation, proposing a wide range of dimensions of simulation properties, and positioning influential notions of expressiveness simulation from the literature within the lattice of these properties. In doing so, we provide the first systematic comparison of existing simulations that were not previously known to be directly comparable, showing how these notions of expressiveness simulation relate to one another.

Looking away from existing notions of simulation and rather *between* them, this work allows us to explore an organized space of simulations to identify areas to explore in future research. For instance, knowing expressiveness results derived using the SMG and ALS simulations, which of these hold true for notions of simulation "between" the two existing notions? What results can be shown for a simulation decomposing to the union of the properties of two existing notions? How far up the lattice do all systems become incomparable? These questions can only be explored thanks to the systematic means of simulation decomposition.

Finally, understanding the systems implications of various simulation properties will enable analysts to select the notion of access control expressiveness that corresponds most closely to the scenario in which they plan to deploy the target access control system(s). Thus, we make inroads toward bringing expressiveness analysis techniques out of the strictly formal realm, and repurpose these techniques to help select the most suitable access control system for a given application.

A question to be explored in future work is the identification of the set of analysis questions that a particular set of simulation properties preserve. For example, Tripunitara and Li showed that the state-matching reduction preserves *compositional security analysis instances*: the set of questions containing a single quantifier ($\exists$ or $\forall$), a propositional formula over queries φ, and a start state γ [13]. Semantically, the question asks whether φ it is {ever, always} true in states reachable from γ. If $\mathcal{T}$ admits a state-matching reduction of $\mathcal{S}$, then all compositional security analysis instances have the same value in $\mathcal{S}$ and $\mathcal{T}$. Identifying the types of analysis questions preserved by other notions of simulation would allow us even greater understanding of the practical and theoretical impacts of simulation property choices.

References

[1] P. Ammann, R. J. Lipton, and R. S. Sandhu, "The expressive power of multi-parent creation in a monotonic access control model," in *5th IEEE Computer Security Foundations Workshop - CSFW'92, Franconia, New Hampshire, USA, June 16-18, 1992, Proceedings*, Jun. 1992, pp. 148–156.

[2] R. S. Sandhu, "Expressive power of the schematic protection model," *Journal of Computer Security*, vol. 1, no. 1, pp. 59–98, 1992.

[3] R. S. Sandhu and S. Ganta, "On testing for absence of rights in access control models," in *CSFW*, 1993, pp. 109–118.

[4] ——, "On the expressive power of the unary transformation model," in *Third European Symposium on Research in Computer Security*, Nov. 1994, pp. 301–318.

[5] S. Ganta, "Expressive power of access control models based on propagation of rights," Ph.D. dissertation, George Mason University, 1996.

[6] P. Ammann, R. J. Lipton, and R. S. Sandhu, "The expressive power of multi-parent creation in monotonic access control models," *Journal of Computer Security*, vol. 4, no. 2/3, pp. 149–166, 1996.

[7] R. S. Sandhu and Q. Munawer, "How to do discretionary access control using roles," in *ACM Workshop on Role-Based Access Control*, 1998, pp. 47–54.

[8] Q. Munawer and R. S. Sandhu, "Simulation of the augmented typed access matrix model (ATAM) using roles," in *INFOSECU99*, 1999.

[9] S. L. Osborn, R. S. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 2, pp. 85–106, 2000.

[10] A. Chander, D. Dean, and J. C. Mitchell, "A state-transition model of trust management and access control," in *CSFW*, 2001, pp. 27–43.

[11] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, "A logical framework for reasoning about access control models," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 1, pp. 71–127, 2003.

[12] M. V. Tripunitara and N. Li, "Comparing the expressive power of access control models," in *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, Oct. 2004, pp. 62–71.

[13] ——, "A theory for comparing the expressive power of access control models," *Journal of Computer Security*, vol. 15, no. 2, pp. 231–272, 2007.

[14] T. L. Hinrichs, D. Martinoia, W. C. Garrison III, A. J. Lee, A. Panebianco, and L. Zuck, "Application-sensitive access control evaluation using parameterized expressiveness," in *Proceedings of the 26th IEEE Computer Security Foundations Symposium (CSF)*, Jun. 2013, pp. 145–160.

[15] W. C. Garrison III, Y. Qiao, and A. J. Lee, "On the suitability of dissemination-centric access control systems for group-centric sharing," in *Proceedings of the Fourth ACM Conference on Data and Application Security and Privacy*, Mar. 2014, pp. 1–12.

[16] W. C. Garrison III, A. J. Lee, and T. L. Hinrichs, "An actor-based, application-aware access control evaluation framework," in *19th ACM Symposium on Access Control Models and Technologies*, Jun. 2014, pp. 199–210.

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

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

[19] T. L. Hinrichs, W. C. Garrison III, A. J. Lee, S. Saunders, and J. C. Mitchell, "TBA : A hybrid of logic and extensional access control systems," in *Formal Aspects of Security and Trust, 8th International Workshop*, Sep. 2011, pp. 198–213.

[20] P. Naldurg and R. H. Campbell, "Dynamic access control: preserving safety and trust for network defense operations," in *SACMAT 2003, 8th ACM Symposium on Access Control Models and Technologies, June 2-3, 2003, Villa Gallia, Como, Italy, Proceedings*, 2003, pp. 231–237.

[21] T. Bourdier, H. Cirstea, M. Jaume, and H. Kirchner, "Formal specification and validation of security policies," in *Foundations and Practice of Security - 4th Canada-France MITACS Workshop, FPS 2011, Paris, France, May 12-13, 2011, Revised Selected Papers*, 2011, pp. 148–163.

[22] B. D. Payne, R. Sailer, R. Cáceres, R. Perez, and W. Lee, "A layered approach to simplified access control in virtualized systems," *Operating Systems Review*, vol. 41, no. 4, pp. 12–19, 2007.

[23] R. Krishnan, R. S. Sandhu, J. Niu, and W. H. Winsborough, "Foundations for group-centric secure information sharing models," in *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings*, 2009, pp. 115–124.

[24] V. C. Hu, D. A. Frincke, and D. F. Ferraiolo, "The policy machine for security policy management," in *Computational Science - ICCS 2001, International Conference, San Francisco, CA, USA, May 28-30, 2001. Proceedings, Part II*, May 2001, pp. 494–506.

[25] D. F. Ferraiolo, S. I. Gavrila, V. C. Hu, and D. R. Kuhn, "Composing and combining policies under the policy machine," in *SACMAT 2005, 10th ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden, June 1-3, 2005, Proceedings*, Jun. 2005, pp. 11–20.

[26] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn, *Assessment of Access Control Systems*. National Institute of Standards and Technology, 2006.

[27] W. C. Garrison III and A. J. Lee, "Decomposing, comparing, and synthesizing access control expressiveness simulations (extended version)," no. arXiv:1504.07948, Apr. 2015. [Online]. Available: http://arxiv.org/abs/1504.07948

[28] K. Kane and J. C. Browne, "On classifying access control implementations for distributed systems," in *SACMAT 2006,11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe, California, USA, June 7-9, 2006, Proceedings*, Jun. 2006, pp. 29–38.