Computers & Security 116 (2022) 102586

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

On the Expressive Power of Negated Conditions and Negative Authorizations in Access Control Models

Padmavathi Iyer, Amirreza Masoumzadeh*, Paliath Narendran

Department of Computer Science, University at Albany – SUNY, New York, USA

ARTICLE INFO

Article history:
Received 2 December 2020
Revised 24 September 2021
Accepted 20 December 2021
Available online 25 December 2021

Keywords:
access control policy
expressiveness
negation
negated condition
negative authorization

ABSTRACT

Access control policies specify which access requests should be allowed or denied in a system. Many access control policy models have used the concept of "negation" as part of their policy language, to enable fine-grained specification of authorizations. We identify two forms of this concept in the literature, namely, negated conditions and negative authorizations (deny rules). We argue that the choice of supporting negated conditions or negative authorizations can affect the expressive power of a policy model. Understanding their differences is crucial for designing an appropriate policy model for an intended application. However, no prior work has concretely analyzed them.

In this work, we formally analyze the expressive power of negated conditions and negative authorizations. We formulate two abstract policy models that support negated conditions and negative authorizations (including consideration of different meta-policies). Then, using a logic-based representation of policies, we prove the relative expressive power of those models in the context of a formal access control expressiveness analysis framework. The main result of our analysis is that models which support negated conditions are more expressive than models that support negative authorizations. That is, using negated conditions, we can represent all policies that can be expressed using negative authorizations. However, the converse is not true, i.e., negative authorizations cannot fully represent policies supporting negated conditions.

1. Introduction

Access control policies are the key security component of systems that determine what actions are authorized or not. There are several formal models of access control policies. Lower-level models such as access control lists (ACL) rely on verbose enumeration of authorizations. Higher-level models rely on concepts such as grouping, as in role-based access control (RBAC) (Ferraiolo and Kuhn, 1992; Sandhu et al., 1996), and conditional rules, as in attribute-based access control (ABAC) (Hu et al., 2013; eXt, 2013) and relationship-based access control (ReBAC) (Crampton and Sellwood, 2014; Fong and Siahaan, 2011), to enable flexible but concise specification of policies. The use of conditions in policies allows assignments or rules to be only effective in certain circumstances. For example, an ABAC policy in a university environment can state that users who meet the condition of being an employee are allowed to access the human resources portal. A natural consideration when using conditions is allowing *negated conditions*. For example, we can further restrict the abovementioned policy by stating that users who are employees and *not on leave* are allowed to access the portal. Such negated conditions allow the specification of fine-grained conditional expressions. An alternative approach to achieve similar flexibility is to allow *negative authorizations* (deny rules), in addition to the commonly-used positive authorizations. In contrast to a positive authorization which determines what is allowed, negative authorization specifies what is not allowed. In a policy system that allows both positive and negative authorizations, conflict situations can arise in cases where both positive and negative authorization are applicable. In order to resolve conflicts, a conflict resolution policy (such as *deny-overrides*) is employed. For example, the abovementioned policy can be expressed as a combination of a positive and negative authorization: 1) an employee is allowed to access the portal, and 2) an on-leave user is not allowed to access the portal.

As seen above, both the negated condition and negative authorization seem to provide comparable expressiveness for specifying flexible policies. For this reason, access control models in the literature typically adopt one of these approaches. While we can see instances of similar expressiveness of the two approaches, our research question is how they compare theoretically. Can both ap-

* Corresponding author.
E-mail addresses: riyer2@albany.edu (P. Iyer), amasoumzadeh@albany.edu (A. Masoumzadeh), pnarendran@albany.edu (P. Narendran).

https://doi.org/10.1016/j.cose.2021.102586
0167-4048/

proaches provide equivalent expressiveness? Is one of them more expressive than the other? Are they comparable at all? Answering these questions is critical for security engineers as it helps with the choice of proper access control models for implementation depending on the desired features and expressiveness. Furthermore, it will influence the design of future access control models that intend to benefit from negated conditions or negative authorization.

In this paper, we investigate the above questions using a formal framework. We contextualize our discussions in a rule-based access control model, which we define as a superset of modern policies such as ABAC and ReBAC. To the best of our knowledge, this is the first work in the literature to formally analyze and compare the expressive power of negated conditions and negative authorization. We summarize our key contributions as follows:

- We propose two abstract policy models, called *Negation* and *Deny*, which model rule-based policies with negated conditions and negative authorizations, respectively (Section 2). In our models, we consider a comprehensive set of meta-policy options that determine default authorization and conflict resolution strategy.
- We further propose a representation of our policies in the form of logic programs (Section 4), which serve as main constructions for our formal analysis and proofs.
- We formalize the Negation and Deny schemes based on a state-transition representation of the proposed models and the set of queries for comparing their expressiveness (Section 3). We formulate our expressiveness comparison problem based on the seminal work by Tripunitara and Li (2007).
- As the major result of this paper, we show that the Negation scheme is strictly more expressive than the Deny scheme (Section 5). More specifically, we prove that the Deny scheme is not as expressive as the Negation scheme (Section 6) *and* that the Negation scheme is at least as expressive as the Deny scheme (Section 7). Additionally, we prove the mapping from the Deny scheme to the Negation scheme (Appendix A).

We provide a comprehensive overview of the closely related work in Section 8 and concluding discussions in Section 9.

2. Formal Policy Language Specification

In this section, we describe the formal policy language specification for the rule-based access control model. In the rule-based access control paradigm, the authorizations are enforced on the basis of a set of rules that specifies the conditions for accessing the protected objects. Our main goal in the design of the policy language in this section is to formalize a comprehensive model that can incorporate or be easily extended based on various proposals in the literature. Our rule-based policy language is not supposed to be yet another model to capture more expressive authorization or administrative policies.

We emphasize that the policy language specified in this paper applies generically to rule-based access control models. For instance, we illustrate how our policy language applies to ABAC and ReBAC. This is because our goal in this work is to provide a formal comparison on the expressive power of two abstract policy models, namely Negation and Deny, which model rule-based policies with negated conditions and negative authorizations, respectively. As a significant consequence, the results regarding the expressiveness comparison demonstrated in this paper are applicable to any rule-based access control model.

In this section, we discuss the formats of various elements of a rule-based access control model like conditions, access requests, and authorization rules and policy (Section 2.1). Further, we discuss special rule-based access control models, namely the Negation model and the Deny model, which provide the basis for our expressiveness results (Section 2.2). Finally, we discuss the different authorization behaviors that can result from a policy specification, which is important in showing our expressiveness results about them (Section 2.3).

2.1. Rule-Based Access Control

An *access control policy* specifies which subjects can perform what operations on which objects. In a typical system, subjects are the users and objects are the protected resources. An *access request* specifies the user initiating the request and the protected resource on which access is requested. In order to determine the *access decision* for an access request, the enforced access control needs to check the satisfaction of conditions on users and resources involved in the access request. We note that access requests may also include a particular requested action, such as reading or writing to the object. However, we omit actions in access requests and policy rules, to simplify the presentation and the discussion of our results.

Definition 1 (Access request)**.** An *access request*, indicated by tuple $\langle subject, object\rangle \in S \times O$, contains the requesting *subject* and the requested *object*, where S is the set of subjects and O is the set of objects in the system. An *access decision* $\in \{\texttt{PERMIT}, \texttt{DENY}\}$ specifies the result of evaluating the submitted access request against the enforced policy.

Example 1 (Access request)**.** $\langle Alice, File1\rangle$ indicates an access request, where user *Alice* requests access to the protected resource *File1*.

Metadata on subjects and objects are used in determining authorizations in a system. An access control policy makes use of constructs called *conditions* to determine access decisions based on the evaluation of metadata on subjects and objects. We formally define conditions as follows.

Definition 2 (Condition)**.** A *condition* C, defined as $C : S \times O \rightarrow \{true, false\}$, evaluates to a Boolean value given a *subject* $\in S$ and an *object* $\in O$.

Note that the above definition is intentionally abstract in terms of metadata structure and internal logic of a condition. This allows the definition to be applicable to general rule-based policies. In the following, we provide examples of conditions in the context of ABAC and ReBAC.

Example 2 (Conditions)**.**

- ABAC: Metadata is specified in terms of attributes of users and resources, so conditions are evaluated based on the attribute information. For instance, in an educational institution, the condition “chairSbj” will be true if the given subject is the chair of a department, “transcriptObj” will be true if the given object is of type transcript, and “sameDeptSbjObj” will be true if the given subject and object belong to the same department.
- ReBAC: Metadata is represented in the form of a graph where nodes indicate users and resources, and edges indicate relationships between them. So, conditions are evaluated based on the paths between a subject (user) and object (resource) in the graph. For instance, in an online social network, the condition “friend-owner” will be true if there is such a path in the graph between the given subject and object. In other words, the subject should be a friend of the person who is the owner of the object.

Note that, although a condition takes both subject and object as parameters, the logic of a condition can ignore one of the arguments during evaluation. This is because in some cases a condition is evaluated only on a subject or only on an object in access control. For instance, in the above example, the condition “*chairSbj*”

considers only the given subject, and ignores the object argument, during access request evaluation. Similarly, the condition "*transcriptObj*" is evaluated only on the given object. But, the conditions "*sameDeptSbjObj*" and "*friend-owner*" consider both subject and object for evaluation.

Rule-based policies consist of *authorization rules*. An authorization rule includes the conjunction of multiple conditions on subjects and objects, and an access decision, called *rule effect*, that indicates the effect of that rule. When all the specified conditions in an authorization rule are satisfied by the subject and object involved in an access request, the rule becomes *applicable* to the request and the corresponding rule effect would be applied; satisfaction of *all* conditions is necessary due to the conjunction of conditions in a rule. Moreover, a condition within a rule can be either positive or negative, indicating whether the condition should be satisfied or not, respectively, for the corresponding rule to be applicable.

Definition 3 (Authorization rule)**.** An *authorization rule* is indicated by the tuple $\langle \phi, decision \rangle$ where $\phi = [\neg]C_1 \wedge [\neg]C_2 \wedge \cdots \wedge [\neg]C_n$ is a conjunction of conditions that are evaluated on the access request subject and object, and $decision \in \{\texttt{PERMIT}, \texttt{DENY}\}$ is the rule effect. Each condition can be optionally negated (using $\neg$) to indicate a condition that should *not* be satisfied for the rule to be applicable.

Example 3 (Authorization rules)**.**

- ABAC: The authorization rule in an educational institution "*The chair of a department can view the transcripts of students in that department*" can be written as: $\langle chairSbj \wedge transcriptObj \wedge sameDeptSbjObj, \text{PERMIT} \rangle$
- ReBAC: The authorization rule in an online social network "*Friends of a user can access her posts*" can be represented as: $\langle friend\text{-}owner, \text{PERMIT} \rangle$

Evaluating an access decision against a set of authorization rules can lead to ambiguities, which can be addressed using a *meta-policy*. Specifically, while checking the applicability of authorization rules to an access request, there could be scenarios where no authorization rule applies to the request, or more than one rule with conflicting (different) decisions are applicable. In order to account for these scenarios, a meta-policy indicates a *default decision* and a *conflict resolution strategy*. The default decision is considered when no authorization rule applies to an access request. The conflict resolution strategy is considered when an access request matches more than one rule with conflicting rule effects. In the context of this paper, we consider *DENY-overrides* and *PERMIT-overrides* conflict resolution strategies. The DENY-overrides strategy gives priority to the DENY decision as long as at least one of the applicable rules indicates the DENY effect. Similarly, PERMIT-overrides gives priority to the PERMIT decision.

Putting together the aforementioned discussion about various components, we define a rule-based access control policy as follows:

Definition 4 (Access control policy)**.** A *rule-based access control policy* or *authorization policy*, denoted as P, is represented as a tuple $\langle R, \delta, \rho \rangle$, where R is a set of authorization rules, $\delta \in \{\texttt{PERMIT}, \texttt{DENY}\}$ is the default decision and $\rho \in \{\texttt{PERMIT}\text{-}overrides, \texttt{DENY}\text{-}overrides\}$ is the conflict resolution strategy.

2.2. Negation Model and Deny Model

Our goal in this paper is to compare the expressiveness of models that employ negated conditions with those that employ negative authorizations. The negated conditions and negative authorizations are typically used in access control policies to deny certain users from accessing certain resources. In order to facilitate formal discussion on the expressiveness comparison between two systems, we need formal policy models that can be utilized to enforce authorizations in the considered systems, and also regulate the evolution of the system authorization state. In this paper, we consider two abstract policy models, namely the *Negation model* and *Deny model* which, respectively, represent the access control configurations for systems utilizing negated conditions and systems utilizing negative authorizations (deny rules).

Based on Definitions 3 and 4, we provide specialized definitions of access control policies for the Negation model and the Deny model to formally highlight the differences in the policy specifications between the two models. We employ the dot notation to indicate an element within a concept.

Definition 5 (Negation model)**.** *Negation model*, denoted as P_N, is a special rule-based access control defined as $\langle R, \{\texttt{DENY}\}, \emptyset \rangle$, where $\forall r \in R : r.decision = \texttt{PERMIT}$ and $r.\phi = [\neg]C_1 \wedge [\neg]C_2 \wedge \cdots \wedge [\neg]C_n$. Here, $[\neg]$ denotes that the negation operator is optional.

In the Negation model, an access request is permitted if at least one of the authorization rules is applicable to the request. Note that in the Negation model the meta-policy is fixed: the default decision δ is DENY and there is no need for the conflict resolution strategy ρ since the Negation model does not include rules with DENY decision. Also note that the default decision cannot be PERMIT, since otherwise no access request would be denied (unable to specify any access control restrictions).

Definition 6 (Deny model)**.** *Deny model*, denoted as P_D, is a special rule-based access control defined as $\langle R, \delta, \rho \rangle$, where $\forall r \in R : r.\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$.

In the Deny model, if an access request satisfies all conditions within an authorization rule, then its access decision is determined by the applicable rule's effect, which can be PERMIT or DENY. The ambiguities that can arise during the evaluation of an access request are solved using a *meta-policy,* which includes the default decision δ and the conflict resolution strategy ρ. Note that in the Deny model we can have different combinations of meta-policy depending on the value of δ and ρ.

A major difference between the behaviors of the Negation model and the Deny model stems from the fact that the former models with *positive-only* authorizations while the latter models with *positive/negative* authorizations. The positive-only policy model consists of a set of only PERMIT rules, and consequently, the default decision is DENY in case no rule is applicable to an access request. Furthermore, there is no conflict resolution strategy associated with positive-only authorizations. On the other hand, the positive/negative policy model consists of a set of PERMIT and DENY rules, requiring the need for a conflict resolution strategy which could be either PERMIT-overrides or DENY-overrides. Also, the default decision could be either PERMIT or DENY in positive/negative authorizations.

Another important distinction between the Negation model and the Deny model is that the former supports both positive as well as negated conditions within each authorization rule. On the other hand, the Deny model only supports positive conditions within a rule. A positive condition within an authorization rule requires satisfaction of the condition by the user and resource of an access request in order for the corresponding rule to be applicable. In contrast, a negated condition requires that a condition not be satisfied for a rule to be applicable.

In the following example, to clarify the differences between policies that can be represented in the Negation model and the Deny model, we use a Venn diagram to specify the authorization space of a policy:

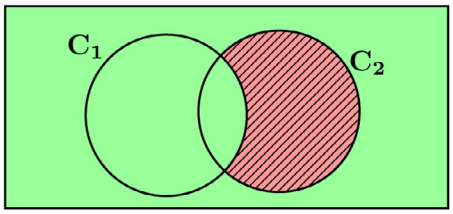

Fig. 1. Authorization space of an abstract policy on two conditions $\mathbf{C_1}$ and $\mathbf{C_2}$. Green area (solid) indicates PERMIT authorization space while red area (dashed) indicates DENY authorization space.

Example 4 (Negation and Deny models)**.** We illustrate how the authorization space of some abstract policy as depicted in Fig. 1 can be expressed in the two models. Here, the universe represents all possible access requests and corresponding decisions in a system. $\mathbf{C_1}$ and $\mathbf{C_2}$ indicate the conditions that can be specified on subjects and objects in the system.

- The given authorization scenario can be represented in the Negation model using the following policy rules: $P_N.R = \{\langle C_1, \texttt{PERMIT}\rangle, \langle \neg C_2, \texttt{PERMIT}\rangle\}$. Note that the entire region corresponding to the rule $\langle C_1, \texttt{PERMIT}\rangle$ is permitted, even though the rule $\langle \neg C_2, \texttt{PERMIT}\rangle$ exists in the policy. This is because, in the Negation model, even if one rule is applicable to a given access request, the authorization decision would be PERMIT.
- The authorization scenario given in Fig. 1 can be expressed in the Deny model using the following policy: $P_D = \langle\{\langle C_1, \text{PERMIT}\rangle, \langle C_2, \text{DENY}\rangle\}, \text{PERMIT}, \text{PERMIT-}\textit{overrides}\rangle$. The access requests in the overlap area $C_1 \cap C_2$ have two applicable rules with different rule effects, and so the PERMIT-overrides strategy is used to resolve this conflict.

2.3. Different Combinations in Meta-Policy

A rule-based access control policy consists of authorization rules and meta-policy, which in turn includes a default decision and a conflict resolution strategy (Definition 4). Different combinations of meta-policy, by varying the default decision and the conflict resolution strategy, will result in different behavior of authorization policies. We note that the discussion in this section applies only to the Deny model since, in the Negation model, there is just one possible meta-policy as there is no concept of conflict resolution and the default decision is fixed to DENY.

Within an access control policy, we consider two values for default decision, that is PERMIT and DENY, and two values for conflict resolution strategy, that is PERMIT-overrides and DENY-overrides (Definition 4). Thus, in total, we would have four possibilities for the meta-policy component in an authorization policy. The distinction between these meta-policy choices is important in showing the applicability of our expressiveness comparison results over a wide scope of access control scenarios that usually arise in practice (as discussed in Sections 6 and 7).

Fig. 2 depicts authorization spaces for a very simple example policy when different meta-policy combinations are chosen. Here, the green (solid) and the red (dashed) areas, respectively, indicate the PERMIT authorization space and the DENY authorization space. The universe represents all possible access requests and corresponding decisions in the system. For clarity of discussions, we consider a very simple abstract policy ruleset $P_D.R = \{\langle C_1, \text{PERMIT}\rangle, \langle C_2, \text{DENY}\rangle\}$. Each circle in the figure corresponds to one of the rules (and corresponding condition) and encompasses the access instances to which it is applicable. So all access requests within the C_1 region would be permitted, and vice-versa for C_2. In case an access request does not lie within either C_1 or C_2, then the appropriate default decision is applied. Moreover, when an access request exists within $C_1 \cap C_2$, the appropriate conflict resolution strategy is employed (as discussed in Section 2.1).

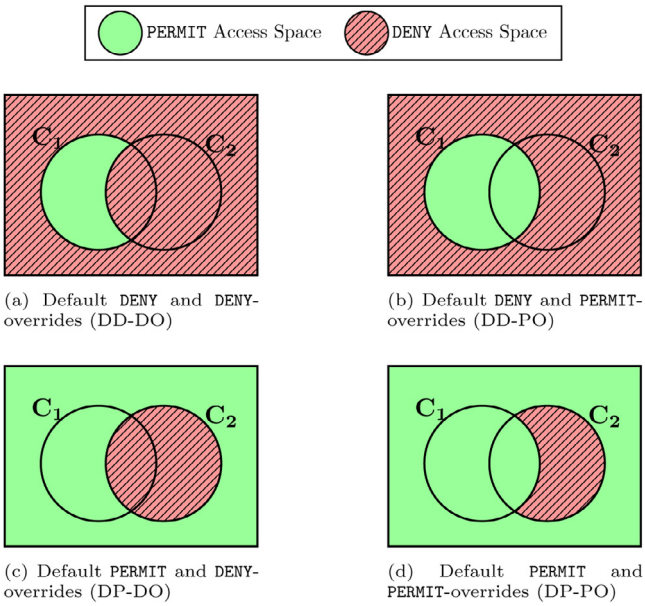

Fig. 2. The effect of different meta-policies (combination of default decision and conflict resolution strategy) on authorization decision. An abstract policy on two conditions $\mathbf{C_1}$ and $\mathbf{C_2}$ is considered, where a PERMIT rule applies to requests satisfying $\mathbf{C_1}$ and a DENY rule applies to requests satisfying $\mathbf{C_2}$.

As shown in the figure, the policy specification will lead to drastically different authorizations depending on the choice of meta-policy combination. We now explain the practicality of each of the above-mentioned meta-policy combinations. However, as we detail here, we limit our consideration to two of those cases (DD-DO and DP-PO) as the most practical choices in the rest of the paper.

2.3.1. Default DENY and DENY-Overrides (DD-DO)

This choice of meta-policy is utilized for systems with very strict security requirements such that only those users who are explicitly given permission are authorized to perform accesses on protected resources. Moreover, if an access request is permitted by some rules, but denied by some other rules, then it will be eventually denied. DD-DO meta-policy is based on the "fail-safe defaults" principle (Saltzer, Schroeder, 1975) according to which whenever a "failure" happens during the evaluation of an access request, the safest option is to deny that request. The default case when no rule applies as well as the conflict case are considered as failure. Fig. 2a demonstrates the DD-DO scenario where the access requests that strictly satisfy only the condition C_1 (i.e., $C_1 \setminus C_2$ where the backslash denotes set difference operator) are permitted access.

2.3.2. Default DENY and PERMIT-Overrides (DD-PO)

According to this meta-policy, if an access request does not satisfy any rule in the authorization policy then it would be denied. But if it satisfies more than one rule with conflicting decisions then it would be eventually permitted to ensure availability of privileges. This kind of meta-policy is generally not practical since the inconsistent choices for the default decision and conflict resolution seem counter-intuitive. With this kind of meta-policy, the effect of the negative rules in an authorization policy is never taken into account. In other words, the system authorizations would remain the same even when all the negative rules are removed from the access control policy. For example, as shown in Fig. 2b, even if we remove the rule $\langle C_2, \texttt{DENY}\rangle$, the authorization space of access re-

quests still remains the same, so C_2 basically has no effect on authorizations in this case.

2.3.3. Default PERMIT and DENY-Overrides (DP-DO)

This meta-policy scenario is the converse of the above one, i.e., access requests that do not satisfy any authorization rule are permitted, while access requests with conflicting access decisions would be denied. Again, like in the previous case, this kind of meta-policy is not practical because of redundant positive rules. For instance, in Fig. 2c, the authorization rule $\langle C_1, \texttt{PERMIT}\rangle$ has no effect while determining the authorizations of access requests.

2.3.4. Default PERMIT and PERMIT-Overrides (DP-PO)

This kind of meta-policy is the converse of the DD-DO scenario. This is used when there is a group of users who are explicitly denied access to certain resources, while everyone else is entitled to all permissions. If a conflict arises during the evaluation of an access request, i.e., it is permitted by some rules while denied by some other rules, then the access request is granted permission. For instance, in Fig. 2d, only the access instances that exist exclusively within the C_2 region, i.e., $C_2 \setminus C_1$, are denied.

3. State-Transition Model & Expressiveness Comparison Framework

In this section, we discuss the framework on the basis of which we compare the expressiveness of the Negation model (Definition 5) and the Deny model (Definition 6). In addition, we describe how a system with rule-based policy can be seen as a state-transition model, which is a requirement for our expressiveness analysis.

3.1. Expressiveness Comparison Framework

A real-world system keeps evolving as users and resources are added/removed, conditions on them are updated, and/or access control policy is updated to reflect new security requirements in the system. So, the state of the system is dynamic, and the permissions that were once granted may be denied in the future or vice versa. We employ a state-transition paradigm for modeling this dynamic behavior.

We adopt the approach proposed by Tripunitara and Li (2007) to compare the expressiveness of the Negation and the Deny systems, based on the preservation of enforced authorizations. This is accomplished using the notion of *queries*, which is used for understanding the authorization behavior of an access control system. In the context of this paper, queries correspond to access requests used to determine which accesses are allowed in a given state of a system. When we discuss about comparing the expressive power, we need to be specific about the queries, states, and state-transition policies while specifying an access control model, which we refer to as an *access control scheme*. Definitions 7, 8, and 9 adopted from the work of Tripunitara and Li (2007) describe access control scheme, state-matching reduction, and expressiveness comparison, respectively.

Definition 7 (Access control scheme)**.** An *access control scheme* is a state-transition system represented in the form of tuple $\langle \Gamma, Q, \vdash, \Psi\rangle$ in which Γ denotes the set of states, Q indicates the set of queries, $\vdash: \Gamma \times Q \rightarrow \{True, False\}$ is called the *entailment* relation, and Ψ denotes the set of state-transition policies.

Definition 8 (State-matching reduction)**.** Given two access control schemes $B = \langle \Gamma, Q, \vdash, \Psi\rangle$ and $B' = \langle \Gamma', Q', \vdash', \Psi'\rangle$, a *mapping* from B to B' is represented as $\lambda : (\Gamma \times \Psi) \cup Q \rightarrow (\Gamma' \times \Psi') \cup Q'$. A mapping λ from B to B' is called a *state-matching reduction* if, for every $\gamma \in \Gamma$ and every $\psi \in \Psi$, $\langle \gamma', \psi'\rangle = \lambda(\langle \gamma, \psi\rangle)$ conforms to the following two properties:

- Property 1: For every state γ_1 in scheme B reachable from γ, that is, $\gamma \overset{*}{\mapsto}_{\psi} \gamma_1$, there exists state γ_1' in scheme B' such that $\gamma' \overset{*}{\mapsto}_{\psi'} \gamma_1'$ and that the states γ_1 and γ_1' are equivalent under the mapping λ.
- Property 2: For every state γ_1' in scheme B' reachable from γ' $(= \lambda(\gamma))$, that is, $\gamma' \overset{*}{\mapsto}_{\psi'} \gamma_1'$, there exists state γ_1 in scheme B such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma_1$ and that the states γ_1 and γ_1' are equivalent under the mapping λ.

Two states γ and γ' are said to be *equivalent* under the mapping λ whenever for every query $q \in Q$, $\gamma \vdash q$ if and only if $\gamma' \vdash' \lambda(q)$.

Definition 9 (Expressiveness Comparison)**.** Given two access control schemes, B and B', we say that B is *at least as expressive as* B', if there exists a state-matching reduction from the scheme B' to the scheme B. Further, if there also exists a state-matching reduction from B to B', then B and B' are *equivalent in expressive power*. If B is at least as expressive as B', and there exists no state-matching reduction from B to B', then scheme B is *strictly more expressive* than scheme B'.

3.2. Modeling Security System as a State-Transition Model

A *protection state* of a system is the subset of the system state that determines authorizations. For example, in the ReBAC model, authorizations are determined based on a set of rules that evaluate some specified conditions (i.e., paths between subject and object) in the system graph (see Example 2). So, the protection state in a ReBAC model consists of users (subjects), resources (objects), relationships among them, and authorization policy (Masoumzadeh, 2018). We formally define the protection state of a rule-based access control model as follows:

Definition 10 (Protection state)**.** A *protection state*, represented as $\gamma = \langle \mathcal{C}, P\rangle$, consists of the set of conditions $\mathcal{C}$ on subjects and objects in a system, and the access control policy P that determines authorizations in the current system state based on evaluations of the conditions in $\mathcal{C}$.

Based on the above definition, a protection state can be altered either through changes to the evaluations of system conditions (i.e., updating the truth assignment of a condition $C \in \mathcal{C}$), or by amending the access control policy P. Changes to access control policy can be made through either insertion or deletion of a rule $r \in P.R$ or modifying the meta-policy $\langle \delta, \rho\rangle$ within P (in case of the Deny model). In other words, any change to the protection state would result in a different set of permissions in the application.

In the context of our rule-based access control model, the different components of an access control scheme (Definition 7) can be described as given below:

- **State**: A state $\gamma \in \Gamma$ is defined as in Definition 10, and contains all the information necessary for determining access controls at a given time instant.
- **Query**: We consider only authorization queries (i.e., access requests). Particularly, a query $q \in Q$ is of the form "*Given the conditions on subjects and objects, can a certain subject be authorized to access a specific object*?". In the context of expressiveness comparison, we consider the conjunction of all the possible access requests in a system since we are interested in preserving the access permissions of the system.
- **Entailment**: The entailment relation $\vdash$ checks the validity of a query in a given state. We write $\gamma \vdash q$ whenever a query $q \in Q$

is satisfied in the state γ, that is if the involved access request is granted permission in that state. Otherwise, $\gamma \nvdash q$.

- **State-transition policy**: A state-transition policy $\psi \in \Psi$ is a binary relation (denoted by $\mapsto_\psi$) defined over Γ. Given $\gamma, \gamma' \in \Gamma$, $\gamma \mapsto_\psi \gamma'$ indicates that the state γ transitions to γ' under the regulation of ψ. We write $\gamma \overset{*}{\mapsto}_\psi \gamma'$ to denote transitive and reflexive closure of $\mapsto_\psi$, and γ' is said to be *reachable* from γ in zero or more allowed transitions.
  A state-transition policy determines how the protection state changes over time. As mentioned earlier, the state changes in our rule-based access control model occur with modifications either to the system conditions $\mathcal{C}$ or to the access control policy P of the current state. In the access control literature, such transitions are commonly referred to as *administrative operations* (Cheng et al., 2016; Crampton and Sellwood, 2016; Jin et al., 2012; Rizvi et al., 2015; Stoller, 2015). We consider the following administrative operations:
  - *add_rule(r)*: Adding a rule r to authorization policy in the current state γ.P
  - *delete_rule(r)*: Deleting a rule r from the current authorization policy γ.P
  - *update_condition*(C, C'): Changing the evaluation (i.e., truth assignment) of a condition C in $\gamma.\mathcal{C}$ to C'

Note that we are limiting our definition of administrative operations to keep our analysis presentation manageable. Otherwise, flexible administrative policies such as conditions on the subject initiating the state-change request and modifying the meta-policy component of authorization policy (in the case of the Deny model) can be considered too.

4. Logic-Based Representation of Rule-Based Policies

In this section, we describe a logic-based (clausal) representation of the Negation and the Deny policies, defined in Section 2, in order to present their semantics and provide a unified framework for our theoretical analysis.[1] Note that even though there are previous works that have proposed logic-based representation for access control policies (Bertino et al., 1999; Chomicki et al., 2003; di Vimercati et al., 2005; Jajodia et al., 1997; Wang et al., 2004), our focus is to provide a simple representation sufficient for our expressiveness comparison.

We utilize the following predicates in our logic program for representing access control policies:

- C_i indicates a condition predicate that evaluates metadata associated with subjects and objects. Condition predicates are used within policy rules to determine authorizations.
- *exec_permit* and *exec_deny* predicates indicate the rule effect (PERMIT and DENY, respectively) for the associated authorization rule.
- *permit* and *deny* predicates are used to determine the final access decision after applying the included meta-policy.

All predicates are parameterized in terms of access requests, i.e., all predicates have two arguments, one for the subject and the other for the object.

4.1. Logic Program for the Deny Model

As shown in Section 2.2, a Deny policy consists of a disjunctive set of positive and negative authorization rules and meta-policy for resolving ambiguities during access request evaluation.

[1] We use the logic programming formalism for representing authorization policies; but not all of them are Horn clauses, as readers would notice.

Positive Authorization Rules. In order to represent a positive authorization rule in logic format, we create a conditional statement with the *exec_permit* predicate as the consequent and the conjunction of conditions in the rule as the antecedent, as given in (#1). Here, *?u* and *?r* denote variables on users and resources and they are universally quantified, i.e., for any user and for any resource in the system the given statement holds. We use the subscript D to indicate that a predicate belongs to the Deny model in contrast to similar predicates that belong to the Negation model (subscripted by N). If any of the positive authorization rules in a policy becomes applicable to an access request, then the *exec_permit* predicate for that access request would be true.

$$exec\_permit_D(?u, ?r) \text{ :- } C_i(?u, ?r), \ldots, C_j(?u, ?r). \quad (\#1)$$

Negative Authorization Rules. The conditional statement associated with a negative authorization rule is similar to that of a positive rule, except that it includes the *exec_deny* predicate as the consequent as given in (#2). So, if a negative rule is applicable to an access request, then the *exec_deny* predicate corresponding to that access request would be true.

$$exec\_deny_D(?u, ?r) \text{ :- } C_n(?u, ?r), \ldots, C_p(?u, ?r). \quad (\#2)$$

We note that, as shown in Definition 6, all conditions in the antecedent of conditional statements associated with the positive as well as the negative authorization rules are positive (i.e., unnegated).

Access Decision and Meta-Policy Rules. After checking the applicability of rules to an access request, there can be four, mutually exclusive scenarios while determining its authorization decision, which is represented in the form of a logic program as follows:

$$\delta_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad (\#3a)$$

$$\rho_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \quad (\#3b)$$

$$permit_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad (\#3c)$$

$$deny_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \quad (\#3d)$$

As mentioned earlier, the *permit* and *deny* predicates in our logic-based representation indicate the PERMIT and DENY access decisions, respectively. We refer to (#3a) and (#3b) as *meta-policy rules*, in which the δ and the ρ are replaced with the *permit* and *deny* predicates (corresponding to the PERMIT and the DENY access decisions). We refer to (#3c) and (#3d) as *access decision rules*.

When none of the authorization rules within a policy is applicable to an access request, i.e., neither *exec_permit* nor *exec_deny* predicate is true, the default decision $\delta \in \{\texttt{PERMIT}, \texttt{DENY}\}$ would be applied (see (#3a)). When both *exec_permit* and *exec_deny* predicates are true for an access request, we need a conflict resolution strategy to produce a final access decision (see (#3b)). If the conflict resolution strategy is DENY-overrides, then the final decision is given by $\rho = \texttt{DENY}$; otherwise, for PERMIT-overrides the final decision would be $\rho = \texttt{PERMIT}$. When only positive authorization rule(s) are applicable to an access request, the *exec_permit* predicate will be true corresponding to that access request, but the *exec_deny* predicate will not be. So, naturally, the final authorization decision would be PERMIT in this case (see (#3c)). Conversely, if only the *exec_deny* predicate is true, and not the *exec_permit* predicate, the final decision would be DENY (see (#3d)).

4.2. Logic Program for the Negation Model

As shown in Section 2.2, a Negation policy consists of a disjunctive set of positive authorization rules. So, we have only *exec_permit* predicate and no *exec_deny* predicate. In addition, as shown in Definition 5, we can have positive as well as negative conditions within an authorization rule.

Authorization Rules. To represent authorization rules in the logic format, similar to Section 4.1, we create a conditional statement with *exec_permit* as the consequent and the conjunction of conditions as the antecedent. For an authorization rule with only positive conditions, the conditional statement would be of the form given in (#1). Whereas, for an authorization rule with both positive and negative conditions, the conditional statement would be as given in (#4). Again, the predicates are parameterized in terms of access requests, and the variables are universally quantified. We use the subscript N to indicate that a predicate belongs to the Negation model. If an access request satisfies all positive conditions and does not satisfy any negative condition in an authorization rule, then the *exec_permit* predicate corresponding to that access request would be true.

$$exec\_permit_N(?u, ?r) \text{ :- } C_i(?u, ?r), \ldots, C_j(?u, ?r), \neg C_n(?u, ?r), \ldots, \neg C_p(?u, ?r). \quad (\#4)$$

Access Decision and Meta-Policy Rules. After evaluating the applicability of authorization rules to an access request, there can be two, mutually exclusive scenarios, given in (#5a) and (#5b), while determining its final authorization decision. Again, the *permit* and the *deny* predicates indicate the PERMIT and the DENY access decisions, respectively. When no authorization rule applies to an access request, the *exec_permit* predicate corresponding to it will not be true, so the default decision DENY is the final authorization decision for that access request (see (#5a)). On the other hand, when at least one of the positive authorization rules is applicable to an access request, i.e., the *exec_permit* predicate is true, the final access decision is PERMIT (see (#5b)).

$$deny_N(?u, ?r) \text{ :- } \neg exec\_permit_N(?u, ?r). \quad (\#5a)$$

$$permit_N(?u, ?r) \text{ :- } exec\_permit_N(?u, ?r). \quad (\#5b)$$

5. Expressive Power: Negation Scheme vs. Deny Scheme

In this section, we state our main result on the expressiveness comparison between the Negation and the Deny models (as defined in Section 2). The detailed proof for the result will be presented in Sections 6 and 7. We rely on the framework for expressiveness comparison that was presented in Section 3, particularly on state-matching reductions (Definition 8). We also rely on the logic-based representation of policies (as presented in Section 4) for our proofs.

As mentioned in Section 3, our formal expressiveness comparison will be done based on *schemes*. As described in Definition 7, an access control scheme specifies the states, state-transition policies, and queries about an access control model. Given the Negation and the Deny schemes, we investigate whether one can simulate the other. In other words, given the Negation scheme, can the Deny scheme preserve all the authorizations? As we will demonstrate, there exists a protection state in the Negation scheme for which there exists no equivalent protection state in the Deny scheme. In addition, can the Negation scheme preserve the authorizations enforced by the Deny scheme? Interestingly, as we will demonstrate, the Negation scheme can represent all authorization policies that can be expressed in the Deny scheme.

In order to facilitate our discussion, we make certain assumptions about the environment in which we develop our expressiveness comparison results. We assume that both schemes operate in the same evaluation environment. Therefore, conditions on subjects and objects used by authorization policies are evaluated in the same manner and shared by the Negation and Deny schemes. Note that we did not use subscripts N and D for conditions in the logic programs, described in Section 4, for this reason. For instance, in Fig. 2, evaluations of the conditions C_1 and C_2 for all access requests are the same for both the schemes. However, note that the authorization policy can vary between the Negation and the Deny schemes.

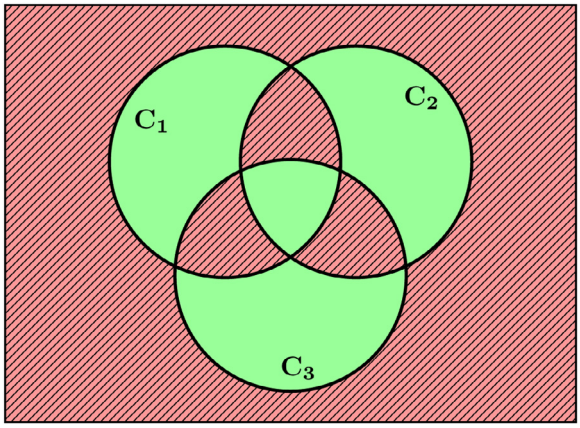

Fig. 3. Authorization configuration of protection state $\widehat{\gamma^N}$ in Negation scheme. $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ indicate three conditions in $\widehat{\gamma^N}$. Authorization decisions for each combination of condition evaluations depicted as: green (solid) for PERMIT and red (dashed) for DENY.

We state our main result on the expressiveness comparison in the following theorem:

Theorem 1. *The Negation scheme is strictly more expressive than the Deny scheme based on state-matching reductions.*

We prove the above theorem in the next two sections. In particular, we prove the following results:

- There exists no state-matching reduction from the Negation scheme to the Deny scheme (Section 6). Therefore, the Deny scheme is not as expressive as the Negation scheme.
- There exists a state-matching reduction from the Deny scheme to the Negation scheme (Section 7). Therefore, the Negation scheme is at least as expressive as the Deny scheme.

Informally, Theorem 1 states that all authorization policies that can be represented in the Deny scheme can be represented in the Negation scheme as well, whereas the converse is not true. Therefore, the Negation scheme can cover wider authorization policies, and can replace the Deny policies if desired.

6. Expressive Power of Deny Scheme

In this section, we show that the Deny scheme cannot capture all the authorizations enforced by the Negation scheme. For this purpose, we consider a specific protection state in the Negation scheme in Section 6.1. Based on state-matching reductions, in Section 6.2, we show that no equivalent protection state can exist in the Deny scheme.

6.1. A Protection State in Negation Scheme

Suppose we have a protection state in the Negation scheme, which we refer to as $\widehat{\gamma^N}$, with the authorization configuration as shown in Fig. 3. $\widehat{\gamma^N}.\mathcal{C}$ consists of three conditions C_1, C_2 and C_3. The figure depicts authorization decisions for each combination of condition evaluations: green (solid) indicates PERMIT and red

(dashed) indicates DENY. This authorization configuration can be expressed using the following ruleset in $\widehat{\gamma^N}$.P:

$$\begin{aligned}
&exec_permit_N(?u, ?r) \text{ :- } C_1(?u, ?r),\ C_2(?u, ?r),\ C_3(?u, ?r).\\
&exec_permit_N(?u, ?r) \text{ :- } C_1(?u, ?r),\ \neg C_2(?u, ?r),\ \neg C_3(?u, ?r).\\
&exec_permit_N(?u, ?r) \text{ :- } \neg C_1(?u, ?r),\ \neg C_2(?u, ?r),\ C_3(?u, ?r).\\
&exec_permit_N(?u, ?r) \text{ :- } \neg C_1(?u, ?r),\ C_2(?u, ?r),\ \neg C_3(?u, ?r).
\end{aligned} \tag{\#6}$$

If any of the authorization rules in (#6) is applicable to an access request, then the *exec_permit* predicate corresponding to that access request would be true. We note that the conditional statements for determining the final authorization decisions for access requests, indicated by the *permit* and *deny* predicates, were shown previously in (#5a) and (#5b).

6.2. Deny is Not As Expressive As Negation

Assuming a particular reachable state in the Negation scheme, we want to show that there is no equivalent (i.e., enforcing the same set of authorizations) state in the Deny scheme. More specifically, considering the reachable state $\widehat{\gamma^N}$ in the Negation scheme, discussed in Section 6.1, our goal is to show that none of the protection states in the Deny scheme can precisely express the authorization configuration of state $\widehat{\gamma^N}$. We assume that there exists a mapping λ that is a state-matching reduction (Definition 8) from the Negation scheme to the Deny scheme. So, there exists a protection state, which we refer to as $\widehat{\gamma^D}$, in the Deny scheme that preserves the authorization configuration in $\widehat{\gamma^N}$. Also, as discussed in Section 5, we consider that the three conditions C_1, C_2 and C_3 are shared between the Negation and the Deny schemes, so the evaluations of the conditions on access requests is the same for both schemes.

We consider generic authorization policy for the Deny scheme since we need to find the particular specification, including rules and meta-policy, that can express the given permissions of $\widehat{\gamma^N}$. In the Deny scheme, we have positive and negative authorization rules in the form of conditional statements as given in (#1) and (#2), respectively. The conditional statements for determining the final authorization decision for access requests are given in (#3a)-(#3d). In the following claims, we consider different possibilities for meta-policy by varying the values of default decision δ (see (#3a)) and conflict resolution strategy ρ (see (#3b)) to generate different policies in the Deny scheme. We show that there *does not* exist a protection state $\widehat{\gamma^D}$ with any of those possible policies in the Deny scheme that is able to represent all the authorizations of state $\widehat{\gamma^N}$ in the Negation scheme.

We first examine the Deny scheme implementing DD-DO meta-policy. In this case, both δ (see (#3a)) and ρ (see (#3b)) would be instantiated to the *deny* predicate as follows:

$$deny_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \tag{\#7a}$$

$$deny_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \tag{\#7b}$$

(#7a) shows the default decision statement and (#7b) indicates the conflict resolution strategy. The conditional statements for the authorization rules (see (#1) and (#2)) and the access decisions (see (#3c) and (#3d)) remain the same.

Claim 1. *There exists no state-matching reduction from the Negation scheme to the Deny scheme implementing DD-DO meta-policy.*

Proof. Let $\widehat{\gamma^N}$ be the protection state in the Negation scheme discussed in Section 6.1. Assume that there exists a mapping that is a state-matching reduction from the Negation scheme to the Deny scheme implementing DD-DO meta-policy. Let $\widehat{\gamma^D}$ be the equivalent state in the Deny scheme under such a mapping.

Consider the policy represented in Fig. 3. Without loss of generality, we consider an access request $\langle u_1, r_1\rangle$ that lies in the intersection $C_1 \cap \neg C_2 \cap \neg C_3$, i.e., $\langle u_1, r_1\rangle$ satisfies only the condition C_1, and does not satisfy either C_2 or C_3:

$$\begin{aligned}
&C_1(u_1, r_1).\\
&\neg C_2(u_1, r_1).\\
&\neg C_3(u_1, r_1).
\end{aligned} \tag{\#8}$$

The access request $\langle u_1, r_1\rangle$ is permitted in $\widehat{\gamma^N}$. Since we are assuming that there exists a state-matching reduction from the Negation scheme to the Deny scheme implementing DD-DO meta-policy, the state $\widehat{\gamma^D}$ should capture that $\langle u_1, r_1\rangle$ is permitted:

$$\begin{aligned}
&permit_N(u_1, r_1).\\
&permit_D(u_1, r_1).
\end{aligned} \tag{\#9}$$

In case of DD-DO meta-policy, the PERMIT authorization in the Deny scheme can only be obtained from the access decision statement in (#3c). Therefore, given (#9), we can make the following inferences about the authorization of access request $\langle u_1, r_1\rangle$ in state $\widehat{\gamma^D}$:

$$exec\_permit_D(u_1, r_1). \tag{\#10a}$$

$$\neg exec\_deny_D(u_1, r_1). \tag{\#10b}$$

Since we cannot use any negated condition in the Deny scheme, based on (#10a) and (#8), we can infer the following authorization rule in $\widehat{\gamma^D}$.P:

$$exec\_permit_D(?u, ?r) \text{ :- } C_1(?u, ?r). \tag{\#11}$$

Next, we consider an access request $\langle u_2, r_2\rangle$ that lies in the intersection $C_1 \cap C_2 \cap \neg C_3$:

$$\begin{aligned}
&C_1(u_2, r_2).\\
&C_2(u_2, r_2).\\
&\neg C_3(u_2, r_2).
\end{aligned} \tag{\#12}$$

$\langle u_2, r_2\rangle$ is denied in $\widehat{\gamma^N}$, so it should be denied in $\widehat{\gamma^D}$ as well:

$$\begin{aligned}
&deny_N(u_2, r_2).\\
&deny_D(u_2, r_2).
\end{aligned} \tag{\#13}$$

In the Deny scheme, the DENY authorization can be obtained from the default decision statement in (#7a), the conflict resolution statement in (#7b), or the access decision statement in (#3d). The default decision and the access decision statements require $\neg exec\_permit_D(u_2, r_2)$ to be true (see (#7a) and (#3d)). However, according to our inferred rule in (#11) and given that $\langle u_2, r_2\rangle$ satisfies the condition C_1 (see (#12)), the $exec\_permit_D(u_2, r_2)$ predicate will be true. So, our only way to obtain the predicate $deny_D(u_2, r_2)$ is through the conflict resolution statement in (#7b). Therefore, we can make the following inferences about the authorization of $\langle u_2, r_2\rangle$:

$$exec\_permit_D(u_2, r_2). \tag{\#14a}$$

$$exec\_deny_D(u_2, r_2). \tag{\#14b}$$

Based on the condition predicates in (#12) and the fact that the Deny scheme cannot support negated conditions, the predicate in (#14b) can be inferred from one of the following potential authorization rules in $\widehat{\gamma^D}$.P:

$$exec\_deny_D(?u, ?r) \text{ :- } C_1(?u, ?r). \tag{\#15a}$$

$$exec\_deny_D(?u, ?r) \text{ :- } C_2(?u, ?r). \tag{\#15b}$$

$$exec\_deny_D(?u, ?r) \text{ :- } C_1(?u, ?r), C_2(?u, ?r). \quad (\#15c)$$

Finally, we consider an access request $\langle u_3, r_3\rangle$ that lies in the intersection $C_1 \cap C_2 \cap C_3$:

$$\begin{array}{l} C_1(u_3, r_3). \\ C_2(u_3, r_3). \\ C_3(u_3, r_3). \end{array} \quad (\#16)$$

$\langle u_3, r_3\rangle$ is permitted in $\widehat{\gamma_N}$, so it should be permitted in $\widehat{\gamma^D}$ as well:

$$\begin{array}{l} permit_N(u_3, r_3). \\ permit_D(u_3, r_3). \end{array} \quad (\#17)$$

In the Deny scheme, since PERMIT authorization can only be obtained from (#3c), we can make the following inferences about the authorization of $\langle u_3, r_3\rangle$ in $\widehat{\gamma^D}$:

$$exec\_permit_D(u_3, r_3). \quad (\#18a)$$

$$\neg exec\_deny_D(u_3, r_3). \quad (\#18b)$$

Since $\langle u_3, r_3\rangle$ satisfies both the conditions C_1 and C_2 (see (#16)), the inferred rules in (#15a)-(#15c) are applicable to the access request. Therefore, we infer the following predicate in $\widehat{\gamma^D}$:

$$exec\_deny_D(u_3, r_3). \quad (\#19)$$

However, the two predicates in (#18b) and (#19) contradict each other. Therefore, our earlier assumption that a state-matching reduction exists is incorrect. □

We next examine the Deny scheme implementing DD-PO meta-policy. Although this case is generally not practical, as discussed in Section 2.3, we still include it for the sake of completeness of our results and also to clarify any doubt about this possibility for the readers. The default decision δ in (#3a) and conflict resolution ρ in (#3b) are instantiated as follows:

$$deny_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad (\#20a)$$

$$permit_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \quad (\#20b)$$

Claim 2. *There exists no state-matching reduction from the Negation scheme to the Deny scheme implementing DD-PO meta-policy.*

Proof. Let $\widehat{\gamma^N}$ be the protection state in the Negation scheme discussed in Section 6.1. Assume that there exists a mapping that is a state-matching reduction from the Negation scheme to the Deny scheme implementing DD-PO meta-policy. Let $\widehat{\gamma^D}$ be the equivalent state in the Deny scheme under such a mapping.

Consider the policy represented in Fig. 3. Without loss of generality, we consider an access request $\langle u_1, r_1\rangle$ that lies in the intersection $C_1 \cap \neg C_2 \cap \neg C_3$:

$$\begin{array}{l} C_1(u_1, r_1). \\ \neg C_2(u_1, r_1). \\ \neg C_3(u_1, r_1). \end{array} \quad (\#21)$$

$\langle u_1, r_1\rangle$ is permitted in state $\widehat{\gamma^N}$ of the Negation scheme, so it should also be permitted in state $\widehat{\gamma^D}$ of the Deny scheme:

$$\begin{array}{l} permit_N(u_1, r_1). \\ permit_D(u_1, r_1). \end{array} \quad (\#22)$$

In the Deny scheme, this PERMIT authorization can be obtained from either the conflict resolution statement in (#20b) or the access decision statement in (#3c). Therefore, we can make the following inference about the authorization of $\langle u_1, r_1\rangle$ in state $\widehat{\gamma^D}$:

$$exec\_permit_D(u_1, r_1). \quad (\#23)$$

Since we cannot use any negated condition in the Deny scheme, based on (#21) and (#23), we can infer the following authorization rule in $\widehat{\gamma^D}$.P:

$$exec\_permit_D(?u, ?r) \text{ :- } C_1(?u, ?r). \quad (\#24)$$

Next, we consider an access request $\langle u_2, r_2\rangle$ that lies in the intersection $C_1 \cap C_2 \cap \neg C_3$:

$$\begin{array}{l} C_1(u_2, r_2). \\ C_2(u_2, r_2). \\ \neg C_3(u_2, r_2). \end{array} \quad (\#25)$$

$\langle u_2, r_2\rangle$ is denied in $\widehat{\gamma^N}$, so it should be denied in $\widehat{\gamma^D}$ as well:

$$\begin{array}{l} deny_N(u_2, r_2). \\ deny_D(u_2, r_2). \end{array} \quad (\#26)$$

In the Deny scheme, this DENY authorization can be obtained from either the default decision statement in (#20a) or the access decision statement in (#3d). Both of them require that $\neg exec\_permit_D(u_2, r_2)$ should be true in order to infer the $deny_D(u_2, r_2)$ predicate. However, based on our inferred rule in (#24) and the condition predicates in (#25), $exec\_permit_D(u_2, r_2)$ is true, thereby causing contradiction. Therefore, our earlier assumption that a state-matching reduction exists is incorrect. □

Finally, we examine the Deny scheme implementing default PERMIT meta-policy. In this case, the default decision δ in (#3a) would be instantiated to the *permit* predicate as follows:

$$permit_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad (\#27)$$

We do not choose a specific value for the conflict resolution strategy, so it could be either PERMIT-overrides or DENY-overrides. Correspondingly, the ρ in (#3b) can be replaced with either *permit* or *deny* predicate.

Claim 3. *There exists no state-matching reduction from the Negation scheme to the Deny scheme implementing default PERMIT (including DP-DO and DP-PO) meta-policy.*

Proof. Let $\widehat{\gamma^N}$ be the protection state in the Negation scheme discussed in Section 6.1. Assume that there exists a mapping that is a state-matching reduction from the Negation scheme to the Deny scheme implementing default PERMIT meta-policy. Let $\widehat{\gamma^D}$ be the equivalent state in the Deny scheme under such a mapping.

Consider the policy represented in Fig. 3. We consider an access request $\langle u_1, r_1\rangle$ that satisfies neither C_1, nor C_2, nor C_3:

$$\begin{array}{l} \neg C_1(u_1, r_1). \\ \neg C_2(u_1, r_1). \\ \neg C_3(u_1, r_1). \end{array} \quad (\#28)$$

$\langle u_1, r_1\rangle$ is denied in state $\widehat{\gamma^N}$ of the Negation scheme, so it should be denied in state $\widehat{\gamma^D}$ of the Deny scheme as well:

$$\begin{array}{l} deny_N(u_1, r_1). \\ deny_D(u_1, r_1). \end{array} \quad (\#29)$$

In the Deny scheme, this DENY authorization can be obtained from either the conflict resolution statement in (#3b) (by replacing ρ with the *deny* predicate) or the access decision statement in (#3d). Both of these conditional statements require that the predicate $exec\_deny_D(u_1, r_1)$ be true in state $\widehat{\gamma^D}$. However, based on the condition predicates in (#28), none of the conditions are satisfied by $\langle u_1, r_1\rangle$, and based on the fact that the Deny policy cannot have

negated conditions, there is no possible negative authorization rule that can infer $exec\_deny_D(u_1, r_1)$ predicate. Therefore, our earlier assumption that a state-matching reduction exists is incorrect. □

Theorem 2. *There exists no state-matching reduction from the Negation scheme to the Deny scheme.*

Proof. Direct result of Claims 1, 2 and 3. □

7. Expressive Power of Negation Scheme

In this section, we show that the Negation scheme can precisely express all the authorizations enforced by the Deny scheme, i.e., the Negation scheme is at least as expressive as the Deny scheme based on state-matching reductions. We first present a mapping from the Deny scheme to the Negation scheme in Section 7.1. Then, in Section 7.2, we show that the proposed mapping satisfies the properties of a state-matching reduction.

7.1. Mapping from Deny Scheme to Negation Scheme

Let $\gamma^D = \langle \mathcal{C}_{\gamma^D}, \mathrm{P}_{\gamma^D} \rangle$ be the given state of the Deny scheme, ψ^D its state-change rule, and Q^D the set of queries. Let γ^N, ψ^N, and Q^N be the corresponding elements in the Negation scheme. Let λ be the mapping from the Deny scheme to the Negation scheme. In particular, λ maps $\langle \gamma^D, \psi^D \rangle$ to $\langle \gamma^N, \psi^N \rangle$. Also, λ maps each $q^D \in Q^D$ to $q^N \in Q^N$.

We will discuss the mapping of states and state transitions in detail in the rest of this section. We already discussed queries and their entailment in Section 3.2. Note that we consider the same set of authorization queries for the Negation scheme and the Deny scheme, i.e., $q^N = q^D$. Since the conditions are also shared between both the schemes, the queries essentially ask whether the two schemes enforce the same set of authorizations. We will use the subscripts D and N in the rules to indicate whether a rule belongs to the Deny or the Negation scheme.

We first define $\gamma^N = \langle \mathcal{C}_{\gamma^N}, \mathrm{P}_{\gamma^N} \rangle$. Since we assume the evaluations of conditions are the same for the Deny and the Negation schemes, we have:

$$\mathcal{C}_{\gamma^N} = \mathcal{C}_{\gamma^D} \tag{\#30}$$

P_{γ^N} is defined depending on the meta-policy in the Deny scheme as follows:

- In the case of DP-PO meta-policy in the Deny scheme: Corresponding to the set of all deny rules $\{\langle \phi_i, \texttt{DENY} \rangle_D\}$ in P_{γ^D}, we add the following rule:

  $$\langle \bigwedge_i \neg \phi_i, \texttt{PERMIT} \rangle_N \tag{\#31}$$

  We preserve all permit rules, i.e., for each $\langle \phi_j, \texttt{PERMIT} \rangle_D$ in P_{γ^D}, we add the following rule:

  $$\langle \phi_j, \texttt{PERMIT} \rangle_N \tag{\#32}$$

- In the case of DD-DO meta-policy in the Deny scheme: For each permit rule $\langle \phi_j, \texttt{PERMIT} \rangle_D$ in P_{γ^D}, we add the following rule that considers the set of all deny rules $\{\langle \phi_i, \texttt{DENY} \rangle_D\}$:

  $$\langle \phi_j \wedge (\bigwedge_i \neg \phi_i), \texttt{PERMIT} \rangle_N \tag{\#33}$$

In (#31), (#32), and (#33), ϕ is a condition expression comprising of a singleton condition or a conjunction of conditions evaluated on the access request subject and object. Also, it is worth noting that, in order to respect the Negation policy model in Definition 5, the abovementioned rules in the Negation scheme may need to be broken up into multiple rules when their conditions' sub-expressions are negated. Even though we do not always show such expansions in the rest of our discussions, this can be always done because every Boolean expression can be transformed into *sum of products* form. As an example of expanding into multiple rules, suppose there are two negative rules in the Deny scheme implementing DP-PO meta-policy:

$$\langle C_1 \wedge C_2, \texttt{DENY} \rangle_D \tag{\#34}$$
$$\langle C_3 \wedge C_4, \texttt{DENY} \rangle_D \tag{\#34}$$

Here, C_1, C_2, C_3 and C_4 are conditions on subjects and objects. Corresponding to the deny rules in (#34), according to (#31), we add the following rule in the Negation scheme:

$$\langle \neg(C_1 \wedge C_2) \wedge \neg(C_3 \wedge C_4), \texttt{PERMIT} \rangle_N \tag{\#35}$$

After applying the De Morgan's and the Distributive laws on the rule in (#35), considering each disjunction as an individual rule, we obtain the following authorization rules:

$$\langle \neg C_1 \wedge \neg C_3, \texttt{PERMIT} \rangle_N$$
$$\langle \neg C_1 \wedge \neg C_4, \texttt{PERMIT} \rangle_N$$
$$\langle \neg C_2 \wedge \neg C_3, \texttt{PERMIT} \rangle_N$$
$$\langle \neg C_2 \wedge \neg C_4, \texttt{PERMIT} \rangle_N \tag{\#36}$$

Instead of considering the expanded form containing four separate positive rules given in (#36), we consider only one positive rule given in (#35) to simplify the presentation.

In Appendix A, we will prove that our proposed mapping from the Deny scheme to the Negation scheme is correct, i.e., it preserves the authorizations between both the schemes. We will discuss mapping of state transitions in Section 7.2 when we discuss the state-matching reduction from the Deny scheme to the Negation scheme.

7.2. Negation is as Expressive as Deny

In this section, we show that the Negation scheme can represent all authorization policies that can be represented in the Deny scheme based on state-matching reductions (Definition 8).

Theorem 3. *There exists a state-matching reduction from the Deny scheme to the Negation scheme.*

Proof. We show that the mapping from Deny to Negation presented in Section 7.1 satisfies the two properties for it to be a state-matching reduction. We consider each assertion from Definition 8 in turn. We demonstrate that after a series of state transitions, the authorizations in the Deny scheme match the authorizations in the corresponding state of the Negation scheme.

Proving Property 1

By construction, we show that the first property of the state-matching reduction (Definition 8) is satisfied by our mapping. Let γ^D be the given state in the Deny scheme, and:

$$\gamma^D \overset{*}{\mapsto}_\psi \gamma^{D'}$$

Since, $\overset{*}{\mapsto}_\psi$ denotes transitive and reflexive closure of $\mapsto_\psi$, we have:

$$\gamma^D = \gamma_0^D \mapsto_\psi \gamma_1^D \ldots \mapsto_\psi \gamma_m^D = \gamma^{D'}$$

Let the corresponding states in the Negation scheme be:

$$\gamma^N = \gamma_0^N, \gamma_1^N, \ldots, \gamma_m^N = \gamma^{N'}$$

Suppose the authorizations in state γ^D of the Deny scheme are the same as the authorizations in the corresponding state γ^N of the

Negation scheme. Each state transition in the Deny scheme is a modification of either a condition or an authorization rule within the current protection state. In the following, we present the mapping of state transitions from the Deny scheme to the Negation scheme, by considering each kind of state transition in turn, so that $\gamma^{D'} \vdash q^D$ if and only if $\gamma^{N'} \vdash q^N$, where $q^N = \lambda(q^D)$.

Changes to system conditions. If the state transition from γ_i^D to γ_{i+1}^D is due to updating the evaluation (i.e., truth assignment) of a condition, then the same changes would be made in the Negation scheme to transition from γ_i^N to γ_{i+1}^N. This is because we assume that the evaluations of conditions are the same for the Deny and the Negation schemes. Also, as mentioned in the beginning of Section 7.1, we consider the same set of authorization queries for both the schemes.

Changes to authorization policy. Depending on the meta-policy of the Deny scheme, if the state transition γ_i^D to γ_{i+1}^D is due to addition/deletion of an authorization rule, then we perform the following changes to transition from γ_i^N to γ_{i+1}^N:

- In case of DD-DO meta-policy: Consider *addition* of the following positive authorization rule in the Deny scheme:

  $\langle \phi_j, \texttt{PERMIT} \rangle_D$

  When a new permit rule is added in the Deny scheme, a corresponding rule needs to be added in the Negation scheme considering the already existing deny rules:

  $$\langle \phi_j \wedge \Big(\bigwedge_{r \in \mathrm{P}_{\gamma_i^D}.R,\ r.decision=\texttt{DENY}} \neg r.\phi \Big), \texttt{PERMIT} \rangle_N$$

  Next, consider the addition of a negative authorization rule in the Deny scheme:

  $\langle \phi_i, \texttt{DENY} \rangle_D$

  When a new deny rule is added in the Deny scheme, we need to amend the existing rules in the Negation scheme by adding $\neg\phi_i$ to their conditions (to ensure condition ϕ_i is not permitted). That is, each rule $\langle \phi, \texttt{PERMIT} \rangle_N$ in $\mathrm{P}_{\gamma_i^N}$ is "replaced" with:

  $\langle \phi \wedge \neg\phi_i, \texttt{PERMIT} \rangle_N$

  Note that when we talk about replacing a rule, essentially we are first deleting the existing rule, and then adding the same rule but also including the conditions from the newly added negative rule. For the particular case of adding a negative rule in the Deny scheme with DD-DO meta-policy, we first delete the existing rules in the Negation scheme, one by one, and then add the same rules, again one by one, with the newly added negated condition.
- In case of DP-PO meta-policy: Consider the addition of a positive authorization rule in the Deny scheme:

  $\langle \phi_j, \texttt{PERMIT} \rangle_D$

  When a new permit rule is added in the Deny scheme, then we add the same authorization rule in the Negation scheme as well:

  $\langle \phi_j, \texttt{PERMIT} \rangle_N$

  Next, consider the addition of a negative authorization rule in the Deny scheme:

  $\langle \phi_i, \texttt{DENY} \rangle_D$

  For this, we need to amend the existing rule in the Negation scheme that consists of negated conditions by adding $\neg\phi_i$ to its condition (to ensure condition ϕ_i is not permitted). That is, the rule $\langle \phi, \texttt{PERMIT} \rangle_N$ in $\mathrm{P}_{\gamma_i^N}$, where $\phi = \bigwedge_{r \in \mathrm{P}_{\gamma_i^D}.R,\ r.decision=\texttt{DENY}} \neg r.\phi$, is replaced with:

  $\langle \phi \wedge \neg\phi_i, \texttt{PERMIT} \rangle_N$

The changes made in the Negation scheme corresponding to the state transitions caused due to the *deletion* of rules in the Deny scheme would be exactly opposite to what was carried out in the case of the addition of authorization rules. For example, if we consider the deletion of a negative authorization rule from the Deny scheme implementing DD-DO meta-policy, we need to delete that rule's condition from the conditions of the existing rules in the Negation scheme. Similarly, if we delete a positive authorization rule from the Deny scheme implementing DP-PO meta-policy, then we delete the same rule from the Negation scheme as well. Since it should be intuitive, we do not present the entire process for mapping state transitions in case of deletion of authorization rules to avoid repetition of content.

Proving Property 2

By construction, we prove that the second property of the state-matching reduction (Definition 8) is satisfied by our mapping. Note that here we consider a Negation scheme with a particular state-transition policy, that is based on the mapping of state transitions from the Deny scheme described earlier. Therefore, we are not dealing with a Negation scheme that can perform unrestricted state transitions. To further clarify this point, consider the mapping of state transitions from Deny to Negation described earlier. Corresponding to a state transition in the Deny scheme, i.e., $\gamma_i^D \mapsto_\psi \gamma_{i+1}^D$, we may need to perform multiple state transitions in the Negation scheme, i.e., $\gamma_i^N = \gamma_0^N \mapsto_{\psi'} \gamma_1^N \mapsto_{\psi'} \dots \mapsto_{\psi'} \gamma_{m-1}^N \mapsto_{\psi'} \gamma_m^N = \gamma_{i+1}^N$. Here, the states γ_i^D and γ_{i+1}^D in the Deny scheme are respectively mapped to the states γ_i^N and γ_{i+1}^N in the Negation scheme. It can be observed that as the Negation scheme simulates the Deny scheme it may enter intermediate states indicated by $\gamma_1^N, \dots, \gamma_{m-1}^N$. Using the second property of the state-matching reduction, we want to show that each of these states in the Negation scheme corresponds to some state in the Deny scheme. This is because, during simulation, we do not want to create any state in the Negation scheme that is unsafe according to the Deny scheme.

Suppose γ^D is a given Deny state and γ^N is the corresponding state in the Negation scheme. Based on our discussion in the previous paragraph, we want to ensure that for every reachable state from γ^N, there exists an equivalent state, under the mapping λ, in the Deny scheme that is reachable from γ^D. Let:

$$\gamma^N \overset{*}{\mapsto}_{\psi'} \gamma^{N'}$$

We construct the Deny state $\gamma^{D'}$ that corresponds to $\gamma^{N'}$ in the following manner. For each rule in $\mathrm{P}_{\gamma^{N'}}.R$, corresponding to *every* negated condition expression, $\neg\phi_i$, in the rule, we add a deny rule with the same condition expression, $\langle \phi_i, \texttt{DENY} \rangle_D$, in $\mathrm{P}_{\gamma^{D'}}.R$. Furthermore, corresponding to the unnegated condition expression ϕ_j in the rule, we add a permit rule with the same ϕ_j, i.e., $\langle \phi_j, \texttt{PERMIT} \rangle_D$, in $\mathrm{P}_{\gamma^{D'}}.R$.

We justify our construction of the state $\gamma^{D'}$ depending on the meta-policy used by the Deny scheme:

- In the case of DP-PO meta-policy: Based on the previously given mappings, (#31) and (#32) are the only formats of authorization rules in the "mapped" Negation scheme. Consider a permitted authorization query q^N in the Negation scheme, i.e., $\gamma^{N'} \vdash q^N$. Therefore, either a rule of the form $\langle \bigwedge_i \neg\phi_i, \texttt{PERMIT} \rangle_N$ or a rule of the form $\langle \phi_j, \texttt{PERMIT} \rangle_N$ should be applicable to the involved access request. That is, there are two means for an access request to be permitted: either none of the ϕ_i's are satisfied, or at least one of the ϕ_j's is satisfied. Since, we added a deny rule in the Deny scheme with respect to every negated

condition expression, $\neg\phi_i$, the default PERMIT policy takes care of the first means. In addition, since we added a permit rule in the Deny scheme for every ϕ_j, the applicability of authorization rules and the PERMIT-overrides policy takes care of the second means. Thus, the authorizations in the Deny scheme and the Negation scheme will be identical, and we can assert that $\gamma^{D'} \vdash q^D$ if and only if $\gamma^{N'} \vdash q^N$.

- In the case of DD-DO meta-policy: Based on the previously given mappings, the authorization rules in the "mapped" Negation scheme can only be of the form given in (#33). Therefore, an authorization rule of the form $\langle \phi_j \wedge (\bigwedge_i \neg\phi_i), \texttt{PERMIT}\rangle_N$ should be applicable to an access request, q^N, so that $\gamma^{N'} \vdash q^N$. This means that only ϕ_j's *and* none of the ϕ_i's should be satisfied. In the context of the Deny scheme, the DD-DO meta-policy requires that only the positive authorization rules, and none of the negative authorization rules, be applicable to an access request, q^D, so that $\gamma^{D'} \vdash q^D$. Since, we added a deny rule for every ϕ_i and a permit rule for every ϕ_j, the access requests that satisfy only condition expressions of the form ϕ_j, *and* none of the condition expressions of the form ϕ_i, are permitted. Thus, the authorizations in both the schemes will be identical, and we can assert that $\gamma^{D'} \vdash q^D$ if and only if $\gamma^{N'} \vdash q^N$.

Note that, in the Negation scheme, addition/deletion of a rule implicitly consists of addition/deletion of multiple rules. This is because, as mentioned in Section 7.1, transforming a condition into sum of products form may lead to multiple rules. We skip presenting those details. But, it is imperative that the state transitions involving adding/deleting rules in the sum of products form be performed in such a way that the format of each reachable state $\gamma^{N'}$ in the Negation scheme follows the format given in the previously presented mappings. □

8. Related Work

8.1. Negated Condition and Negative Authorization in Access Control Models

Many access control models have considered the notions of negated conditions and/or negative authorizations. In the following, we survey a representative set of such proposals in the literature. Table 1 summarizes our survey results in terms of access control model and features, support for negated conditions, support for negative authorization, and practical applications discussed in each work. We show several examples of the notions of negated conditions and negative authorization from these works that correspond to what we captured in our Negation model (Definition 5) and Deny model (Definition 6), respectively.

Access Matrix. One of the earliest and most influential models, the *access control matrix model* (Graham and Denning, 1971; Harrison et al., 1976; Lampson, 1974), represents authorizations in the form of a matrix based on subjects (represented as rows), objects (indicated as columns), and rights/privileges that subjects can exercise over objects (represented in cells). The access matrix can be modified only if the specified conditions about current system authorizations evaluate to true. But, the conditions check only for the *presence* of rights in the matrix cells. A variant of the access matrix model, called the *augmented typed access matrix (ATAM)* (Ammann et al., 1992), allows conditions to check for the presence as well as the *absence* of a right in an access matrix cell. This is similar to the optional negation operator associated with conditions within an authorization rule in our Negation model. The following is an example of ATAM command, which transfers the ownership of a file F from one user U to another user V in the matrix M:

command *transfer-ownership*(U:*user*, V:*user*, F:*file*)
  if *own* $\in M[U, F]$ **and** *own* $\notin M[V, F]$
  then
    delete *own* **from** $M[U, F]$;
    enter *own* **into** $M[V, F]$;
end

Role-Based Access Control (RBAC). In the RBAC policy model, access permissions are assigned based on user roles (Sandhu, Coyne, Feinstein, Youman, 1996). The traditional RBAC, however, supports neither negation nor negative authorization. The *generalized role-based access control (GRBAC)* (Moyer, Abamad, 2001) extends traditional RBAC by incorporating subject roles, object roles and environment roles. A policy rule in GRBAC indicates a subject role, object role, and environment role. There is also an indicator for whether the rule is allowing or prohibiting. Thus, it supports negative authorizations. In a subsequent work, Al-Kahtani and Sandhu (2004) investigated different aspects of negative authorization in *rule-based RBAC (RB-RBAC)* family of models such as user authorization, conflict among rules and conflict resolution polices. In the RB-RBAC family, negative authorization is represented by *negative roles*. For example, in a military unit that has a Commander and four staff officers G_1, G_2, G_3, G_4, the requirement that "The Commander cannot delegate his role to a staff officer whose rank is lower than a Lt. Colonel" can be represented in the RB-RBAC system as (Al-Kahtani, Sandhu, 2004):

Attributes in the System :
  a_1 : *rank-type* = *officer*
  a_2 : *Staff course* = T,
  a_3 : *Leadership course* = T
  a_4 : *Rank* $\geq$ *Lt. Colonel*,
  a_5 : *Assignment Order* = T
Authorization Rules :
  $a_1 \wedge a_2 \Rightarrow \{G_1, G_2, G_3, G_4\}$
  $a_1 \wedge a_2 \wedge a_3 \wedge a_4 \wedge a_5 \Rightarrow$ Commander
  $\neg a_4 \Rightarrow \neg$ Commander
can_delegate(Commander, G_i, *duration*, *start-time*)

Also, negated conditions can be specified in RB-RBAC using negation in attribute expressions.

Trust Management. Trust management is a form of distributed access control that allows a principal to delegate some access decisions to other principals over one or more resources. The family of *role-based trust-management (RT)* languages (Li et al., 2002; Li and Mitchell, 2003) combines the strengths of RBAC and previous trust management systems (Blaze et al., 1999; Clarke et al., 2001; Ellison et al., 1999; Rivest and Lampson, 1996). An access control permission is represented as a role, where role membership is defined using different types of *policy statements*, also called *credentials* or *certificates*. For instance, the *simple inclusion* statement $A.r \leftarrow B.r_1$ means that every member of B's r_1 role is also a member of A's r role, thereby causing a delegation of access decisions from principal A to principal B. However, similar to RBAC, the standard RT language does not support specifying negation or negative authorization. Adding a policy statement to the system can only increase trust, and in turn, granting additional privileges. Czenko et al. (2006) proposed an extension to the standard RT language, called $RT_{\ominus}$, that includes a restricted form of negation, and demonstrated its usefulness in virtual communities such as P2P file sharing systems. Their extended language enables more expressive policies based on inclusion/exclusion of target communities rather than individual members. For example, a policy stating "A community coordinator distrusts any entity whom she does not accept locally" can be specified in $RT_{\ominus}$ as follows (Czenko, Tran, Doumen,

Table 1
Access control models employing negated condition and negative authorization. Some works use alternative terminologies for negated condition and negative authorization (deny rules), which are indicated in parentheses.

Work	Applications	Access Control Model	Features	Negated Condition	Negative Authorization
ATAM (Ammann et al., 1992)		Access matrix	Checking absence of rights	✓(absence of a right)	
GRBAC (Moyer, Abamad, 2001)	Company	RBAC	Subject, object, environment roles		✓
RB-RBAC (Al-Kahtani, Sandhu, 2004)	Military unit	RBAC		✓	✓(negative roles)
$RT_{\ominus}$ (Czenko, Tran, Doumen, Etalle, Hartel, den Hartog, 2006)	P2P applications	Trust Mgt.		✓	
REFEREE (Chu, Feigenbaum, LaMacchia, Resnick, Strauss, 1997)	Web applications	Trust Mgt.	Logic programs		✓
TPL (Herzberg, Mass, Mihaeli, Naor, Ravid, 2000)	E-business, healthcare	Trust Mgt.	Mapping from certificates to roles	✓(negative certificates)	
Shinren (Dong, Dulay, 2010)	E-market, healthcare	Trust Mgt.	Logic programs	✓	✓(distrust policy)
(Park, Sandhu, 2004)	Digital resources (e.g. e-book) usage	UCON	Authorizations, obligations, and conditions	✓	∼ [a]
WS-ABAC (Shen, Hong, 2006)	Web services	ABAC		✓	
$ABAC_{\alpha}$ (Jin, Krishnan, Sandhu, 2012)		ABAC		✓	
HGABAC (Servos and Osborn, 2014)	University library	ABAC	Attribute inheritance using groups	✓	
RAdAC (Kandala, Sandhu, Bhamidipati, 2011)	Classified documents	ABAC	Risk as a decision factor		✓
ABMAC (Lang, Foster, Siebenlist, Ananthakrishnan, Freeman, 2009)	Grid computing	ABAC	Combining heterogeneous policies		✓
XACML (eXt, 2013)		ABAC	Combining policies and rules	✓(negative rules)	✓
PTaCL (Crampton, Morisset, 2012)	Company confidential resources	ABAC	Comprehensive policy target and composition spec.	✓	✓
(Carminati et al., 2009, 2011)	Online social networks	ReBAC	Semantic web		✓(filtering policies)
OSNAC (Masoumzadeh, Joshi, 2010)	Online social networks	ReBAC	Semantic web		✓
(Hu, Ahn, 2011)	Online social networks	ReBAC	Multiparty authorization		✓
(Bruns et al., 2012; Fong, 2011; Fong and Siahaan, 2011)	Online social networks, healthcare	ReBAC	Uses modal logic, hybrid logic	✓	
(Cheng et al., 2012a,b, 2014)	Online social networks	ReBAC	Regular expressions	✓	
(Crampton, Sellwood, 2014)	Corporate project management	ReBAC	Path conditions & principal matching		✓
RPPM (Crampton, Sellwood, 2015)	University	ReBAC		✓(forbidden path)	✓
RPPM2 (Stoller, 2015)	Healthcare	ReBAC	Administrative model		✓
ARPPM (Crampton, Sellwood, 2016)	University	ReBAC	Administrative model	✓	✓
(Pasarella, Lobo, 2017)	Online social networks	ReBAC	Datalog programs	✓	✓
(Bertino et al., 1997, 1999)	Relational data management systems	Rule-based DAC			✓
(Gal-Oz, Gudes, Fernández, 1993)	University	Group-based	Object methods access in object-oriented databases		✓

[a] UCON can trigger a *stopped procedure* to revoke usage rights. This can be considered to be similar to negative authorization, although a *stopped procedure* deals with *ongoing-authorizations* rather than *pre-authorizations*.

Etalle, Hartel, den Hartog, 2006):

`A.disagreeToAdd` ← `A.allCandidates` ⊖ `A.agreeToAdd`

Other trust management proposals that incorporate statements which can decrease privileges, thereby supporting specification of mutual exclusion policies like separation of duty, include REFEREE (Chu, Feigenbaum, LaMacchia, Resnick, Strauss, 1997), TPL (Herzberg, Mass, Mihaeli, Naor, Ravid, 2000), and Shinren (Dong, Dulay, 2010). In the REFEREE trust management system for web applications (Chu, Feigenbaum, LaMacchia, Resnick, Strauss, 1997), the evaluation of policies as well as credentials is put under policy control to mitigate the risks associated with executing arbitrary programs as part of the authorization decision (policies and credentials in REFEREE contain executable programs). The REFEREE supports negative authorization since a requested action can be permitted/denied based on the presence of sufficient credentials, and resolves conflicting information from multiple sources. However, REFEREE has no support for negative certificates (negated conditions in the context of this paper). TPL (Herzberg, Mass, Mihaeli, Naor, Ravid, 2000) specifies the mapping of unknown users to predefined business roles, based on public key certificates issued by third parties. Then, RBAC is utilized to determine the access control decisions based on the assigned roles. A disjunctive set of rules for each role determines how a certificate holder can become a member in the role. Each rule defines a set of necessary certificates for the role membership. The certificates can be either positive or negative depending on whether the certificate should or should not exist for a rule to hold, which correspond to positive and negated conditions in our Negation model. For example, in an electronic marketplace to become a recognized e-vendor, a role membership rule stating "A recognized e-vendor should have a recommendation certificate and no warning certifi-

cate from any other recognized e-vendor" can be expressed in TPL (denoted in XML) as follows:

```
<POLICY>
  <GROUP NAME="E-Vendors">
    <RULE>
      <INCLUSION ID="reco" TYPE="Recommendation"
      FROM="E-Vendors"></INCLUSION>
      <EXCLUSION ID="warn" TYPE="Warning"
      FROM="E-Vendors"></EXCLUSION>
      <FUNCTION></FUNCTION>
    </RULE>
  </GROUP>
</POLICY>
```

The Shinren (Dong, Dulay, 2010) trust management system considers negative information and supports reasoning with incomplete, uncertain, and inconsistent information, such as the information collected from an electronic marketplace. The syntax of the Shinren policy language is based on an extension of Datalog and supports negated conditions. Besides, the Shinren trust policy language allows policy makers to define trust as well as distrust policies, and employs prioritization to resolve conflicts between trust and distrust policies. The trust policies, distrust policies and prioritization, respectively, correspond to positive authorization rules, negative authorization rules, and conflict resolution strategy component in our context.

Usage Control (UCON). Park and Sandhu (2004) proposed the family of UCON_{ABC} models for usage control, which is a generalization of access control that integrates authorizations, obligations, conditions, ongoing controls, and mutable subject and object attributes. UCON supports the concept of *ongoing-authorizations* in which authorization decisions are made continuously while usage rights are exercised. If certain attributes are changed, causing certain requirements to become dissatisfied, a *stopped procedure* can be performed to revoke the currently allowed usage right. The *stopped procedure* can be considered to be similar to the negative authorization; however, it deals with ongoing-authorizations rather than *pre-authorization* decision-making as in other models. UCON also supports negated conditions.

Attribute-Based Access Control (ABAC). In ABAC, authorization policies are specified in terms of attributes of users and resources. Shen and Hong (2006) present an attribute-based access control model to address web services security, called WS-ABAC, which grants access to web services based on attributes of the related entities. Further, Jin et al. (2012) develop a formal ABAC model, which they call ABAC_α, that has just sufficient features to be configured to do DAC, MAC and RBAC. Additionally, Servos and Osborn (2014) propose the *hierarchical group and attribute-based access control (HGABAC)* that allows attribute inheritance through user and object groups, and also includes environment, connection and administrative attributes. In all of these ABAC models, specification of only granted permissions is considered, but the policy language supports negated conditions. For example, in a university library, a policy stating "Undergraduate students may check out any unrestricted book and any course materials for a course in which they are enrolled" can be represented in the HGABAC policy language as (Servos and Osborn, 2014):

```
{{" "undergrad" IN user.user_type AND ((object.
object_type = "book" AND NOT object.restricted)
OR (object.object_type = "course" AND user.
enrolled_in IN object.req_course))",
check_out_book}}
```

Researchers have proposed a formal framework to develop abstract models for *risk-based adaptive access control (RAdAC)* (Kandala, Sandhu, Bhamidipati, 2011), employing attribute-based policy specification. In addition, there has been work on developing a flexible access control mechanism for Grid computing, namely *attribute-based multipolicy access control (ABMAC)* model, that is based on ABAC and supports multiple security policies (Lang, Foster, Siebenlist, Ananthakrishnan, Freeman, 2009). A policy in ABMAC consists of sub-policies and not rules, so a policy is the unit of decision-making in their model. Both of these works support specification of explicit negative authorizations, and utilize a combining algorithm, similar to conflict resolution strategy in our context, to determine a final access decision.

The eXtensible Access Control Markup Language, or commonly referred to as XACML (eXt, 2013), is an OASIS standard that describes an XML-based access control policy language for specifying attribute-based policies and an access control request/response language for policy enforcement. In XACML, decision-making is supported at rule-level as well as policy-level, and involves rule-combining and policy-combining algorithms for combining decisions from multiple rules and policies, respectively. The rule-combining algorithm in XACML corresponds to the conflict resolution strategy component in our Deny policy. Crampton and Morisset (2012) propose a language, which they call PTaCL, based on two sub-languages, namely the policy target language (PTL) for determining when a policy should be evaluated and the policy composition language (PCL) for specifying complex policies. In their language, *atomic policies* correspond to access control rules in XACML, whereas *policy trees* correspond to policies and policy sets. Both XACML and PTaCL support negated conditions as well as negative authorizations. In XACML, negated conditions are referred as *negative rules*.

Relationship-Based Access Control (ReBAC). In ReBAC, policy rules are specified based on relationships among users and resources captured in the system graph. Carminati et al. (2009, 2011) proposed an access control framework which employs the *semantic web rule language (SWRL)* to define authorization, administration and filtering policies. Another semantic web-based approach allows both users and the system to express policies based on access control ontologies (Masoumzadeh, Joshi, 2010). In addition, Hu and Ahn (2011) proposed a formal model to address multi-party access control in online social networks (OSNs) with a policy conflict resolution mechanism based on voting. These works support specification of negative authorization; however their policy language does not support negated conditions.

Fong (2011) proposed a ReBAC policy language for social computing applications based on modal logic. The policy language was later extended and improved with more expressive power (Fong, Siahaan, 2011). In a subsequent work, hybrid logic was used for expressing ReBAC policies to achieve greater flexibility and better efficiency in policy specification (Bruns, Fong, Siahaan, Huth, 2012). These modal and hybrid logic-based policy languages support negated condition, but the rules can specify only the granted permissions. An example of a ReBAC rule using negated condition, "Grant access unless the accessor is a family of the owner", can be specified using modal logic as $\neg \langle \texttt{family} \rangle \, \texttt{a}$. Cheng et al. (2012a, 2012b, 2014) proposed a series of ReBAC models for OSNs that use regular expressions to specify policies. Initially, the authors considered only relationships among users (Cheng, Park, Sandhu, 2012). They later extended their model to incorporate relationships among users and resources (Cheng, Park, Sandhu, 2012) and attribute information of users and relationships for the purpose of access control (Cheng, Park, Sandhu, 2014). Their policy language supports negated condition (i.e., checking absence of the specified relationship sequence), but does not support negative authorization. For

instance, the authorization rule "Grant access to the *event post* if the subject is not an immediate friend and is connected through users whose *occupation* is *student*" can be represented in their attribute-aware policy language as:

$\langle read, EventPost, (u_a, (\neg(f, 1) : \forall[+1, -1], occupation(u)=\text{'student'}, \_))\rangle$

Crampton and Sellwood (2014, 2015) proposed ReBAC models for general-purpose computing systems that introduced the concept of authorization principals which are analogous to roles in RBAC. The policies are specified in terms of path conditions which are similar to regular expressions. Their initial model (Crampton, Sellwood, 2014) does not support negation, but can support positive as well as negative authorization rules. For example, in a corporate project management scenario, the requirement that "Grant write access to all resources (i.e., files, folders, printers) that are allocated to the accessor's team" can be specified in their model as given below (Crampton, Sellwood, 2014):

Principal-Matching Rule: (`Member-of`; $\overline{\texttt{Resource-for}}$, `Team-Resource-User`)

Authorization Rule: (`Team-Resource-User`, $\star$, `write`, 1)

The model succeeding the above work, called RPPM (Crampton, Sellwood, 2015), supports both negated conditions, in terms of *forbidden paths*, and deny authorization rules. For example, in a university, the requirement that "Grant the grading right to all coursework answers if the accessor is a teaching-assistant but not the enrolled-student of the course" can be expressed in the RPPM model through the specification of the following rules (Crampton, Sellwood, 2015):

Principal-Matching Rule:

(`is-ta-for`; $\overline{\texttt{is-coursework-for}}$, `is-enrolled-on`; $\overline{\texttt{is-coursework-for}}$, `course-ta`)

Authorization Rule: (`course-ta`, $\star$, `grade`, 1)

There have been proposals in the literature that define administrative ReBAC models for regulating changes to all aspects of the ReBAC policy in general computing systems (Crampton and Sellwood, 2016; Stoller, 2015). The authorization policy in the RPPM2 model (Stoller, 2015) supports only negative authorizations, whereas that in the ARPPM model (Crampton, Sellwood, 2016) also supports negated conditions in terms of positive and negative targets.

More recently, Pasarella and Lobo (2017) presented a very general framework to specify and implement ReBAC policies by introducing a class of Datalog programs suitable for modeling ReBAC, which they refer to as ReBAC Datalog policies. Their work supports both negated conditions and negative authorizations. For example, the requirement that "Friends of a user who are not family are allowed to view party posts" can be specified in their Datalog framework using the following rules, considering the DENY-overrides conflict resolution strategy:

$rel(O, owns, Res), rel(O, friend, Req), prop(Res, partyPost) \rightarrow grant(Req, Res)$

$rel(O, owns, Res), rel(O, family, Req), prop(Res, partyPost) \rightarrow deny(Req, Res)$

The same authorizations can be specified using negated condition in their framework as follows:

$rel(O, owns, Res), rel(O, friend, Req), \neg rel(O, family, Req), prop(Res, partyPost) \rightarrow grant(Req, Res)$

Database Management Systems (DBMS). The support for negative authorization has also been explored in the domain of DBMS. Bertino et al. (1999b, 1997) presented an authorization model for relational DBMS that, along with positive authorizations, also considers negative authorizations for specifying explicit denial for a user to access an object (i.e., a table). For example, the rule ⟨`Alice,select,-,Reports,Bob,strong`⟩ states that `Alice` cannot select records from the `Reports` table. It also indicates that this negative authorization has been specified by `Bob` and cannot be overridden. In addition, Gal-Oz et al. (1993) have presented an authorization model for object-oriented databases that controls access to objects through methods to conform with the concept of encapsulation (i.e., access to the data in an object should only be performed through a method). Their model also provides support for negative authorization.

Expressiveness Analysis. As demonstrated above, many access control proposals in the literature have used negated conditions and negative authorizations (deny rules) as part of their policy language in order to provide flexible specification. However, they have not concretely analyzed them in terms of expressiveness. Our work provides a formal comparison of the expressive power of models that utilize negated conditions with those that utilize negative authorizations.

8.2. Access Control Security Analysis & Expressiveness Comparison

Early works on the analysis of access control policies focused on the *safety problem* (Harrison et al., 1976; Lipton and Snyder, 1977; Sandhu, 1988, 1992), where the objective is to determine if there exists a reachable state in which a presumably untrusted subject has access to a certain object. Li et al. (2005) introduced *security analysis* as a generalization of the policy analysis problems to verify whether other desired security properties, such as availability, are maintained across state transitions in an access control system. Security analysis was initially demonstrated for the RT policy languages (Li et al., 2002; Li and Mitchell, 2003). It has since been extended to the context of other models such as RBAC (Li, Tripunitara, 2006), ABAC (Jha, Sural, Atluri, Vaidya, 2019), and ReBAC (Masoumzadeh, 2018).

Tripunitara and Li (2007) present a theory for comparing the expressive power of access control models based on two notions of simulations called state-matching reductions and reductions. Their concept is based on simulations that preserve security properties such as availability, mutual exclusion, and bounded safety. According to their theory, when one scheme can represent all types of policies that another can, then the former is deemed to be at least as expressive as the latter. That is, using state-matching reduction, they reduce the *compositional security analysis* problem in one scheme to that in another scheme. Compositional security analysis generalizes security analysis to consider logical combinations of queries in security analysis. A reduction is used to reduce the security analysis problem in one scheme to that in another scheme. We adopt the notion of access control scheme (Definition 7) from their work to specify a set of states and state-transition policies in the proposed Negation and Deny models. In addition, we adopt the notion of state-matching reduction (Definition 8) based on which we compare the expressive power of the Negation model, i.e., policies that use negated conditions, and the Deny model, i.e., policies that use negative authorization.

9. Discussions and Conclusion

As demonstrated in our literature review in Section 8, the concept of "negation" has been widely used in access control policy models to provide flexible authorization. In particular, we identify two forms of this concept: negated conditions and negative authorizations. Our survey shows that many models support either of those, and some models even provide both. Therefore, the choice of supporting either one seems arbitrary. Our key contribution in

this paper is to analyze this choice formally. In particular, we design abstract rule-based models that support negated conditions (i.e., the Negation model) or negative authorization (i.e., the Deny model). Then, in the context of a formal access control expressiveness analysis framework (Tripunitara, Li, 2007), we compare the two proposed models and formally prove our results. The abstract design of the models ensures the intuitive generalizability of our results.

The key result of this work is that *Negation is more expressive than Deny*. In particular, we prove that Negation is at least as expressive as Deny *and* that Deny is not as expressive as Negation. This means that models which support negated conditions can safely express policies that one might express using negative authorizations (assuming all other required concepts unrelated to negation are supported). It also means that supporting both negated conditions and negative authorization in the same model is redundant in terms of expressive power. We hope that this formal result sheds light on the usage of negation concepts in the design and implementation of access control models.

While the expressiveness advantage of the negated conditions over negative authorization is important, it is also important to emphasize that expressive power is not the only deciding factor. It is important that access control administrators can easily maintain and manage policies. For example, it might seem intuitive that when dealing with a complex policy with many rules, negative authorizations (deny rules) would be easier to comprehend and reason about by human administrators than rules with negated conditions. Further scientific studies of human factors in policy administration and how they may affect the design of policy models seem necessary. It is also interesting to theoretically characterize the space of access control policies that are representable using various Deny models.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Padmavathi Iyer: Conceptualization, Methodology, Formal analysis, Writing – original draft. **Amirreza Masoumzadeh:** Conceptualization, Methodology, Formal analysis, Writing – review & editing. **Paliath Narendran:** Conceptualization, Methodology, Formal analysis, Writing – review & editing.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and helpful suggestions that guided us in improving the final manuscript.

Appendix A. Correctness of Mapping from Deny to Negation

We prove that the mapping from the Deny scheme to the Negation scheme, presented in Section 7.1, is correct. In other words, the Negation scheme preserves the authorizations enforced by the Deny scheme during state transitions. We show that our proposed mapping does not lead to *over-assignment* or *under-assignment* of permissions during the mapping from the Deny scheme to the Negation scheme. In the *over-assignment* scenario, an access request denied in the Deny scheme is permitted in the mapped Negation scheme. In the *under-assignment* scenario, an access request permitted in the Deny scheme is denied in the mapped Negation scheme.

In order to demonstrate the correctness of our proposed mapping, we employ the strategy of *proof by contradiction*. Specifically, to show that there are no over-assignments or under-assignments, we prove that the claim that an access request was permitted in the Deny scheme but was incorrectly denied in the Negation scheme leads to a contradiction, and vice-versa. Furthermore, we provide separate proofs for the cases of DP-PO and DD-DO meta-policies.

A1. Case of DP-PO Meta-Policy in the Deny Scheme

The mapping of access control rules from the Deny scheme implementing DP-PO meta-policy to the Negation scheme are specified in (#31) and (#32).

We first discuss the policy in the Deny scheme. We consider the following positive authorization rule represented in logic format:

$$exec\_permit_D(?u, ?r) \text{ :- } C_i(?u, ?r), \ldots, C_j(?u, ?r). \quad \text{(\#A.1)}$$

The conditional statements associated with negative authorization rules are as follows:

$$\begin{aligned} exec\_deny_D(?u, ?r) \text{ :- } & C_n(?u, ?r), \ldots, C_p(?u, ?r). \\ & \ldots \\ exec\_deny_D(?u, ?r) \text{ :- } & C_q(?u, ?r), \ldots, C_r(?u, ?r). \end{aligned} \quad \text{(\#A.2)}$$

The meta-policy and the access decision rules in the DP-PO Deny scheme would be as follows:

$$permit_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad \text{(\#A.3a)}$$

$$permit_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \quad \text{(\#A.3b)}$$

$$permit_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad \text{(\#A.3c)}$$

$$deny_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \quad \text{(\#A.3d)}$$

In the above conditional statements, according to the DP-PO policy, we assigned the predicate *permit* to both default decision, δ, given in (#3a) and conflict resolution strategy, ρ, given in (#3b).

Given the policy in the Deny scheme, we now discuss the mapping of rules from the Deny scheme to the Negation scheme. Corresponding to the positive rule in (#A.1), based on the mapping given in (#32), we have the same rule in the Negation scheme represented in logic format as follows:

$$exec\_permit_N(?u, ?r) \text{ :- } C_i(?u, ?r), \ldots, C_j(?u, ?r). \quad \text{(\#A.4)}$$

Corresponding to the negative rules in (#A.2), based on the mapping given in (#31), we have the following conditional statement in the Negation scheme:

$$\begin{aligned} exec\_permit_N(?u, ?r) \text{ :- } & (C_n(?u, ?r), \ldots, C_p(?u, ?r)), \ldots, \\ & \neg(C_q(?u, ?r), \ldots, C_r(?u, ?r)). \end{aligned} \quad \text{(\#A.5)}$$

We note that, in the Negation scheme, the conditional statements for determining the final authorization decisions for access requests, indicated by the *permit* and *deny* predicates, were shown previously in (#5a) and (#5b).

A1.1. Under-Assignment: Permitted in Deny but Denied in Negation

Suppose that we have under-assignment. Therefore, there is an access request $\langle u_1, r_1\rangle$ that is permitted in the Deny scheme, but denied in the Negation scheme:

$$permit_D(u_1, r_1). \tag{\#A.6a}$$

$$deny_N(u_1, r_1). \tag{\#A.6b}$$

In the context of the Negation scheme, the above DENY authorization can be only obtained through default decision (see (#5a)). Therefore, based on (#5a) and (#A.6b), we infer:

$$\neg exec\_permit_N(u_1, r_1). \tag{\#A.7}$$

In the context of the DP-PO Deny scheme, there are two ways to obtain the PERMIT authorization in (#A.6a). One way is through default decision (see (#A.3a)). The other way is through a combination of conflict resolution strategy (see (#A.3b)) and PERMIT access decision rule (see (#A.3c)). In the following, we consider each of these possibilities in turn, and prove that our proposed mapping is correct using a contradiction strategy.

Obtaining PERMIT Authorization through Default Decision Suppose that (#A.6a) is obtained through the default decision statement given in (#A.3a). Based on (#A.3a), we infer:

$$\neg exec\_permit_D(u_1, r_1). \tag{\#A.8a}$$

$$\neg exec\_deny_D(u_1, r_1). \tag{\#A.8b}$$

Based on (#A.8b), none of the negative authorization rules in the Deny scheme can be applicable to $\langle u_1, r_1\rangle$. Therefore, all conditional statements associated with negative rules (see (#A.2)) are unsatisfied:

$$\begin{aligned}&\neg(C_n(u_1, r_1), \cdots, C_p(u_1, r_1)).\\ &\qquad\cdots\\ &\neg(C_q(u_1, r_1), \cdots, C_r(u_1, r_1)).\end{aligned} \tag{\#A.9}$$

Since the conditional evaluation is the same in the Deny and the Negation schemes, (#A.9) applies to both schemes. Based on (#A.5) and (#A.9), we infer the following predicate about the authorization of $\langle u_1, r_1\rangle$ in the Negation scheme:

$$exec\_permit_N(u_1, r_1). \tag{\#A.10}$$

However, the two predicates in (#A.10) and (#A.7) contradict each other. Therefore, we conclude that (#A.6a) could not have been obtained through default decision.

Obtaining PERMIT Authorization through Conflict Resolution and Access Decision Suppose (#A.6a) is obtained through a combination of conflict resolution strategy (see (#A.3b)) and access decision rule (see (#A.3c)). From both these conditional statements, we infer the following predicate about the authorization of $\langle u_1, r_1\rangle$ in the Deny scheme:

$$exec\_permit_D(u_1, r_1). \tag{\#A.11}$$

In order to infer the predicate given in (#A.11), the positive authorization rule in (#A.1) should be applicable to $\langle u_1, r_1\rangle$ in the Deny scheme. So, we infer the following clause regarding satisfaction of conditions by $\langle u_1, r_1\rangle$:

$$C_i(u_1, r_1), \ldots, C_j(u_1, r_1). \tag{\#A.12}$$

Since the conditions are shared between the Negation and the Deny schemes, the clause given in (#A.12) should be true in the Negation scheme as well. Therefore, based on (#A.12) and the authorization rule in (#A.4), we infer the following predicate regarding the authorization of $\langle u_1, r_1\rangle$ in the Negation scheme:

$$exec\_permit_N(u_1, r_1). \tag{\#A.13}$$

However, the two predicates in (#A.13) and (#A.7) contradict each other. Therefore, (#A.6a) could not have been obtained through conflict resolution and access decision. Considering this and our earlier result, that (#A.6a) could not have been obtained through default decision, we conclude that the assumption stated in (#A.6) cannot be satisfied.

A1.2. Over-Assignment: Denied in Deny but Permitted in Negation

Suppose that we have over-assignment. Thus, there is an access request $\langle u_2, r_2\rangle$ that is denied in the Deny scheme but is permitted in the Negation scheme:

$$deny_D(u_2, r_2). \tag{\#A.14a}$$

$$permit_N(u_2, r_2). \tag{\#A.14b}$$

In context of the Negation scheme, the above PERMIT authorization can be only obtained through the access decision statement in (#5b). Therefore, based on (#5b) and (#A.14b), we infer:

$$exec\_permit_N(u_2, r_2). \tag{\#A.15}$$

In the context of the DP-PO Deny scheme, the DENY authorization in (#A.14a) can be only obtained through the access decision statement given in (#A.3d). Therefore, based on (#A.3d), we infer:

$$\neg exec\_permit_D(u_2, r_2). \tag{\#A.16a}$$

$$exec\_deny_D(u_2, r_2). \tag{\#A.16b}$$

Based on (#A.16a), none of the positive authorization rules in the Deny scheme can be applicable to $\langle u_2, r_2\rangle$. Therefore, all conditional statements associated with positive rules (see (#A.1)) are unsatisfied:

$$\neg \quad (C_i(u_2, r_2), \ldots, C_j(u_2, r_2)). \tag{\#A.17}$$

In addition, based on (#A.16b), at least one negative authorization rule in the Deny scheme should be applicable to $\langle u_2, r_2\rangle$. Without loss of generality, we consider that the first negative rule in (#A.2) is applicable to $\langle u_2, r_2\rangle$. Thus, we infer the following clause regarding the satisfaction of conditions by $\langle u_2, r_2\rangle$:

$$C_n(u_2, r_2), \ldots, C_p(u_2, r_2). \tag{\#A.18}$$

Since the conditional evaluation is the same in the Deny and the Negation schemes, (#A.17) and (#A.18) apply to both schemes. Based on (#A.17) and (#A.18), we can observe that none of the authorization rules in the Negation scheme, given in (#A.4) and (#A.5), is applicable to $\langle u_2, r_2\rangle$. So, we infer the following predicate regarding the authorization of $\langle u_2, r_2\rangle$ in the Negation scheme:

$$\neg exec\_permit_N(u_2, r_2). \tag{\#A.19}$$

However, the two predicates in (#A.19) and (#A.15) contradict each other. Therefore, we conclude that the assumption stated in (#A.14) cannot be satisfied.

A2. Case of DD-DO Meta-Policy in the Deny Scheme

The mapping of access control rules from the Deny scheme implementing DD-DO meta-policy to the Negation scheme is specified in (#33).

We first discuss the policy in the Deny scheme. The conditional statements representing the positive and negative authorization rules are given in (#A.1) and (#A.2), respectively. We specialize the conditional statements of meta-policy given in (#3a) and (#3b) according to the DD-DO policy, by instantiating δ and ρ to

the *deny* predicate. Thus, the meta-policy and the access decision rules are as follows:

$$deny_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad \text{(\#A.20a)}$$

$$deny_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \quad \text{(\#A.20b)}$$

$$permit_D(?u, ?r) \text{ :- } exec\_permit_D(?u, ?r), \neg exec\_deny_D(?u, ?r). \quad \text{(\#A.20c)}$$

$$deny_D(?u, ?r) \text{ :- } \neg exec\_permit_D(?u, ?r), exec\_deny_D(?u, ?r). \quad \text{(\#A.20d)}$$

For the given policy in the Deny scheme, the conditional statements representing the "mapped" authorization rules in the Negation scheme, according to (#33), are as follows:

$$exec\_permit_N(?u, ?r) \text{ :- } (C_i(?u, ?r), \ldots, C_j(?u, ?r)), \neg(C_n(?u, ?r), \ldots, C_p(?u, ?r)), \ldots, \neg(C_q(?u, ?r), \ldots, C_r(?u, ?r)). \quad \text{(\#A.21)}$$

We note that the conditional statements corresponding to the meta-policy and the access decision rules in the Negation scheme were shown earlier in (#5a) and (#5b).

A2.1. Under-Assignment: Permitted in Deny but Denied in Negation

Suppose that we have under-assignment. Therefore, there is an access request $\langle u_3, r_3\rangle$ that is permitted in the Deny scheme but is denied in the Negation scheme:

$$permit_D(u_3, r_3). \quad \text{(\#A.22a)}$$

$$deny_N(u_3, r_3). \quad \text{(\#A.22b)}$$

In the Negation scheme, the above DENY authorization can be only obtained through default decision given in (#5a). Thus, based on (#5a) and (#A.22b), we infer:

$$\neg exec\_permit_N(u_3, r_3). \quad \text{(\#A.23)}$$

In the DD-DO Deny scheme, the PERMIT authorization in (#A.22a) can be only obtained through the access decision statement given in (#A.20c). Based on (#A.20c), we infer:

$$exec\_permit_D(u_3, r_3). \quad \text{(\#A.24a)}$$

$$\neg exec\_deny_D(u_3, r_3). \quad \text{(\#A.24b)}$$

In order to infer the predicate given in (#A.24a), the positive authorization rule in (#A.1) should be applicable to $\langle u_3, r_3\rangle$ in the Deny scheme. So, we infer the following clause regarding satisfaction of conditions by $\langle u_3, r_3\rangle$:

$$(C_i(u_3, r_3), \ldots, C_j(u_3, r_3)). \quad \text{(\#A.25)}$$

In addition, based on (#A.24b), none of the negative rules in the Deny scheme can be applicable to $\langle u_3, r_3\rangle$. Therefore, all conditional statements associated with negative rules (see (#A.2)) are unsatisfied:

$$\neg(C_n(u_3, r_3), \cdots, C_p(u_3, r_3)). \quad \ldots \quad \neg(C_q(u_3, r_3), \cdots, C_r(u_3, r_3)). \quad \text{(\#A.26)}$$

Since the conditional evaluation is the same in the Negation and the Deny schemes, (#A.25) and (#A.26) apply to both schemes. Based on (#A.25) and (#A.26), we can observe that the authorization rule given in (#A.21) becomes applicable to $\langle u_3, r_3\rangle$. Therefore, we infer the following predicate about the authorization of $\langle u_3, r_3\rangle$ in the Negation scheme:

$$exec\_permit_N(u_3, r_3). \quad \text{(\#A.27)}$$

However, the two predicates in (#A.23) and (#A.27) contradict each other. Therefore, we can conclude that the assumption stated in (#A.22) cannot be satisfied.

A2.2. Over-Assignment: Denied in Deny but Permitted in Negation

Suppose that we have over-assignment. Thus, there is an access request $\langle u_4, r_4\rangle$ that is denied in the Deny scheme but is permitted in the Negation scheme:

$$deny_D(u_4, r_4). \quad \text{(\#A.28a)}$$

$$permit_N(u_4, r_4). \quad \text{(\#A.28b)}$$

In context of the Negation scheme, the above PERMIT authorization can be only obtained through the access decision statement given in (#5b). Therefore, based on (#5b) and (#A.28b), we infer:

$$exec\_permit_N(u_4, r_4). \quad \text{(\#A.29)}$$

In the context of the DD-DO Deny scheme, there are two ways to obtain the DENY authorization in (#A.28a). One way is through default decision (see (#A.20a)). The other way is through a combination of conflict resolution statement (see (#A.20b)) and DENY access decision statement (see (#A.20d)). In the following, we consider each of these possibilities in turn, and prove that our proposed mapping is correct using a contradiction strategy.

Obtaining DENY Authorization through Default Decision Suppose that (#A.28a) is obtained through default decision (see (#A.20a)). Based on (#A.20a), we infer:

$$\neg exec\_permit_D(u_4, r_4). \quad \text{(\#A.30a)}$$

$$\neg exec\_deny_D(u_4, r_4). \quad \text{(\#A.30b)}$$

In order to infer the predicate given in (#A.30a), none of the positive authorization rules in the Deny scheme can be applicable to $\langle u_4, r_4\rangle$. Thus, the conditional statement associated with positive rules (see (#A.1)) is unsatisfied:

$$\neg(C_i(u_4, r_4), \ldots, C_j(u_4, r_4)). \quad \text{(\#A.31)}$$

Since the conditional evaluation is the same in the Deny and the Negation schemes, (#A.31) applies to both schemes. Based on (#A.31) and (#A.21), we infer the following predicate about the authorization of $\langle u_4, r_4\rangle$ in the Negation scheme:

$$\neg exec\_permit_N(u_4, r_4). \quad \text{(\#A.32)}$$

However, the two predicates in (#A.29) and (#A.32) contradict with each other. Therefore, we conclude that (#A.28a) could not have been obtained through default decision.

Obtaining DENY Authorization through Access Decision and Conflict Resolution Suppose (#A.28a) is obtained using a combination of conflict resolution strategy (see (#A.20b)) and access decision rule (see (#A.20d)). From both these conditional statements, we infer the following predicate about the authorization of $\langle u_4, r_4\rangle$ in the Deny scheme:

$$exec\_deny_D(u_4, r_4). \quad \text{(\#A.33)}$$

In order to infer the predicate in (#A.33), without loss of generality, we consider that the first negative rule in (#A.2) is applicable

to $\langle u_4, r_4 \rangle$ in the Deny scheme. So, we infer the following clause regarding satisfaction of conditions by $\langle u_4, r_4 \rangle$:

$$C_n(u_4, r_4), \ldots, C_p(u_4, r_4). \quad \text{(\#A.34)}$$

Since the conditions are shared between the Negation and the Deny schemes, the clause in (#A.34) should be true in the Negation scheme as well. Therefore, based on (#A.34) and the authorization rule given in (#A.21), we infer the following predicate regarding the authorization of $\langle u_4, r_4 \rangle$ in the Negation scheme:

$$\neg exec\_permit_N(u_4, r_4). \quad \text{(\#A.35)}$$

However, the two predicates in (#A.29) and (#A.35) contradict each other. Therefore, (#A.28a) could not have been obtained through conflict resolution and access decision. Considering this and our earlier result, that (#A.28a) could not have been obtained through default decision, we conclude that the assumption stated in (#A.28) cannot be satisfied.

References

Al-Kahtani, M.A., Sandhu, R., 2004. Rule-based RBAC with negative authorization. In: 20th Annual Computer Security Applications Conference. IEEE, pp. 405–415.

Ammann, P., Sandhu, R.S., et al., 1992. Implementing transaction control expressions by checking for absence of access rights. In: ACSAC. Citeseer, pp. 131–140.

Bertino, E., Buccafurri, F., Ferrari, E., Rullo, P., 1999. A logical framework for reasoning on data access control policies. In: Proceedings of the 12th IEEE Computer Security Foundations Workshop. IEEE, pp. 175–189.

Bertino, E., Jajodia, S., Samarati, P., 1999. A flexible authorization mechanism for relational data management systems. ACM Transactions on Information Systems (TOIS) 17 (2), 101–140.

Bertino, E., Samarati, P., Jajodia, S., 1997. An extended authorization model for relational databases. IEEE Transactions on Knowledge and Data Engineering 9 (1), 85–101.

Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D., 1999. The keynote trust management system version 2. Technical Report. Internet RFC 2704, September.

Bruns, G., Fong, P.W., Siahaan, I., Huth, M., 2012. Relationship-based access control: its expression and enforcement through hybrid logic. In: Proceedings of the second ACM conference on Data and Application Security and Privacy, pp. 117–124.

Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M., Thuraisingham, B., 2009. A semantic web based framework for social network access control. In: Proceedings of the 14th ACM symposium on Access control models and technologies, pp. 177–186.

Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M., Thuraisingham, B., 2011. Semantic web-based social network access control. computers & security 30 (2-3), 108–115.

Cheng, Y., Bijon, K., Sandhu, R., 2016. Extended ReBAC administrative models with cascading revocation and provenance support. In: Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies, pp. 161–170.

Cheng, Y., Park, J., Sandhu, R., 2012. Relationship-based access control for online social networks: Beyond user-to-user relationships. In: 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing. IEEE, pp. 646–655.

Cheng, Y., Park, J., Sandhu, R., 2012. A user-to-user relationship-based access control model for online social networks. In: IFIP Annual Conference on Data and Applications Security and Privacy. Springer, pp. 8–24.

Cheng, Y., Park, J., Sandhu, R., 2014. Attribute-aware relationship-based access control for online social networks. In: IFIP Annual Conference on Data and Applications Security and Privacy. Springer, pp. 292–306.

Chomicki, J., Lobo, J., Naqvi, S., 2003. Conflict resolution using logic programming. IEEE Transactions on Knowledge and Data Engineering 15 (1), 244–249.

Chu, Y.-H., Feigenbaum, J., LaMacchia, B., Resnick, P., Strauss, M., 1997. REFEREE: Trust management for Web applications. Computer Networks and ISDN systems 29 (8-13), 953–964.

Clarke, D., Elien, J.-E., Ellison, C., Fredette, M., Morcos, A., Rivest, R.L., 2001. Certificate chain discovery in SPKI/SDSI. Journal of Computer security 9 (4), 285–322.

Crampton, J., Morisset, C., 2012. PTaCL: A language for attribute-based access control in open systems. In: International Conference on Principles of Security and Trust. Springer, pp. 390–409.

Crampton, J., Sellwood, J., 2014. Path conditions and principal matching: a new approach to access control. In: Proceedings of the 19th ACM symposium on Access control models and technologies, pp. 187–198.

Crampton, J., Sellwood, J., 2015. Relationships, paths and principal matching: a new approach to access control. arXiv preprint arXiv:1505.07945.

Crampton, J., Sellwood, J., 2016. ARPPM: Administration in the RPPM Model. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pp. 219–230.

Czenko, M., Tran, H., Doumen, J., Etalle, S., Hartel, P., den Hartog, J., 2006. Nonmonotonic trust management for P2P applications. Electronic Notes in Theoretical Computer Science 157 (3), 113–130.

Dong, C., Dulay, N., 2010. Shinren: Non-monotonic trust management for distributed systems. In: IFIP International Conference on Trust Management. Springer, pp. 125–140.

Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T., 1999. SPKI certificate theory. Technical Report. RFC 2693.

eXtensible Access Control Markup Language (XACML) Version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013.

Ferraiolo, D., Kuhn, R., 1992. Role-based access control. In: Proceedings of the 15th NIST-NSA National Computer Security Conference, pp. 13–16.

Fong, P.W., 2011. Relationship-based access control: protection model and policy language. In: Proceedings of the first ACM conference on Data and application security and privacy, pp. 191–202.

Fong, P.W., Siahaan, I., 2011. Relationship-based access control policies and their policy languages. In: Proceedings of the 16th ACM symposium on Access control models and technologies, pp. 51–60.

Gal-Oz, N., Gudes, E., Fernández, E.B., 1993. A Model of Methods Access Authorization in Object-Oriented Databases. In: Proceedings of the 19th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 52–61.

Graham, G.S., Denning, P.J., 1971. Protection: principles and practice. In: Proceedings of the May 16-18, 1972, spring joint computer conference, pp. 417–429.

Harrison, M.A., Ruzzo, W.L., Ullman, J.D., 1976. Protection in operating systems. Communications of the ACM 19 (8), 461–471.

Herzberg, A., Mass, Y., Mihaeli, J., Naor, D., Ravid, Y., 2000. Access control meets public key infrastructure, or: Assigning roles to strangers. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. IEEE, pp. 2–14.

Hu, H., Ahn, G.-J., 2011. Multiparty authorization framework for data sharing in online social networks. In: IFIP Annual Conference on Data and Applications Security and Privacy. Springer, pp. 29–43.

Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, A.R., Lang, A.J., Cogdell, M.M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., 2013. Guide to attribute based access control (ABAC) definition and considerations (draft). NIST special publication 800 (162).

Jajodia, S., Samarati, P., Subrahmanian, V., 1997. A logical language for expressing authorizations. In: Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097). IEEE, pp. 31–42.

Jha, S., Sural, S., Atluri, V., Vaidya, J., 2019. Security analysis of ABAC under an administrative model. IET information security 13 (2), 96–103.

Jin, X., Krishnan, R., Sandhu, R., 2012. A unified attribute-based access control model covering DAC, MAC and RBAC. In: IFIP Annual Conference on Data and Applications Security and Privacy. Springer, pp. 41–55.

Kandala, S., Sandhu, R., Bhamidipati, V., 2011. An attribute based framework for risk-adaptive access control models. In: 2011 Sixth International Conference on Availability, Reliability and Security. IEEE, pp. 236–241.

Lampson, B.W., 1974. Protection. ACM SIGOPS Operating Systems Review 8 (1), 18–24.

Lang, B., Foster, I., Siebenlist, F., Ananthakrishnan, R., Freeman, T., 2009. A flexible attribute based access control method for grid computing. Journal of Grid Computing 7 (2), 169.

Li, N., Mitchell, J.C., 2003. RT: A role-based trust-management framework. In: Proceedings DARPA Information Survivability Conference and Exposition, Vol. 1. IEEE, pp. 201–212.

Li, N., Mitchell, J.C., Winsborough, W.H., 2002. Design of a role-based trust-management framework. In: Proceedings 2002 IEEE Symposium on Security and Privacy. IEEE, pp. 114–130.

Li, N., Mitchell, J.C., Winsborough, W.H., 2005. Beyond proof-of-compliance: security analysis in trust management. Journal of the ACM (JACM) 52 (3), 474–514.

Li, N., Tripunitara, M.V., 2006. Security analysis in role-based access control. ACM Transactions on Information and System Security (TISSEC) 9 (4), 391–420.

Lipton, R.J., Snyder, L., 1977. A linear time algorithm for deciding subject security. Journal of the ACM (JACM) 24 (3), 455–464.

Masoumzadeh, A., 2018. Security analysis of relationship-based access control policies. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, pp. 186–195.

Masoumzadeh, A., Joshi, J., 2010. OSNAC: An ontology-based access control model for social networking systems. In: 2010 IEEE Second International Conference on Social Computing. IEEE, pp. 751–759.

Moyer, M.J., Abamad, M., 2001. Generalized role-based access control. In: Proceedings 21st International Conference on Distributed Computing Systems. IEEE, pp. 391–398.

Park, J., Sandhu, R., 2004. The UCONABC usage control model. ACM Transactions on Information and System Security (TISSEC) 7 (1), 128–174.

Pasarella, E., Lobo, J., 2017. A datalog framework for modeling relationship-based access control policies. In: Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, pp. 91–102.

Rivest, R.L., Lampson, B., 1996. SDSI-a simple distributed security infrastructure. Crypto.

Rizvi, S.Z.R., Fong, P.W., Crampton, J., Sellwood, J., 2015. Relationship-based access control for an open-source medical records system. In: Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, pp. 113–124.

Saltzer, J.H., Schroeder, M.D., 1975. The protection of information in computer systems. Proceedings of the IEEE 63 (9), 1278–1308.

Sandhu, R.S., 1988. The schematic protection model: its definition and analysis for acyclic attenuating schemes. Journal of the ACM (JACM) 35 (2), 404–432.

Sandhu, R.S., 1992. The typed access matrix model. In: IEEE Symposium on Security and Privacy, pp. 122–136.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E., 1996. Role-based access control models. Computer 29 (2), 38–47.

Servos, D., Osborn, S.L., 2014. HGABAC: Towards a formal model of hierarchical attribute-based access control. In: International Symposium on Foundations and Practice of Security. Springer, pp. 187–204.
Shen, H.-b., Hong, F., 2006. An attribute-based access control model for web services. In: 2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06). IEEE, pp. 74–79.
Stoller, S.D., 2015. An administrative model for relationship-based access control. In: IFIP Annual Conference on Data and Applications Security and Privacy. Springer, pp. 53–68.
Tripunitara, M.V., Li, N., 2007. A theory for comparing the expressive power of access control models 1. Journal of Computer Security 15 (2), 231–272.
di Vimercati, S.D.C., Samarati, P., Jajodia, S., 2005. Policies, models, and languages for access control. In: International Workshop on Databases in Networked Information Systems. Springer, pp. 225–237.
Wang, L., Wijesekera, D., Jajodia, S., 2004. A logic-based framework for attribute based access control. In: Proceedings of the 2004 ACM workshop on Formal methods in security engineering, pp. 45–55.

Padmavathi Iyer is a PhD candidate of Computer Science at the University at Albany - SUNY, and a member of the Albany Lab for Privacy and Security (ALPS). She received her bachelor's and master's degrees in Computer Science/Engineering from GGSIPU, Delhi, India and University at Albany - SUNY, respectively. Her research interests include analysis and mining of access control policies. Her prior work has received the best paper award from ACM SACMAT 2020.

Amirreza Masoumzadeh is an Assistant Professor of Computer Science and co-director of the Albany Lab for Privacy & Security (ALPS) at the University at Albany - SUNY. He received his PhD in Information Science from the University of Pittsburgh, and earlier, his BS and MS degrees in Computer Science/Engineering from Ferdowsi University, Iran and Sharif University of Technology, Iran, respectively. His research interests revolve around information security, privacy, and trust in modern computing systems, including theoretical models and mechanisms for specification, verification, and analysis of access control policies.

Paliath Narendran is a Professor in the Department of Computer Science at the University at Albany, State University of New York. He received his PhD in Computer Science from the Rensselaer Polytechnic Institute in 1984. Before joining UAlbany in 1988, Dr. Narendran worked in the Computer Science Branch of General Electric Corporate Research and Development in Schenectady, New York. His research interests include automated reasoning, formal verification and pattern matching.