Converting Rule-Based Access Control Policies: From Complemented Conditions to Deny Rules

Josué A. Ruiz
University at Albany – SUNY
Albany, New York, USA
jaruiz@albany.edu

Paliath Narendran
University at Albany – SUNY
Albany, New York, USA
pnarendran@albany.edu

Amir Masoumzadeh
University at Albany – SUNY
Albany, New York, USA
amasoumzadeh@albany.edu

Padmavathi Iyer
Drury University
Springfield, Missouri, USA
riyer@drury.edu

ABSTRACT

Using access control policy rules with deny effects (i.e., negative authorization) can be preferred to using complemented conditions in the rules as they are often easier to comprehend in the context of large policies. However, the two constructs have different impacts on the expressiveness of a rule-based access control model. We investigate whether policies expressible using complemented conditions can be expressed using deny rules instead. The answer to this question is not always affirmative. In this paper, we propose a practical approach to address this problem for a given policy. In particular, we develop theoretical results that allow us to pose the problem as a set of queries to an SAT solver. Our experimental results using an off-the-shelf SAT solver demonstrate the feasibility of our approach and offer insights into its performance based on access control policies from multiple domains.

CCS CONCEPTS

• **Security and privacy → Formal security models**; **Authorization**; **Access control**.

KEYWORDS

access control, complemented conditions, deny rules, negative authorization

ACM Reference Format:
Josué A. Ruiz, Paliath Narendran, Amir Masoumzadeh, and Padmavathi Iyer. 2024. Converting Rule-Based Access Control Policies: From Complemented Conditions to Deny Rules. In *Proceedings of the 29th ACM Symposium on Access Control Models and Technologies (SACMAT 2024), May 15–17, 2024, San Antonio, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3649158.3657040

SACMAT 2024, May 15–17, 2024, San Antonio, TX, USA

ACM ISBN 979-8-4007-0491-8/24/05
https://doi.org/10.1145/3649158.3657040

1 INTRODUCTION

Rule-based access control policies determine authorizations on system resources based on evaluating a set of rules each involving a conditional expression. In attribute-based access control [1, 14, 31], a form of rule-based policy model, the conditional expressions in rules are based on testing subject and object attributes. For instance, a rule in a university policy can authorize subjects who are faculty members of the CS department to view objects that are transcripts of CS students. A more restrictive version of such a rule may only allow CS faculty members who are not currently on leave. Assuming that "being on leave" can be tested using subject attributes (i.e., a condition), this rule can be expressed as a rule with a conjunctive conditional expression that tests for the condition of "being a CS faculty" and the complement of condition "being on leave" (i.e., a complemented condition). Such a construction of rule-based policies is not universal. As shown in a previous work [29], depending on the building blocks of a rule-based model, those models can vary in their expressiveness power. Following the terminology in that work, let the policy model that allows conditional expressions with both complemented and uncomplemented conditions be called *Negation.* An alternative model of policies, called *DDDO* (short for deny-by-default and deny-overrides), does not support such complemented conditions but instead allows deny rules in addition to permit rules. This approach is also known as negative authorization. For instance, a DDDO policy can address the example above using two rules: a first rule that permits all CS professors to access the transcripts, and a second rule that denies such access if on leave. Here, the deny effect of the second rule overrides the permit effect of the first rule whenever an on-leave CS professor requests access.

Using deny rules (with uncomplemented conditions) could be preferred to using complemented conditions in rules as complemented conditions may complicate the formulation and comprehension of larger policies. However, as shown previously [29] the Negation model is more expressive than the DDDO model. Therefore, a policy that is specified using the Negation model may or may not be expressible in (or *convertible to*) DDDO. Hence, when such a policy migration is considered it is important to test for the *convertibility* of the policy. It can be shown that the convertibility problem is computationally hard (see Appendix A). Our goal in this paper is to approach this problem from a practical point of view: is it feasible to determine the convertibility of real-world Negation policies to DDDO policies in a reasonable time (despite being a computationally hard problem)?

We approach this problem by formulating queries about the semantics of a Negation policy that can be answered using an SAT solver. The goal of these queries is to verify a Boolean disjunctive normal form (DNF) [13] expressible by the Negation model is of convex shape (See Section 2 for background on policy semantics). Since Sat solvers are highly efficient and widely used, this approach is quite practical as we have found.

We summarize our contributions in this work as follows:

- We introduce the concepts of *gap* and *separator* that formally capture the subsets of policy semantics which possibly could lead to the semantics being not convex.
- We develop theories that allow us to use the gap/separator concepts to formulate queries to a Sat solver that determine the convertibility of a Negation policy to a DDDO policy. In policy semantics terms, these establish how to formally verify the convexity of a disjunctive normal form expression semantics.
- We develop a prototype implementation of our convertibility testing tool and extensively experiment with various sizes of policies derived from real-world domains. Our results demonstrate both the feasibility and scalability of this approach in practice.

The rest of this paper is organized as follows. In Section 2, we briefly discuss the necessary background on the semantics of rule-based access control models [29] including some of our extended notations. In Section 3, we introduce the concepts of *gap* and *separator*, and establish how they can be utilized to reason about convertibility of a Negation semantics to a DDDO semantics. Following those theories, we discuss the corresponding algorithms in Section 4 and present an experimental evaluation of our approach in Section 5 using policies from three domains. Section 6 reviews the closely related work, and we provide concluding remarks in Section 7.

2 BACKGROUND AND PRELIMINARIES

In this section, we provide the necessary background, which is inspired by a prior work [29], accompanied by additional notations required in our paper.

A rule-based access control policy consists of a set of rules. Each rule is a pair of a term (also known as a conditional expression) and an effect. Each term is a conjunctive expression on the set of Boolean variables $X = \{x_1, \ldots, x_n\}$. Each x_i corresponds to a condition that can be tested in the authorization process. A literal based on x_i may be present in a term in uncomplemented (x_i) or complemented ($\overline{x_i}$) form. For a term t, we denote the variables and literals that appear in it by $Var(t)$ and $Lit(t)$, respectively. Furthermore, we use $t|_V$, restriction of t to variables V, to indicate the term resulting from omitting any literal in t that is not in V.

We consider two policy models in this paper. In the *Negation* model, terms may include complemented variables. However, the only acceptable rule effect is `PERMIT`. In the *DDDO* model, terms may only include uncomplemented variables. But each rule effect may be either `PERMIT` or `DENY`. Potential conflicts as the result of multiple rules being applicable for an access request are resolved using a *deny-overrides* strategy. In the case of both models, a *deny-by-default* strategy is applied to a request with no applicable rule.

The concept of *minterm* defined below can be used to capture the semantics of expressions and policies.

Definition 1 (Minterm [29]). *A* minterm *over Boolean variables X is defined as the conjunction of all $x_i \in X$ either in complemented ($\overline{x_i}$) or uncomplemented (x_i) form.*

The set of all minterms over variables X is denoted by $\mathcal{M}_{\mathcal{X}}$. A partial ordering on minterms (denoted by "$\geq$") can be formed such that $m_1 \geq m_2$ if and only if the positive literals of m_1 include the positive literals of m_2. For example, $m_1 \geq m_2$ given $X = \{x_1, x_2, x_3, x_4\}$, $m_1 = x_1\overline{x_2}x_3x_4$, and $m_2 = \overline{x_1}\,\overline{x_2}x_3\overline{x_4}$. The ordering is strict ($m_1 > m_2$) if they are distinct minterms. We can also define the notion of partial ordering for terms that have the same set of variables. For example, $t_1 \geq m_1|_{Var(t_1)}$ where $t_1 = x_1x_2$.

The semantics of a term t, denoted by $\mu(t)$ is the set of all minterms that are implicants of t. For example, assuming $X = \{x_1, x_2, x_3\}$, $\mu(x_1\overline{x_3}) = \{x_1x_2\overline{x_3}, x_1\overline{x_2}\,\overline{x_3}\}$. The semantics of a policy, also denoted by $\mu()$, is the set of all minterms that the policy permits. Since the Negation model only supports permit rules, any Negation policy can be viewed as a single boolean expression in disjunctive normal form (DNF). Throughout this paper, we use Φ to refer to such a DNF expression. Furthermore, as its rule terms support both complemented and uncomplemented variables it can be shown that the Negation model can express any set of minterms (or any DNF) [29]. In contrast, the DDDO model is capable of capturing all and only convex sets of minterms as defined below.

Definition 2 (Convex Set of Minterms [29]). *A set of minterms M is* convex *iff for every pair of minterms $m_1 \leq m_2$ in M: $\{m \mid m_1 \leq m \leq m_2\} \subseteq M$.*

3 CHARACTERIZING CONVERTIBILITY TO DDDO

As presented in Section 2, the DDDO model can only express convex semantics while the Negation model can express any semantics. Therefore, the decision problem of convertibility of Negation policy P_N to DDDO policy P_D can be viewed as testing the convexity of the set of minterms $\mu(P_N)$.

In this section, we define the notion of *gaps* between terms (rules) in a policy that could lead to non-convexity of policy minterms. We characterize such gaps using the concept of *seperator*, and establish theoretical results on how such separators can be used to decide about the convexity of the policy minterms.

3.1 Gap Between Two Terms

Definition 3 (Gap between Terms). *Given two terms t_1 and t_2, we say there is a* gap *between t_2 and t_1 if and only if there exist minterms $m_1 > m > m_2$ such that*

(1) $m_1 \in \mu(t_1)$,
(2) $m_2 \in \mu(t_2)$, and
(3) $m \notin \mu(t_1) \cup \mu(t_2)$.

This is illustrated in Figure 1.

Our aim is to develop a syntactic criterion (that can be easily checked) for the existence of a gap between two terms t_2 and t_1.

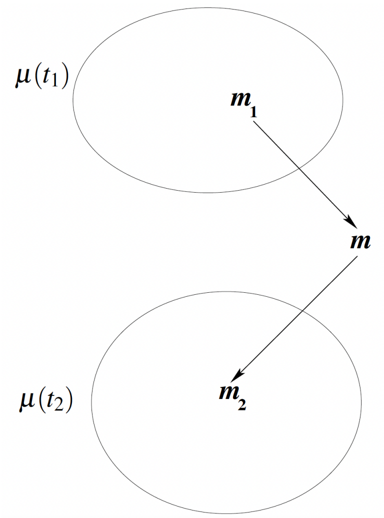

Figure 1: A *Gap* Between Terms t_1 and t_2, Identified by Minterm m Where $m_1 > m > m_2$, $m_1 \in \mu(t_1)$, and $m_2 \in \mu(t_2)$

Table 1: Term Comparison Functions

Function	Definition
$\Omega(t_1, t_2)$	Consists of literals that are common to both terms (i.e., their common factor).
$\aleph(t_1, t_2)$	Consists of variables that appear uncomplemented in t_1 and complemented in t_2.
$\mathcal{B}_1(t_1, t_2)$	Consists of positive literals that appear in t_1 whose variables do not appear in t_2.
$\mathcal{B}_2(t_1, t_2)$	Consists of positive literals that appear in t_2 whose variables do not appear in t_1.
$\Gamma_1(t_1, t_2)$	Consists of negative literals that appear in t_1 whose variables do not appear in t_2.
$\Gamma_2(t_1, t_2)$	Consists of negative literals that appear in t_2 whose variables do not appear in t_1.

As a first step, it will be useful to carefully express the possible differences between the two terms in terms of the literals they do and do not share. There can be no gap between t_2 and t_1 if there is a variable that appears complemented in t_1 and uncomplemented in t_2. Thus that case can be immediately ruled out. For the rest of the analysis *we only consider terms t_1 and t_2 in which no complemented variable in t_1 appears uncomplemented in t_2*. Formally, let t_1 and t_2 be two distinct product terms. We say that t_1 is *potentially higher than* t_2, denoted as

$$t_1 \sqsupset t_2$$

if and only if there is no variable that appears uncomplemented in t_2 and complemented in t_1. Note that this is equivalent to saying

$$\exists m_1 \in \mu(t_1)\, \exists m_2 \in \mu(t_2) : \; m_1 \; > \; m_2$$

Let t_1 and t_2 be two distinct product terms. We define a set of functions for comparing the two terms (from t_1's point of view) in Table 1. Note that $\mathcal{B}_1(t_1, t_2) = \mathcal{B}_2(t_2, t_1)$ and $\Gamma_1(t_1, t_2) = \Gamma_2(t_2, t_1)$. Furthermore, $t_1 \sqsupset t_2$ if and only if $|\aleph(t_2,\, t_1)| \; = \; 0$.

For the sake of brevity and clarity, we use the following abbreviations when the two terms under consideration are t_1 and t_2:

(1) $\omega \; = \; \Omega(t_1, t_2)$,
(2) $\alpha \; = \; \aleph(t_1, t_2)$,
(3) $\beta_1 \; = \; \mathcal{B}_1(t_1, t_2)$,
(4) $\beta_2 \; = \; \mathcal{B}_2(t_1, t_2)$,
(5) $\gamma_1 \; = \; \Gamma_1(t_1, t_2)$, and
(6) $\gamma_2 \; = \; \Gamma_2(t_1, t_2)$.

We can then symbolically express the two terms as follows:

$$\begin{aligned} t_1 &= \omega\, \alpha\, \beta_1\, \gamma_1 \\ t_2 &= \omega\, \widetilde{\alpha}\, \beta_2\, \gamma_2 \end{aligned}$$

A better depiction of this would be:

$$\begin{aligned} t_1 &= \omega\, \alpha\, \beta_1\, \gamma_1 \,\_\_\,\_\_ \\ t_2 &= \omega\, \widetilde{\alpha} \,\_\_\,\_\_\, \beta_2\, \gamma_2 \end{aligned}$$

where the blanks are for positive and negative literals that do exist in one term but not the other. Also, note that $\widetilde{\alpha}$ is the product term resulting from complementing all the variables in α.

Example 1. *Let $V = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ be a set of Boolean variables and t_1 and t_2 terms. To show the asymmetry (non-commutativity) of the functions α, β_1, etc., we evaluate the functions in both directions.*

$$t_1 = x_1 \overline{x_2} x_3 x_6$$
$$t_2 = x_1 x_4 \overline{x_5}\, \overline{x_6}$$

Observe that

$$\Omega(t_1, t_2) = \{x_1\}\,,\ \aleph(t_1, t_2) = \{x_6\}\,,\ \mathcal{B}_1(t_1, t_2) = \{x_3\}$$
$$\Omega(t_2, t_1) = \{x_1\}\,,\ \aleph(t_2, t_1) = \{\,\}\,,\ \ \mathcal{B}_1(t_2, t_1) = \{x_4\}$$
$$\mathcal{B}_2(t_1, t_2) = \{x_4\}\,,\ \Gamma_1(t_1, t_2) = \{x_2\}\,,\ \Gamma_2(t_1, t_2) = \{x_5\}$$
$$\mathcal{B}_2(t_2, t_1) = \{x_3\}\,,\ \Gamma_1(t_2, t_1) = \{x_5\}\,,\ \Gamma_2(t_2, t_1) = \{x_2\}$$

Note that Ω is commutative since it returns the common factor *of the two terms.*

Definition 4. *A term t* dominates *a minterm m if and only if there is a minterm $m' \in \mu(t)$ such that $m < m'$.*

Lemma 1. *Let t be a term and m be a minterm such that $m \notin \mu(t)$. Then t dominates m if and only if $t \; > \; m|_{Var(t)}$.*

Proof. First of all, note that t is a subterm of every minterm in $\mu(t)$. So if $t \; = \; m|_{Var(t)}$, then $m \; \in \; \mu(t)$. Furthermore, if there is a variable x such that x appears (uncomplemented) in m and $\overline{x}$ appears in t, then no term in $\mu(t)$ can be higher than m.

If $t \; > \; m|_{Var(t)}$, then let $m'' \; = \; \frac{m}{m|_{Var(t)}}$, i.e., write m as $m'm''$ where

$$\begin{aligned} m' &= m|_{Var(t)}, \\ m'' &= m|_{X \setminus Var(t)}, \end{aligned}$$

and horizontal bar stands for algebraic division. Then $tm'' \; > \; m$. $\square$

Definition 5. *A minterm m* leads *a term t if and only if there is a minterm $m' \in \mu(t)$ such that $m > m'$.*

Thus a minterm m is in the gap between t_2 and t_1 if and only if $m \notin \mu(t_1) \cup \mu(t_2)$, t_1 dominates m, and m leads t_2.

Lemma 2. *Let t be a term and m be a minterm such that $m \notin \mu(t)$. Then m leads t if and only if $m|_{Var(t)} > t$.*

Proof. Note that t is a subterm of every term of $\mu(t)$. If $t = m|_{Var(t)}$, then $m \in \mu(t)$. Additionally, if there is a variable x such that x appears complemented in m but uncomplemented in t, then m cannot be higher than any minterm in $\mu(t)$.

If $m|_{Var(t)} > t$, then let $m'' = \frac{m}{m|_{Var(t)}}$, i.e., write m as $m'm''$ where

$$\begin{aligned} m' &= m|_{Var(t)} \\ m'' &= m|_{X \setminus Var(t)} \end{aligned}$$

Then $m > tm''$. □

Lemma 3. *Let t_1, t_2 be terms and let m be a minterm such that t_1 dominates m and m leads t_2. Then $\omega\beta_2\gamma_1$ must be a subterm of m.*

Proof. We split the proof into 3 cases, comparing m with ω, β_2 and γ_1.

Case (a): Consider t_1, t_2 and m restricted to variables of ω, namely $t_1|_{Var(\omega)}$, $m|_{Var(\omega)}$, and $t_2|_{Var(\omega)}$. First of all, we know that $t_1|_{Var(\omega)} = t_2|_{Var(\omega)} = \omega$. Note that given that t_1 dominates m there cannot be a variable x complemented in $t_1|_{Var(\omega)}$ but uncomplemented in $m|_{Var(\omega)}$. Similarly, given m leads t_2 then there cannot be a complemented variable in $m|_{Var(\omega)}$ that appears uncomplemented in $t_2|_{Var(\omega)}$. Thus it must be that

$$\omega \geq m|_{Var(\omega)} \geq \omega$$

which implies that

$$\omega = m|_{Var(\omega)}.$$

Case (b): Consider t_1, t_2 and m restricted to variables of β_2, i.e.,

$$t_1|_{Var(\beta_2)} = \varepsilon, \quad t_2|_{Var(\beta_2)} = \beta_2$$

If m leads t_2 then there cannot be a complemented variable in $m|_{Var(\beta_2)}$ which appears uncomplemented in $t_2|_{Var(\beta_2)}$. Besides, there are no complemented variables in β_2. Thus we have that

$$\beta_2 = m|_{Var(\beta_2)}.$$

Case (c): Consider t_1, t_2 and m restricted to variables of γ_1. Similar to the previous case,

$$t_1|_{Var(\gamma_1)} = \gamma_1, \quad t_2|_{Var(\gamma_1)} = \varepsilon$$

Since t_1 dominates m there cannot be a variable x complemented in $t_1|_{Var(\gamma_1)}$ but uncomplemented in $m|_{Var(\gamma_1)}$. Besides, by definition, there are no uncomplemented variables in γ_1. Then we have that

$$\gamma_1 = m|_{Var(\gamma_1)}.$$ □

We later denote the term $\omega\beta_2\gamma_1$ as the <u>separator</u> of t_1 from t_2.

Lemma 4. *Suppose $t_1 \sqsupset t_2$. Then*

(a) If $|\alpha\beta_1| = 0$, then there is no gap between t_2 and t_1.
(b) If $|\alpha\gamma_2| = 0$, then there is no gap between t_2 and t_1.

Proof. Suppose there exists a minterm m such that t_1 dominates m and m leads t_2, i.e., $t_1 > m|_{Var(t_1)}$ and $m|_{Var(t_2)} > t_2$.

Case (a): If $|\alpha| = |\beta_1| = 0$, then

$$\begin{aligned} t_1 &= \omega\,\gamma_1 \_\_\,\_\_ \\ t_2 &= \omega \_\_\, \beta_2\,\gamma_2 \end{aligned}$$

By Lemma 3, $\omega\beta_2\gamma_1$ is a subterm of m. Now the result follows directly from Lemma 1, since $m|_{Var(t_1)} = \omega\gamma_1 = t_1$.

Case (b): If $|\alpha| = |\gamma_2| = 0$, then $t_2 = \omega\beta_2 = m|_{Var(t_2)}$. But for t_2 to lead m, by Lemma 2 we must have that $m|_{Var(t_2)} > t_2$. □

The next result is a necessary condition for the existence of a gap between two terms.

Lemma 5. *There is a gap between terms t_2 and t_1 with $t_1 \sqsupset t_2$ only if*

$$|\alpha\beta_1| > 1 \;\vee\; |\beta_1\gamma_2| > 1 \;\vee\; |\alpha\gamma_2| > 1.$$

Proof. Suppose $|\alpha\beta_1| \leq 1 \;\wedge\; |\beta_1\gamma_2| \leq 1 \;\wedge\; |\alpha\gamma_2| \leq 1$ and still there is a minterm m such that t_1 dominates m and m leads t_2, i.e., $t_1 > m|_{Var(t_1)}$ and $m|_{Var(t_2)} > t_2$. By the previous lemma, it must be that $|\alpha| + |\beta_1| = 1$ and $|\alpha| + |\gamma_2| = 1$. Thus $|\alpha| = 1$, for otherwise $|\beta_1\gamma_2| = 2$. It follows that $|\beta_1| = |\gamma_2| = 0$.

By Lemma 3, $\omega\beta_2\gamma_1$ is a subterm of m. Let $\alpha = x$ where x is a variable. Thus $\widetilde{\alpha} = \overline{x}$.

$$\begin{aligned} t_1 &= \omega\,x\,\gamma_1 \_\_ \\ t_2 &= \omega\,\overline{x} \_\_\, \beta_2 \end{aligned}$$

We need to consider two cases: (a) x appears in m, and (b) $\overline{x}$ appears in m.

Case (a): Here $\omega x \beta_2\gamma_1$ is a subterm of m. Thus, $m|_{Var(t_1)} = \omega x \gamma_1 = t_1$. But for t_1 to dominate m, by Lemma 1 we must have that $t_1 > m|_{Var(t_1)}$.

Case (b): $\omega\overline{x}\beta_2\gamma_1$ is a subterm of m. For t_2 to lead m, by Lemma 2 we must have that $m|_{Var(t_2)} > t_2$. But note that $m|_{Var(t_2)} = \omega\overline{x}\beta_2 = t_2$. □

Lemma 6. *If $|\alpha| > 1$ and $t_1 \sqsupset t_2$ then there is a gap between t_2 and t_1, i.e., there is a minterm $m \notin \mu(t_1) \cup \mu(t_2)$ such that t_1 dominates m and m leads t_2.*

Proof. Let x_1 and x_2 be two variables in α, i.e., x_1x_2 is a subterm of t_1 and $\overline{x_1}\,\overline{x_2}$ is a subterm of t_2. Let $\alpha = x_1x_2\eta$ where η is the rest of α, i.e., the conjunction of the remaining variables in α. Clearly $\mu(t_1)$ and $\mu(t_2)$ are disjoint. Besides, $x_1\overline{x_2}$ is not a subterm of any of the minterms in $\mu(t_1) \cup \mu(t_2)$. Now consider the term

$$t = \omega\,x_1\,\overline{x_2}\,\eta\,\beta_1\,\beta_2\,\gamma_1\,\gamma_2 \text{ and let } m \in \mu(t).$$

Note that

$$m|_{Var(t_1)} = \omega x_1\overline{x_2}\eta\beta_1\gamma_1 < \omega x_1 x_2\eta\beta_1\gamma_1 = t_1,$$

by Lemma 1 we must have that t_1 dominates m. Similarly, we have that

$$m|_{Var(t_2)} = \omega x_1\overline{x_2}\eta\beta_2\gamma_2 > \omega\overline{x_1}\,\overline{x_2}\eta\beta_2\gamma_2 = t_2,$$

then by Lemma 2 we must have that m leads t_2. □

Theorem 1. *There is a gap between terms t_2 and t_1 with $t_1 \sqsupset t_2$ if and only if*

$$|\alpha| > 1 \;\vee\; \text{any two of } |\alpha|,\ |\beta_1|,\ |\gamma_2| \text{ are greater than } 0.$$

Proof. The "only if" part follows from Lemma 6. For proving the "if" part, suppose that there is no m such that t_1 dominates m and m leads t_2 and still the condition is true. If $|\alpha| > 1$ then there is a gap by Lemma 6. We again need to consider 3 cases:

Case (a): $|\alpha| = 1, |\beta_1| > 0.$

Consider the term $t_3 = \omega\alpha\widetilde{\beta_1}\beta_2\gamma_2$. It is not hard to see that $t_1 > t_3|_{Var(t_1)}$ and $t_3|_{Var(t_2)} > t_2$. Thus the same holds for any $m \in \mu(t_3)$, i.e.,

$$t_1 > m|_{Var(t_1)} \text{ and } m|_{Var(t_2)} > t_2.$$

Case (b): $|\alpha| = 1, |\gamma_2| > 0.$

Consider the term $t_4 = \omega\widetilde{\alpha}\beta_1\gamma_1\beta_2\gamma_2$. As above, $t_1 > t_4|_{Var(t_1)}$ and $t_4|_{Var(t_2)} > t_2$. Similarly to case (a) it will hold for any minterm $m \in \mu(t_4)$, i.e.,

$$t_1 > m|_{Var(t_1)} \text{ and } m|_{Var(t_2)} > t_2.$$

Case (c): $|\alpha| = 0, |\beta_1| > 0, |\gamma_2| > 0.$

Consider the term $t_5 = \omega\widetilde{\beta_1}\gamma_1\beta_2\gamma_2$. As above $t_1 > t_5|_{Var(t_1)}$ and $t_5|_{Var(t_2)} > t_2$. Similarly to case (a) it will hold for any minterm $m \in \mu(t_5)$, i.e.,

$$t_1 > m|_{Var(t_1)} \text{ and } m|_{Var(t_2)} > t_2. \qquad \square$$

3.2 Gap in an Expression

In the previous section, we only considered gaps between two terms without considering any larger expression that they may be a part of. Here we consider the general problem.

Definition 6 (Gap wrt. an Expression). *Given two terms t_1, t_2 and a Boolean expression Φ, we say that there is a* gap *between t_1 and t_2* with respect to Φ *if and only if there exist minterms $m_1 > m > m_2$ such that*

(1) $m_1 \in \mu(t_1)$,
(2) $m_2 \in \mu(t_2)$, and
(3) m is not an implicant of $\Phi \vee t_1 \vee t_2$.

If t_1 and t_2 are part of Φ then we only need to say "m is not an implicant of Φ."

Definition 7. *Let t_1 and t_2 terms in Φ and $t_1 \sqsupset t_2$. We denote $\omega\beta_2\gamma_1$ as the* separator *of t_1 from t_2.*

We introduce the following notation:

$$sep(t_1, t_2) = \Omega(t_1, t_2)\, \mathcal{B}_2(t_1, t_2)\, \Gamma_1(t_1, t_2)$$

to represent the separator of t_1 from t_2.

Note also that this symbolic expression, $\omega\beta_2\gamma_1$, already appeared in Lemma 3. The following lemma strengthens Lemma 3.

Lemma 7. *Let Φ be an expression in DNF and t_1 and t_2 be terms in Φ such that $t_1 \sqsupset t_2$. Let $s = sep(t_1, t_2)$. Then for any $m \in \mu(s)$ either $m \in \mu(t_1)$, or $m \in \mu(t_2)$, or there exist minterms $m_1 \in \mu(t_1)$ and $m_2 \in \mu(t_2)$ such that $m_1 > m > m_2$.*

Proof. Suppose m does not belong to either $\mu(t_1)$ or $\mu(t_2)$. Clearly, t_1 and t_2 cannot be subterms of m. Note also that $Var(t_1) = Var(\omega) \uplus Var(\alpha) \uplus Var(\beta_1) \uplus Var(\gamma_1)$. Since $\omega\gamma_1$ is a subterm of $m|_{Var(t_1)}$ and t_1 is not a subterm of m, there must be a variable in $\alpha\beta_1$ (in t_1) that appears complemented in m. Thus $t_1\beta_2 > m|_{Var(t_1\beta_2)}$. Similarly, there must be a variable $\widetilde{\alpha}\gamma_2$ (in t_2) that appears uncomplemented in m. Thus $t_2\gamma_1 < m|_{Var(t_2\gamma_1)}$.

Now consider the terms $s_1 = t_1\beta_2\widetilde{\gamma_2}$ and $s_2 = t_2\widetilde{\beta_1}\gamma_1$. Let $V = Var(s_1) = Var(s_2) = Var(t_1) \cup Var(t_2)$. It is not hard to show that

$$s_1 > m|_V > s_2$$

and the result follows. $\square$

Theorem 2. *Let Φ be an expression in DNF and t_1 and t_2 be terms in Φ such that $t_1 \sqsupset t_2$. There is no gap between t_2 and t_1 with respect to Φ if and only if $sep(t_1, t_2)$ is an implicant of Φ.*

Proof. "If": If $sep(t_1, t_2)$ is an implicant of Φ, then every minterm m in $\mu(sep(t_1, t_2))$ is also an implicant of Φ.

"Only if": Follows from the previous lemma, since if $sep(t_1, t_2)$ is not an implicant of Φ, then there is a minterm of $sep(t_1, t_2)$ that is strictly below some minterm $m_1 \in \mu(t_1)$ and above some minterm $m_2 \in \mu(t_2)$. $\square$

Theorem 3. *Let Φ be an expression in DNF. Φ is convex if and only if for all distinct terms t_i, t_j in Φ such that $t_i \sqsupset t_j$, $sep(t_i, t_j)$ is an implicant of Φ.*

Proof. Follows as a result of Theorem 2 $\square$

Example 2. *Consider an excerpt of an educational system consisting of faculty who teach courses, students who enroll in courses, and the chairs of the department in which the respective courses are taught. In this example, we focus on the access controls of different users over an abstract permission with respect to a specific course which is controlling who can(not) attend the lectures for that course. In particular, in the context of our policy model framework, we consider the following variables whose English translation is provided next to them:*

- x_1 = *isTeachingCourse, which is* `True` *if a user is currently assigned to teach that course, else* `False`.
- x_2 = *isEnrolledIntoCourse, which is* `True` *if a user is currently enrolled into the given course, else* `False`.
- x_3 = *isRemoteAccess, which is* `True` *if a user is trying to access the lectures from outside the school campus, and* `False` *otherwise.*
- x_4 = *isDeptChair, which is* `True` *if a user is the chair of the department in which the course is currently being taught, else* `False`.

Based on the above conditions, suppose we create a sample policy Φ with three rules t_1, t_2, and t_3 in our educational system as follows:

$$\Phi = t_1 \vee t_2 \vee t_3, \text{ where}$$

$$t_1 = \overline{x_1}\, x_2 \overline{x_4} \qquad t_2 = \overline{x_1}\, \overline{x_3} x_4 \qquad t_3 = x_1 \overline{x_2}\, \overline{x_3}$$

Following are English translations for the three rules provided above:

- t_1 = *Students enrolled in the course can remotely attend (say, from their homes) the course's lectures.*
- t_2 = *A chair can attend the lectures for the courses in the department for which (s)he is the chair as long as they are present in the classroom when the course is in session.*

- t_3 = *A faculty member, irrespective of whether s(he) is the department chair or not, should be present in the classroom to attend (and present) the lectures for that course.*

Note that we do not introduce yet another variable corresponding to our abstract permission because if that were the case then that variable would have been present in all *the rules, and so that would have made the evaluation of that variable trivially redundant.*

Based on Theorem 3, we now determine whether policy Φ is convex.

- $t_2 \sqsupset t_1$ *because* $|\aleph(t_1, t_2)| = 0$ *and* $sep(t_2, t_1) = \overline{x_1}x_2\overline{x_3}$
- $t_1 \not\sqsupset t_2$ *because* $|\aleph(t_2, t_1)| = 1$
- $t_3 \sqsupset t_2$ *because* $|\aleph(t_2, t_3)| = 0$ *and* $sep(t_3, t_2) = \overline{x_2}\,\overline{x_3}x_4$
- $t_2 \not\sqsupset t_3$ *because* $|\aleph(t_3, t_2)| = 1$
- $t_3 \not\sqsupset t_1$ *because* $|\aleph(t_1, t_3)| = 1$
- $t_1 \not\sqsupset t_3$ *because* $|\aleph(t_3, t_1)| = 1$
- $sep(t_2, t_1) \land \neg\Phi$ *is unsatisfiable*
- $sep(t_3, t_2) \land \neg\Phi$ *is unsatisfiable*

Because all separators produce unsatisfiability, therefore Φ is convex. This means that we can represent the given Negation policy Φ in the DDDO model. The following are the corresponding DDDO rules:

- $\langle x_1, \texttt{PERMIT}\rangle$
- $\langle x_2, \texttt{PERMIT}\rangle$
- $\langle x_4, \texttt{PERMIT}\rangle$
- $\langle x_1x_2, \texttt{DENY}\rangle$
- $\langle x_1x_3, \texttt{DENY}\rangle$
- $\langle x_3x_4, \texttt{DENY}\rangle$

As one can observe from the below equation, the Negation policy (on the left) and the DDDO policy (on the right) produce equivalent expressions:

$$\Phi = \overline{x_1}\,x_2\overline{x_4} \lor \overline{x_1}\,\overline{x_3}x_4 \lor x_1\overline{x_2}\,\overline{x_3} = (x_1 \lor x_2 \lor x_4) \land \neg(x_1x_2 \lor x_1x_3 \lor x_3x_4)$$

Now that we have seen an example of a policy that is convex, next we provide another example of a policy Ψ that is very similar to the policy Φ shown in the above example, but which comes out to be non-convex based on Theorem 3 as we demonstrate below:

Example 3. *Suppose again we consider three rules t_1, t_2, and t_3, which are exactly the same as in Example 2, but with a minor modification on rule t_1 as follows:*

$$t_1 = \overline{x_1}x_2x_4$$

Therefore, our new Negation policy Ψ becomes:

$$\Psi = \overline{x_1}x_2x_4 \lor \overline{x_1}\,\overline{x_3}x_4 \lor x_1\overline{x_2}\,\overline{x_3}$$

In the following, we show that there is at least one separator that is not an implicant of Ψ, which in turn proves that Ψ is not *convex:*

- $t_2 \sqsupset t_1$ *because* $|\aleph(t_1, t_2)| = 0$ *and* $sep(t_2, t_1) = \overline{x_1}x_2\overline{x_3}$
- $t_1 \sqsupset t_2$ *because* $|\aleph(t_2, t_1)| = 0$ *and* $sep(t_1, t_2) = \overline{x_1}x_4$
- $t_2 \not\sqsupset t_3$ *because* $|\aleph(t_3, t_2)| = 1$
- $t_3 \sqsupset t_2$ *because* $|\aleph(t_2, t_3)| = 0$ *and* $sep(t_3, t_2) = \overline{x_2}\,\overline{x_3}x_4$
- $t_3 \not\sqsupset t_1$ *because* $|\aleph(t_1, t_3)| = 1$
- $t_1 \not\sqsupset t_3$ *because* $|\aleph(t_3, t_1)| = 1$
- $sep(t_1, t_2) \land \neg\Psi$ *is satisfiable*

Therefore, since Ψ is not convex, it cannot be expressed in the DDDO model, i.e., there is no set of permit and deny rules that can capture the authorizations produced by Ψ.

4 ALGORITHM

In this section, we will explore the practical implications of our two fundamental theorems by developing an algorithm for testing the convexity of policy semantics.

Algorithm 1 Convexity-Check

1: **Input:** A boolean expression Φ in DNF
2: **Output:** `True` if Φ is convex; otherwise `False`
3: **function** Convexity-Check(Φ)
4:     **for** t_1 in Φ **do**
5:         **for** t_2 in Φ **do**
6:             **if** $t_1 \neq t_2$ and Gap-Check(t_1,t_2) **then**
7:                 $sep(t_1, t_2) = \Omega(t_1, t_2)\, \mathcal{B}_2(t_1, t_2)\, \Gamma_1(t_1, t_2)$
8:                 **if** ($sep(t_1, t_2) \land \neg\Phi$) is satisfiable **then**
9:                     **return** `False`
10:                 **end if**
11:             **end if**
12:         **end for**
13:     **end for**
14:     **return** `True`
15: **end function**

Algorithm 2 Gap-Check

Input: Product terms t_1 and $t_2 \in \Phi$
Output: `True` if a gap between t_2 and t_1 exists; otherwise `False`
function Gap-Check(t_1,t_2)
    if $|\aleph(t_2, t_1)| > 0$ **then**
        return `False`
    else
        $c_1 = |\aleph(t_1, t_2)\, \mathcal{B}_1(t_1, t_2)|$
        $c_2 = |\aleph(t_1, t_2)\, \Gamma_2(t_1, t_2)|$
        $c_3 = |\mathcal{B}_1(t_1, t_2)\, \Gamma_2(t_1, t_2)|$
        return $\aleph(t_1, t_2) > 1 \lor (c_1 > 0) \lor (c_2 > 0) \lor (c_3 > 0)$
    end if
end function

Algorithm 1 is constructed upon the main result presented in Theorem 3 to validate the convexity property of the set of rules in the policy. Algorithm 1 efficiently examines each separator, providing a method to verify whether it is an implicant of Φ using an SAT solver (Line 8). This algorithm utilizes Algorithm 2 as a criterion to avoid computing separators that fail to meet the conditions specified in Theorem 1. The purpose of this criterion is to streamline the search for separators to be evaluated. In practice, it helps reduce the running time.

The complexity of Algorithm 1 is influenced by various factors. First, to compute each separator, the algorithm requires $O(n^2)$ comparisons in the worst-case scenario, where n represents the number of terms in the DNF. Additionally, each separator computation involves a Gap-Check criterion, which is linear in the number of variables. Furthermore, the algorithm involves solving the satisfiability problem (SAT) as a subroutine. Therefore, considering m as the maximum number of variables across all the terms in the formula, the overall complexity of Algorithm 1 can be expressed as $O(n^2 * (m + SAT))$.

Table 2: Characteristics of Policy Datasets (Each Rule Is a Conjunction of Variables)

Dataset	# Variables	# PERMIT Rules	# DENY Rules
healthDDDO	69	49	26
teachDDDO	49	39	14
paperDDDO	35	18	11
healthNeg	64	58	-
teachNeg	46	40	-
paperNeg	34	20	-

5 EXPERIMENTS

In this section, we present the experiments conducted to evaluate the performance of checking convexity for access control policies across three distinct datasets. These datasets contain rules consisting of a conditional expression and a decision. The two different types of decisions for each rule are `PERMIT` and `DENY`. The experiments aim to investigate various aspects of convex and non-convex policies, including:

(1) Efficiency of the SAT solver
(2) Their performance across different DDDO policies,
(3) Difference in computational complexity between convex and non-convex and
(4) Performance under a fixed number of rules.

Considering the novelty of our approach in this domain, there are currently no existing methods available to serve as a baseline for comparison. In the next sections, we start by providing details of our experimental setup and the datasets used therein.

5.1 Policy Datasets

We adopted policy datasets from Slankas et al. [35] who propose inferring access control rules from natural language texts such as requirements documents. The policies are from real-world systems across three domains – healthcare, conference management (paper review), and education (teaching). There are both `PERMIT` and `DENY` rules in those policies. In addition, there are rules with complemented conditions. We form DDDO rule sets using `PERMIT`/`DENY` rules with only uncomplemented conditions. We also form Negation rule sets using `PERMIT` rules that may have either complemented or uncomplemented conditions. Therefore, we create six reference rule sets, namely, `healthDDDO`, `healthNeg`, `paperDDDO`, `paperNeg`, `teachDDDO`, and `teachNeg`, considering the three systems and the type of policy model. Table 2 summarizes the characteristics of these rule sets.

Although the previous work considers both complemented conditions (in the form of negative adjectives and nouns such as unable, none, and nothing) and `DENY` rules (in the form of verbs and adverbs with negative connotations such as stop, prohibit, and never) within sentences, they only consider the broader notion of negativity or denying certain users or roles access to a specific action and/or resource. So, to adapt their inferred access control rules to our context of DDDO and Negation policies, we needed to explicitly distinguish the two notions of negativity. Furthermore, to be able to convert to our policy format, we identify the variables within their English rules so that our converted rule consists of the conjunction of identified variables. For instance, for the rule "Professors can change student grades", we create a permit rule consisting of three variables x_1="subject.isProfessor", x_2="action.isChange", and x_3="resource.isStudentGrades" (we use the terminologies *subject, action,* and *resource* in accordance with the previous work).

For our experiments, we formed policies of the expected number of rules by randomly selecting rules from each rule set, ensuring a diverse and representative set of rules. For experiments where we strictly needed Negation policies with convex semantics as our input, we first form DDDO policies (from DDDO rule sets) and then prepare the Negation version of those. For each DDDO policy, we first construct two DNFs: $\mathcal{A}$ containing the disjunction of conjunctive expressions in `PERMIT` rules, and similarly, $\mathcal{B}$ for `DENY` rules. Then, we construct $\mathcal{A} \wedge \neg\mathcal{B}$ and subsequently convert that to DNF using a Boolean algebra library for Python. The conjunctive expressions in the resulting DNF are the rules in the resulting Negation policy. Note that the number of rules in such policies will be much more than the figures reported in Table 2.

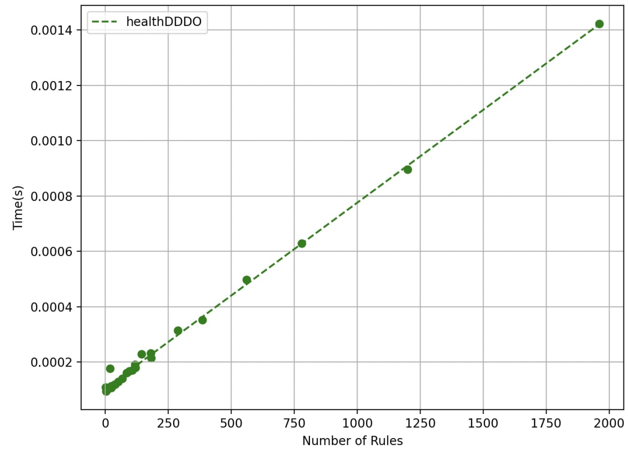

Figure 2: Performance of SAT Solver for Convex Policy

5.2 Implementation and Efficiency of the SAT Solver

We implemented Algorithm 1 in Python to verify the convexity of each input policy using Glucose SAT solver [4] as a subroutine. For all experiments, the performance of the algorithm is measured in seconds.

We first conduct an experiment to measure the performance of the SAT solver in the context of Algorithm 1. We generate policies from 2 up to 1900 rules based on the `healthDDDO` rule set. Figure 2 presents the performance of Algorithm 1 normalized by the number of times the SAT solver is invoked. The linear trend suggests that the SAT solver is highly efficient even when confronted with a substantial number of rules. This contributes to the performance of our implementation as discussed next.

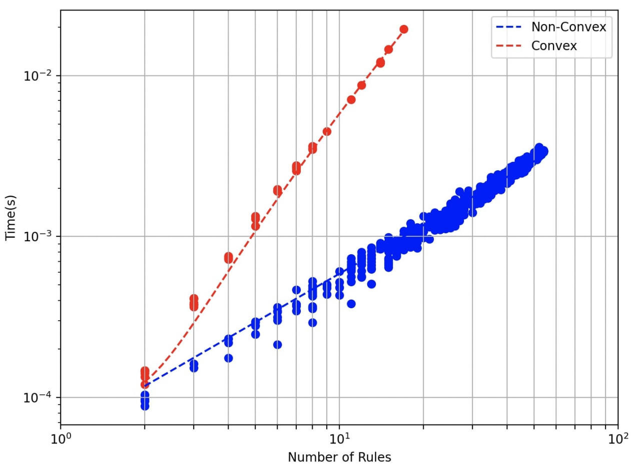

Figure 3: Performance for Convex vs. Non-Convex Policies (Using `healthNeg` Dataset)

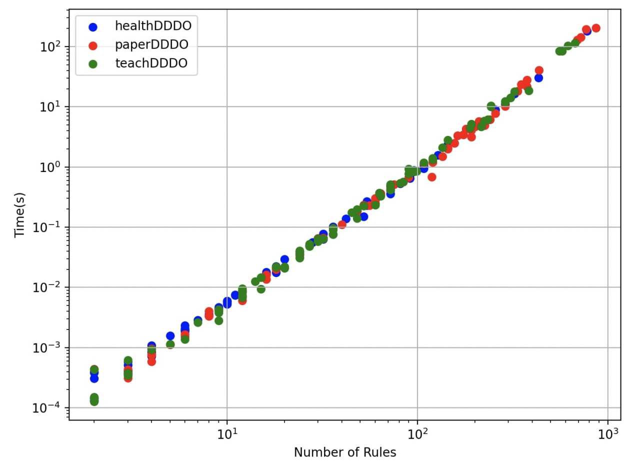

Figure 4: Performance for Convex Policies From Different Domains

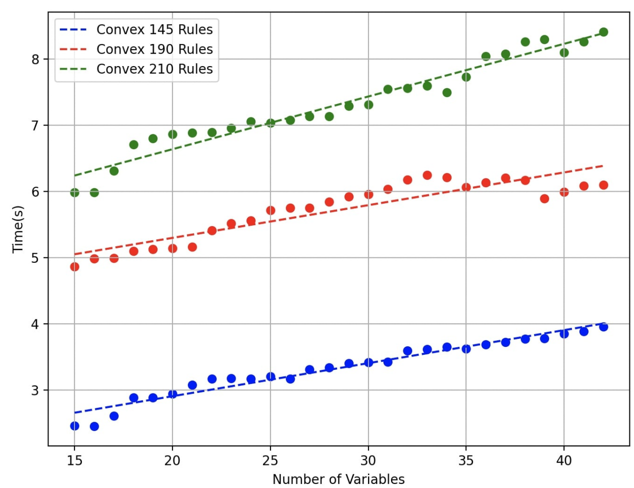

Figure 5: Impact of Rule Size (# of Variables) on Testing Convex Policies (Using `healthDDDO` Dataset)

5.3 Convex vs. Non-Convex Policies

In this experiment, we use the `healthNeg` rule set to generate Negation policies and test whether they have convex semantics (i.e., can be represented in DDDO) or not. The objective is to discern any notable difference in computational complexity between convex and non-convex policies.

The results presented in Figure 3 reveal the complexity differences between convex and non-convex policies using a log-log plot. The convex policies are quadratic in complexity relative to the number of rules and are illustrated as a line with a slope close to two, while the non-convex policies have linear complexity and are depicted by a line with a slope close to one. Note that the quadratic/linear behavior can be determined based on the slope of the line due to the log scale of the axes. The computational challenge of convex policies arises from the requirement to verify whether every separator between any two terms (rules) constitutes an implicant of the policy. The number of such separators is quadratic relative to the number of rules. Conversely, for non-convex policies, the test will be terminated as soon as the first non-implicant separator is checked, resulting in more efficient run time.

5.4 Performance on Different DDDO Policies

In this experiment, we investigate the impact of the characteristics of the datasets on the performance of our solution. Specifically, we test the performance of Algorithm 1 across convex policies generated from different rule sets. For this experiment, we used the DDDO datasets (`healthDDDO`, `teachDDDO`, and `paperDDDO`) to generate our policies to focus on the more computationally challenging policy cases. To ensure comprehensive analysis, we generated 80 policies for each dataset, restricting the number of rules to a range of 2 to 865.

Figure 4 illustrates that policies from all three rule sets exhibit very similar performance trends. This consistent quadratic performance across all three datasets suggests that the performance of the algorithm remains stable irrespective of the dataset used. The quadratic behavior is illustrated across the dataset as a straight line with a slope close to 2.

5.5 Impact of Rule Complexity

In this experiment, we investigate the impact of rule complexity, specifically, the number of variables used in a policy, on the performance of our solution. Our goal is to assess the performance as we change the number of variables in a policy while maintaining a fixed number of rules in the policy. We also ensure that the tested policies are convex. We first generated 3 policies with different rule sizes from the `healthDDDO` dataset. We then generated other policies by progressively eliminating the variables in each case while ensuring that each resulting policy remains convex.

The results depicted in Figure 5 reveals a consistent trend in the running time of our solution as the rule complexity is varied (by eliminating variables), regardless of the number of rules. It is evident that as the number of variables increases while holding the number of rules constant, the computational complexity exhibits a linear growth pattern. This phenomenon can be attributed to the proportional increase in the size of the separator that needs to be computed with the increment of variables.

6 RELATED WORK

Our policy model revolves around rule-based policies which consist of a set of rules to capture the authorizations of a system, based on the condition that all variables in at least one of the rules should evaluate to `True` for access to be permitted. Two contemporary rule-based policy models that have received great research attention include attribute-based access control (ABAC) [1, 14, 25, 26, 31–34] and relationship-based access control (ReBAC) [5, 10, 11, 15, 17, 18, 20, 28, 30, 36]. The ABAC model specifies which users can access what resources in terms of the attributes of the requesting user and the requested resource. The ReBAC model makes the authorization approach of ABAC model more flexible by taking into account the sequence of relationships between users and resources, where relationships are expressed as binary predicates instead of using unary predicates such as attributes and roles as in ABAC.

There are variations of ABAC and ReBAC models in the literature that include different combinations of complemented conditions and negative authorizations, depending on the use case application(s) considered in the respective works, to limit access for only certain types of users. Some works use only complemented conditions (which is the Negation model in our context), whereas some employ deny rules along with the deny-overrides conflict resolution strategy (which is the DDDO model in our context), and there are yet works that utilize both complemented conditions and deny rules for prohibiting access(es). In this work, we are not concerned with proposing yet another rule-based policy model, but rather interested in capturing the essential components of rule-based policies spanning the access control models proposed in the literature that is sufficient enough for demonstrating our convexity analysis. Moreover, the focus of this work is to present a systematic and efficient approach to convert policies specified in the Negation model to corresponding policies in the Deny model. Our motivation is that deny rules are usually much easier to apprehend and manage for a security administrator than complemented conditions, even though a previous work argues that using the latter construct produces a more expressive policy than using the former construct [24].

Later works focused on migrating or refactoring policies specified in a conventional access control model such as access control lists to a rule-based policy format. Such line of works, also famously known as *policy mining*, takes as input the low-level policy along with the required knowledge about the attributes of and relationships between users and resources, and proposes different approaches to extract high-level and concise rule-based policies that preserve the authorizations captured in the given low-level policy. The policy mining concept has been extensively explored with respect to both ABAC and ReBAC models. In the context of ABAC, researchers have proposed algorithms to mine concise rules in terms of the attributes of users and resources from given low-level authorizations such as access logs or access control lists [2, 12, 21, 37]. In the context of ReBAC, previous mining algorithms have focused on inferring rules in terms of the relationships between users and resources from given low-level policy and information about the entity relationships [6, 7, 22, 23]. The problem of migrating a security policy enforced in a particular device to be enforced in a different device has also been studied that considers differences in the computational capabilities of the devices [27].

In more recent literature, there are also works that consider the problem of determining the feasibility of mining ReBAC policies from the given authorizations and relationships data [8, 9]. Although at first glance this work seems closely related to our work, since we are also determining the feasibility of policy convertibility, a closer inspection will reveal that the input to our algorithm is a policy, which is a DNF Boolean expression in the Negation model and we are checking if we can represent that policy in the DDDO model. So, we are also considering the feasibility of conversion between two specific constructs within rule-based policies, which are complemented conditions and negative authorizations. Importantly, using our policy semantics framework, we are able to determine the feasibility of Negation to DDDO policy conversion in quadratic time.

Our work is also related to the expressiveness analysis of access control policies. As discussed in Sections 1 and 2, we build on a previously-proposed approach to policy semantics and expressiveness [29]. While that work focuses on capturing the semantics of various rule-based models and comparing their expressiveness, our contribution is on determining if a policy from a more expressive model (Negation) is convertible to a policy in a less expressive model (DDDO). Other notable related work in the area involves analyzing the expressiveness of XACML policies [16].

7 CONCLUSION

In this work, we proposed an empirical approach to test the convertibility of policies that use complemented conditions to those that use `DENY` rules. We formally characterized how a Negation policy semantics may be tested for convexity and therefore, being expressible using `DENY` rules (in the context of DDDO model). We also showed how that can be employed as a strategy to test convertibility by relying on existing SAT solver solutions. Our experimental results are promising, demonstrating the feasibility and scalability of the approach in the context of multiple policies. As part of our future work, we are planning to extend our approach to *derive* the `PERMIT` and `DENY` rules if the policy semantics is convex, i.e., the equivalent DDDO policy. Our key intuition is that the separators can help in dividing up the positive and negative parts of the individual terms which could be used to construct the DNF-equivalent of `PERMIT` and `DENY` rules.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments and suggestions. This material is based upon work partially supported by the National Science Foundation under Grant No. 2047623.

REFERENCES

[1] 2013. 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.

[2] Manar Alohaly, Hassan Takabi, and Eduardo Blanco. 2018. A deep learning approach for extracting attributes of ABAC policies. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. 137–148.
[3] Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: a modern approach*. Cambridge University Press.
[4] Gilles Audemard and Laurent Simon. 2018. On the Glucose SAT Solver. *Int. J. Artif. Intell. Tools* 27, 1 (2018), 1840001:1–1840001:25. https://doi.org/10.1142/S0218213018400018
[5] Glenn Bruns, Philip WL Fong, Ida Siahaan, and Michael Huth. 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*. 117–124.
[6] Thang Bui and Scott D Stoller. 2020. A Decision Tree Learning Approach for Mining Relationship-Based Access Control Policies. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*. 167–178.
[7] Thang Bui, Scott D Stoller, and Jiajie Li. 2019. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers & Security* 80 (2019), 317–333.
[8] Shuvra Chakraborty and Ravi Sandhu. 2021. Formal analysis of rebac policy mining feasibility. In *Proc. ACM CODASPY*. 197–207.
[9] Shuvra Chakraborty and Ravi Sandhu. 2021. On Feasibility of Attribute-Aware Relationship-Based Access Control Policy Mining. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 393–405.
[10] Yuan Cheng, Jaehong Park, and Ravi Sandhu. 2012. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *2012 International Conference on Privacy, Security, Risk and Trust*. IEEE, 646–655.
[11] Yuan Cheng, Jaehong Park, and Ravi Sandhu. 2014. Attribute-aware relationship-based access control for online social networks. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 292–306.
[12] Carlos Cotrini, Thilo Weghorn, and David Basin. 2018. Mining ABAC rules from sparse logs. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 31–46.
[13] Yves Crama and Peter L. Hammer. 2011. *Boolean Functions - Theory, Algorithms, and Applications*. Encyclopedia of mathematics and its applications, Vol. 142. Cambridge University Press.
[14] Jason Crampton and Charles Morisset. 2012. PTaCL: A language for attribute-based access control in open systems. In *International Conference on Principles of Security and Trust*. Springer, 390–409.
[15] Jason Crampton and James Sellwood. 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*. 187–198.
[16] Jason Crampton and Conrad Williams. 2016. On completeness in languages for attribute-based access control. In *Proc. 21st ACM Symposium on Access Control Models and Technologies*. 149–160.
[17] Philip WL Fong. 2011. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and application security and privacy*. 191–202.
[18] Philip WL Fong and Ida Siahaan. 2011. Relationship-based access control policies and their policy languages. In *Proc. 16th ACM Symposium on Access Control Models and Technologies*. 51–60.
[19] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
[20] Hongxin Hu and Gail-Joon Ahn. 2011. Multiparty authorization framework for data sharing in online social networks. In *IFIP Annual Conf. Data and Applications Security and Privacy*. Springer, 29–43.
[21] Padmavathi Iyer and Amirreza Masoumzadeh. 2018. Mining positive and negative attribute-based access control policy rules. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. 161–172.
[22] Padmavathi Iyer and Amirreza Masoumzadeh. 2019. Generalized Mining of Relationship-Based Access Control Policies in Evolving Systems. In *Proc. 24th ACM Symposium on Access Control Models and Technologies*. ACM, 135–140.
[23] Padmavathi Iyer and Amirreza Masoumzadeh. 2020. Active learning of relationship-based access control policies. In *Proc. 25th ACM Symposium on Access Control Models and Technologies*. 155–166.
[24] Padmavathi Iyer, Amirreza Masoumzadeh, and Paliath Narendran. 2022. On the Expressive Power of Negated Conditions and Negative Authorizations in Access Control Models. *Computers & Security* 116 (2022), 102586.
[25] Savith Kandala, Ravi Sandhu, and Venkata Bhamidipati. 2011. An attribute based framework for risk-adaptive access control models. In *2011 Sixth International Conference on Availability, Reliability and Security*. IEEE, 236–241.
[26] Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. 2009. A flexible attribute based access control method for grid computing. *Journal of Grid Computing* 7, 2 (2009), 169.
[27] Jorge Lobo, Elisa Bertino, and Alessandra Russos. 2020. On security policy migrations. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*. 179–188.
[28] Amirreza Masoumzadeh and James Joshi. 2010. OSNAC: An ontology-based access control model for social networking systems. In *2nd International Conference on Social Computing*. IEEE, 751–759.
[29] Amirreza Masoumzadeh, Paliath Narendran, and Padmavathi Iyer. 2021. Towards a Theory for Semantics and Expressiveness Analysis of Rule-Based Access Control Models. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies* (New York, NY, USA, 2021-06-11) *(SACMAT '21)*. Association for Computing Machinery, 33–43. https://doi.org/10.1145/3450569.3463569
[30] Edelmira Pasarella and Jorge Lobo. 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*. 91–102.
[31] Daniel Servos and Sylvia L Osborn. 2014. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *International Symposium on Foundations and Practice of Security*. Springer, 187–204.
[32] Daniel Servos and Sylvia L Osborn. 2017. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)* 49, 4 (2017), 1–45.
[33] Daniel Servos and Sylvia L Osborn. 2018. HGAA: An Architecture to Support Hierarchical Group and Attribute-Based Access Control. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*. 1–12.
[34] Hai-bo Shen and Fan Hong. 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, 74–79.
[35] John Slankas, Xusheng Xiao, Laurie Williams, and Tao Xie. 2014. Relation extraction for inferring access control rules from natural language artifacts. In *Proceedings of the 30th annual computer security applications conference*. 366–375.
[36] Scott D Stoller. 2015. An administrative model for relationship-based access control. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 53–68.
[37] Zhongyuan Xu and Scott D Stoller. 2014. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing* 12, 5 (2014), 533–545.

A COMPUTATIONAL COMPLEXITY OF CONVERTIBILITY OF NEGATION TO DDDO

In this section, we briefly discuss and prove the computational complexity of checking whether a Negation policy is convertible to a DDDO policy. We note this discussion as out of the scope of this paper and is provided only to support our motivation argument in Section 1.

Semantically, the decision problem of convertibility from Negation to DDDO is about whether the given set of minterms implemented by a Negation policy is *convex*. We show that this problem is computationally hard, and more specifically, co-NP-complete.

We first show that the problem is in co-NP. Given a Negation policy p_n that is not convertible to DDDO, there exists a certificate which consists of 3 minterms $m_1 \leq m_2 \leq m_3$ where $\{m_1, m_3\} \subseteq \mu(p_n)$ and $m_2 \notin \mu(p_n)$. We can verify the correctness of the certificate in polynomial time. The verification will take $O(|p|)$ time by checking each given minterm against the conditions of all policy rules in the worst case.

We prove the co-NP-hardness of the problem by showing that the validity problem [3, 19] can be polynomially reduced to Negation-DDDO convertibility. The validity problem is to determine whether a given propositional formula evaluates to true for every assignment of truth values to its variables. Let Ψ be a propositional formula in DNF (disjunctive normal form) over variables in set Y, i.e., $\Psi = \Pi_1 \vee \ldots \vee \Pi_k$ where each Π_i is a product of variables in Y or their complements, e.g., $y_1\overline{y_2}y_3$ (assuming $\{y_1, y_2, y_3\} \subseteq Y$). We define set of conditions $\mathcal{X} = \{x_1, x_2\} \cup Y$, and form negation policy p_n based on it as follows:

$$\langle x_1 x_2, \mathtt{PERMIT}\rangle$$
$$\langle \overline{x_1}\,\overline{x_2}, \mathtt{PERMIT}\rangle$$
$$\langle \Psi, \mathtt{PERMIT}\rangle$$

Note that since Ψ is in DNF format, the last rule above should be expanded into multiple rules with conditional expressions of purely conjunctive form: $\langle \Pi_i, \texttt{PERMIT} \rangle$. However, we keep it as above for the brevity of the discussion.

We now show that p_n is convertible to DDDO if and only if Ψ is valid. Let us consider the case that Ψ is valid (always equivalent to true). In this case, the corresponding rule in p_n always results in PERMIT. In other words, $\mu(p_n) = \mathcal{M}_X$ (set of minterms over conditions in X). Since the policy-minterms will be convex, we can express p_n in DDDO.

Now, consider the case where Ψ is not valid. In this case, there will exist a minterm $m \in \mathcal{M}_Y$ that does not belong to $\mu(\Psi)$, i.e., Ψ evaluates to false on the truth assignment corresponding to m. Given the abovementioned first and second rules in p_n, we will have: $\overline{x_1}\,\overline{x_2}m \in \mu(p_n)$ and $x_1 x_2 m \in \mu(p_n)$. However, $x_1\overline{x_2}m \notin \mu(p_n)$ since none of the rules in p_n can authorize that minterm. Since, $\overline{x_1}\,\overline{x_2}m \leq x_1\overline{x_2}m \leq x_1 x_2 m$, we conclude that $\mu(p_n)$ will not be convex, and thus, it cannot be expressed in DDDO. This concludes our reduction and proves the co-NP-hardness of the Negation-DDDO convertibility problem.