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 expressive 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

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 \(\text{Var}(t)\) and \(\text{Lit}(t)\), 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 \textsc{permit}. In the DDDO model, terms may only include uncomplemented variables. But each rule effect may be either \textsc{permit} or \textsc{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 \(M_X\). 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 \(m_2\) include the positive literals of \(m_1\). For example, \(m_1 \geq m_2\) given \(X = \{x_1, x_2, x_3, x_4\}\), \(m_1 = x_1 \overline{x_2} x_3 x_4\), and \(m_2 = \overline{x_1} \overline{x_2} x_3 x_4\). 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, \(t_1 \geq m_1|_{\text{Var}(t_1)}\) where \(t_1 = x_1 x_2\).

The semantics of a term \(t\), denoted by \(\mu(t)\) is the set of all minterms that are implicants of \(t\). For example, assuming \(X = \{x_1, x_2, x_3\}\), \(\mu(x_1 \overline{x_3}) = \{x_1 x_2 \overline{x_3}, x_1 x_2 \overline{x_3}, x_1 \overline{x_2} x_3\}\). The semantics of a policy, also denoted by \(\mu(P)\), 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 \(\Phi\) 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 \(M\): \(m_1 \leq m \leq m_2\) \(\subseteq M\).

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 \(P_X\) to DDDO policy \(P_D\) can be viewed as testing the convexity of the set of minterms \(\mu(P_X)\).

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 separator, 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 \(t_1\).
A Definition

Consists of negative literals that appear in \( x_\beta \) 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 \( |N(t_2,t_1)| = 0 \).

Table 1: Term Comparison Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Omega(t_1,t_2) )</td>
<td>Consists of literals that are common to both terms (i.e., their common factor).</td>
</tr>
<tr>
<td>( \mathcal{N}(t_1,t_2) )</td>
<td>Consists of variables that appear uncomplemented in ( t_1 ) and complemented in ( t_2 ).</td>
</tr>
<tr>
<td>( \mathcal{B}_1(t_1,t_2) )</td>
<td>Consists of positive literals that appear in ( t_1 ) whose variables do not appear in ( t_2 ).</td>
</tr>
<tr>
<td>( \mathcal{B}_2(t_1,t_2) )</td>
<td>Consists of positive literals that appear in ( t_2 ) whose variables do not appear in ( t_1 ).</td>
</tr>
<tr>
<td>( \Gamma_1(t_1,t_2) )</td>
<td>Consists of negative literals that appear in ( t_1 ) whose variables do not appear in ( t_2 ).</td>
</tr>
<tr>
<td>( \Gamma_2(t_1,t_2) )</td>
<td>Consists of negative literals that appear in ( t_2 ) whose variables do not appear in ( t_1 ).</td>
</tr>
</tbody>
</table>

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 \( |N(t_2,t_1)| = 0 \).

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. \( \alpha = \mathcal{N}(t_1,t_2) \),
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:

\[ t_1 = \omega \alpha \beta_1 \gamma_1 \]
\[ t_2 = \omega \alpha \beta_2 \gamma_2 \]

A better depiction of this would be:

\[ t_1 = \omega \alpha \beta_1 \gamma_1 \]
\[ t_2 = \omega \alpha \beta_2 \gamma_2 \]

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

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 \( \alpha, \beta_1, \) etc., we evaluate the functions in both directions.

\[ t_1 = x_1 x_2 x_3 x_5 \]
\[ t_2 = x_1 x_3 x_5 \]

Observe that

\[ \Omega(t_1,t_2) = \{ x_1 \} \]
\[ \mathcal{N}(t_1,t_2) = \{ x_5 \} \]
\[ \mathcal{B}_1(t_1,t_2) = \{ x_3 \} \]
\[ \mathcal{B}_2(t_1,t_2) = \{ x_4 \} \]
\[ \Gamma_1(t_1,t_2) = \{ x_2 \} \]
\[ \Gamma_2(t_1,t_2) = \{ x_5 \} \]

Note that \( \Omega \) 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 \( t > m|_{\bar{V}_{ar}(t)} \).

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

If \( t > m|_{\bar{V}_{ar}(t)} \), then let \( m'' = m|_{\bar{V}_{ar}(t)} \), i.e., write \( m \) as \( m' m'' \) where

\[ m' = m|_{\bar{V}_{ar}(t)} \]
\[ m'' = m|_{\bar{V}_{ar}(t)} \]

and horizontal bars stand for algebraic division. Then \( m'' > m \). □

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 \( m \not\in \mu(t) \). Then \( m \) leads \( t \) if and only if \( m|_{\text{Var}(t)} > t \).

Proof. Note that \( t \) is a subterm of every term of \( \mu(t) \). If \( t = m|_{\text{Var}(t)} \), 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 of \( \mu(t) \).

If \( m|_{\text{Var}(t)} > t \), then let \( m'' = m|_{\text{Var}(t)} \), i.e., write \( m \) as \( m'm'' \) where
\[
m' = \frac{m}{m|_{\text{Var}(t)}},
m'' = m|_{\text{Var}(t)}\text{X} \cdot VAr(t)
\]
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 \( \omega \beta \gamma \) must be a subterm of \( m \).

Proof. We split the proof into 3 cases, comparing \( m \) with \( \omega, \beta, \gamma \) and \( y_1 \).

Case (a): Consider \( t_1, t_2 \) and \( m \) restricted to variables of \( \omega, \beta, \gamma \), 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 \( \beta, \gamma \), i.e.,
\[
t_1|_{VAr(\beta)} = t_2|_{VAr(\beta)} = \beta
\]
If \( m \) leads \( t_2 \) then there cannot be a complemented variable in \( m|_{VAr(\beta)} \) which appears uncomplemented in \( t_1|_{VAr(\beta)} \). Besides, there are no complemented variables in \( \beta \). Thus we have that
\[
\beta = m|_{VAr(\beta)}.
\]

Case (c): Consider \( t_1, t_2 \) and \( m \) restricted to variables of \( \gamma \). Similar to the previous case,
\[
t_1|_{VAr(\gamma)} = \gamma, \quad t_2|_{VAr(\gamma)} = \gamma
\]
Since \( t_1 \) dominates \( m \) there cannot be a variable \( x \) complemented in \( t_1|_{VAr(\gamma)} \) but uncomplemented in \( m|_{VAr(\gamma)} \). Besides, by definition, there are no uncomplemented variables in \( \gamma \). Then we have that
\[
\gamma = m|_{VAr(\gamma)}.
\]
We later denote the term \( \omega \beta \gamma \) as the separator of \( t_1 \) from \( t_2 \).

Lemma 4. Suppose \( t_1 \nparallel t_2 \). Then
(a) If \( |\alpha \beta_1 \gamma| = 0 \), then there is no gap between \( t_2 \) and \( t_1 \).
(b) If \( |\alpha \gamma| = 0 \), 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 \( |\alpha \beta_1 \gamma| = 0 \), then
\[
t_1 = \omega \gamma_1 \beta_2 \gamma_2,
t_2 = \omega \beta_2 \gamma_2
\]
By Lemma 3, \( \omega \beta_2 \gamma \) 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 \( |\alpha \gamma| = 0 \), 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 \( t_1 \nparallel t_2 \) if and only if
\[
|\alpha \beta_1 \gamma| > 0 \vee |\beta_1 \gamma_2| > 0 \vee |\alpha \gamma| > 0.
\]

Proof. Suppose \( |\alpha \beta_1 \gamma| = 1 \) and \( |\beta_1 \gamma_2| \leq 1 \) and \( |\alpha \gamma| > 1 \) 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 \( |\alpha \beta_1 \gamma| = 1 \) and \( |\alpha \gamma| = 1 \). Thus \( |\alpha \gamma| = 1 \), for otherwise \( |\beta_1 \gamma_2| = 2 \). It follows that \( |\beta_1 \gamma_2| = 0 \).

By Lemma 3, \( \omega \beta_2 \gamma \) is a subterm of \( m \). Let \( \alpha = x \) where \( x \) is a variable. Thus \( \alpha = \alpha \).

\[
t_1 = \omega \alpha \gamma_1 \gamma_2,
t_2 = \omega \beta_2 \gamma_2
\]
We need to consider two cases: (a) \( x \) appears in \( m \), and (b) \( \alpha \) appears in \( m \).

Case (a): Here \( \omega \alpha \beta_2 \gamma_1 \gamma_2 \) is a subterm of \( m \). Thus, \( m|_{VAr(t_1)} = \omega \alpha \gamma_1 \gamma_2 = t_1 \). But for \( t_1 \) to dominate \( m \), by Lemma 1 we must have that \( t_1 > m|_{VAr(t_1)} \).

Case (b): \( \omega \alpha \beta_2 \gamma_1 \gamma_2 \) 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 \( m|_{VAr(t_2)} = \omega \beta_2 \gamma_2 = t_2 \).

Lemma 6. If \( |\alpha \gamma| > 1 \) and \( t_1 \nparallel t_2 \) then there is a gap between \( t_2 \) and \( t_1 \), i.e., there is a minterm \( m \not\in \mu(t_1) \cup \mu(t_2) \) such that \( t_1 \) dominates \( m \) and \( m \) leads \( t_2 \).

Proof. Let \( x_1 \) and \( x_2 \) be two variables in \( \alpha \), i.e., \( x_1, x_2 \) is a subterm of \( t_1 \) and \( x_1, x_2 \) is a subterm of \( t_2 \). Let \( \alpha = x_1, x_2, \eta \) where \( \eta \) is the rest of \( \alpha \), i.e., the conjunction of the remaining variables in \( \alpha \). Clearly \( \mu(t_1) \) and \( \mu(t_2) \) are disjoint. Besides, \( x_1, x_2, \eta \) is not a subterm of any of the minterms in \( \mu(t_1) \cup \mu(t_2) \). Now consider the term
\[
t = \omega x_1 \eta \beta_1 \beta_2 \gamma_1 \gamma_2 \text{ and let } m \in \mu(t).
\]
Note that
\[
m|_{VAr(t_1)} = \omega x_1, \eta \beta_1 \beta_2 \gamma_1 \gamma_2 < \omega x_1, \eta \beta_1 \gamma_1 = t_1,
\]
by Lemma 1 we must have that \( t_1 \) dominates \( m \). Similarly, we have that
\[
m|_{VAr(t_2)} = \omega x_1, \eta \beta_2 \gamma_2 > \omega x_1, \eta \beta_2 \gamma_2 = t_2,
\]
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 \( t_1 \nparallel t_2 \) if and only if
\[
|\alpha \gamma| > 1 \vee \text{any two of } |\alpha|, |\beta_1|, |\gamma_2| \text{ are greater than 0}.
\]
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): $|\gamma| > 1$, $|\beta_2| > 0$.

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

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

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

Case (b): $|\gamma| = 1$, $|\beta_2| > 0$.

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

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

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

Case (c): $|\gamma| = 0$, $|\beta_2| > 0$.

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

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

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

\[\square\]

3.2 Gap in an Expression

In the previous section, we only considered gaps between two terms without considering any larger expression that 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 $\Phi$, we say that there is a gap between $t_1$ and $t_2$ with respect to $\Phi$ if and only if there exist minterms $m_1 > m > m_2$ such that

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

If $t_1$ and $t_2$ are part of $\Phi$ then we only need to say "$m$ is not a part of $\Phi$".

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

We introduce the following notation:

$$\text{sep}(t_1, t_2) = \Omega(t_1, t_2) \mathcal{B}(t_1, t_2) \bar{t}_1(t_1, t_2)$$

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 $\Phi$ be an expression in DNF and $t_1$ and $t_2$ be terms in $\Phi$ such that $t_1 \supseteq t_2$. Let $s = \text{sep}(t_1, t_2)$. 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 $\text{Var}(t_1) = \text{Var}(\omega) \cup \text{Var}(\alpha) \cup \text{Var}(\beta_1) \cup \text{Var}(\gamma_1)$. Since $\omega \gamma_1$ is a subterm of $m |_{\text{Var}(t_1)}$ 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 $t_1 \beta_2 > m |_{\text{Var}(t_1 \beta_2)}$. Similarly, there must be a variable $\tilde{\alpha} \gamma_2$ (in $t_2$) that appears uncomplemented in $m$. Thus $t_2 \gamma_3 < m |_{\text{Var}(t_1 \gamma_3)}$.

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

$$s_1 > m |_{\text{Var}(s_1)}$$

and the result follows.

\[\square\]

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

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

"Only if": Follows from the previous lemma, since if $\text{sep}(t_1, t_2)$ is not an implicant of $\Phi$, then there is a minterm of $\text{sep}(t_1, t_2)$ 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 $\Phi$ be an expression in DNF. $\Phi$ is convex if and only if for all distinct terms $t_1, t_2$ in $\Phi$ such that $t_1 \supseteq t_2$, $\text{sep}(t_1, t_2)$ is an implicant of $\Phi$.

Proof. Follows as a result of Theorem 2.

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 $\Phi$ with three rules $t_1, t_2,$ and $t_3$ in our educational system as follows:

$$\Phi = t_1 \lor t_2 \lor t_3,$$

where

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

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

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 \( \Phi \) is convex:

- \( t_2 \not\subset t_1 \) because \( \{t_1, t_2\} = 0 \) and \( \text{sep}(t_2, t_1) = \overline{x_1}x_2x_3 \)
- \( t_1 \not\subset t_2 \) because \( \{t_2, t_1\} = 1 \)
- \( t_3 \not\subset t_2 \) because \( \{t_2, t_3\} = 0 \) and \( \text{sep}(t_3, t_2) = \overline{x_2}x_3x_4 \)
- \( t_2 \not\subset t_3 \) because \( \{t_3, t_2\} = 1 \)
- \( t_3 \not\subset t_1 \) because \( \{t_1, t_3\} = 1 \)
- \( \text{sep}(t_2, t_1) \land \neg \Phi \) is unsatisfiable
- \( \text{sep}(t_3, t_2) \land \neg \Phi \) is unsatisfiable

Because all separators produce unsatisfiability, therefore \( \Phi \) is convex.

This means that we can represent the given Negation policy \( \Phi \) in the DDDO model. The following are the corresponding DDDO rules:

- \((x_1, \text{PERMIT})\)
- \((x_2, \text{PERMIT})\)
- \((x_3, \text{PERMIT})\)
- \((x_1x_2, \text{DENY})\)
- \((x_1x_3, \text{DENY})\)
- \((x_3x_4, \text{DENY})\)

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

\[
\Phi = \overline{x_1}x_2x_4 \lor \overline{x_1}x_3x_4 \lor x_1 \overline{x_2} \overline{x_3} = (x_1 \lor x_2 \lor x_3) \land \neg (x_1x_2 \lor x_1x_3 \lor x_3x_4)
\]

Now that we have seen an example of a policy that is convex, next we provide another example of a policy \( \Psi \) that is very similar to the policy \( \Phi \) 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 \( t_3, \) as follows:

\( t_1 = \overline{x_1}x_2x_4 \)

Therefore, our new Negation policy \( \Psi \) becomes:

\[
\Psi = \overline{x_1}x_2x_4 \lor \overline{x_1}x_3x_4 \lor x_1 \overline{x_2} \overline{x_3}
\]

In the following, we show that there is at least one separator that is not an implicant of \( \Psi, \) which in turn proves that \( \Psi \) is not convex:

- \( t_2 \not\subset t_1 \) because \( \{t_1, t_2\} = 0 \) and \( \text{sep}(t_2, t_1) = \overline{x_1}x_2x_3 \)
- \( t_1 \not\subset t_2 \) because \( \{t_2, t_1\} = 1 \)
- \( t_2 \not\subset t_3 \) because \( \{t_3, t_2\} = 1 \)
- \( t_3 \not\subset t_2 \) because \( \{t_2, t_3\} = 0 \) and \( \text{sep}(t_3, t_2) = \overline{x_2}x_3x_4 \)
- \( t_2 \not\subset t_3 \) because \( \{t_3, t_2\} = 1 \)
- \( t_1 \not\subset t_3 \) because \( \{t_3, t_1\} = 1 \)
- \( \text{sep}(t_2, t_1) \land \neg \Psi \) is satisfiable

Therefore, since \( \Psi \) 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 \( \Psi. \)

### 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 \( \Phi \) in DNF
2. **Output:** True if \( \Phi \) is convex; otherwise False
3. **function** \( \text{CONVEXITY-CHECK}(\Phi) \)
   4. for \( t_1 \) in \( \Phi \) do
   5. for \( t_2 \) in \( \Phi \) do
   6. if \( t_1 \neq t_2 \) and \( \text{Gap-Check}(t_1, t_2) \) then
   7. \( \text{sep}(t_1, t_2) = \Omega(t_1, t_2) \Gamma(t_1, t_2) \Gamma(t_1, t_2) \)
   8. if \( \text{sep}(t_1, t_2) \land \neg \Phi \) 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**

1. **Input:** Product terms \( t_1 \) and \( t_2 \in \Phi \)
2. **Output:** True if a gap between \( t_2 \) and \( t_1 \) exists; otherwise False
3. **function** \( \text{Gap-Check}(t_1, t_2) \)
4. if \( \{t_2, t_1\} > 0 \) then
5. return False
6. else
7. \( c_1 = |\{t_1, t_2\} \cap |t_1| = |t_2| \}
8. \( c_2 = |\{t_1, t_2\} \cap |t_1| = |t_2| \}
9. \( c_3 = |\{t_1, t_2\} \cap |t_1| = |t_2| \}
10. return \( N(t_1, t_2) > 1 \lor (c_1 > 0) \lor (c_2 > 0) \lor (c_3 > 0) \)
11. end if
12. 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 \( \Phi \) 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 \( O(n^3) \) 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 \( O(n^2 \times (m + SAT)) \).
We adopted policy datasets from Slankas et al. [35] who propose converting to our policy format, we identify the variables within their 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 sentences, they only consider the broader notion of negativity or none, and nothing) and conditions (in the form of negative adjectives and nouns such as unable, or uncomplemented conditions. Therefore, we create six reference 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.

![Figure 2: Performance of SAT Solver for Convex Policy](image)

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Variables</th>
<th># PERMIT Rules</th>
<th># DENY Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>healthDDDO</td>
<td>69</td>
<td>49</td>
<td>26</td>
</tr>
<tr>
<td>teachDDDO</td>
<td>49</td>
<td>39</td>
<td>14</td>
</tr>
<tr>
<td>paperDDDO</td>
<td>35</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>healthNeg</td>
<td>64</td>
<td>58</td>
<td>-</td>
</tr>
<tr>
<td>teachNeg</td>
<td>46</td>
<td>40</td>
<td>-</td>
</tr>
<tr>
<td>paperNeg</td>
<td>34</td>
<td>20</td>
<td>-</td>
</tr>
</tbody>
</table>

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 \(x_1 = \text{subject.isProfessor}\), \(x_2 = \text{action.isChange}\), and \(x_3 = \text{resource.isStudentGrades}\) (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: \(A\) containing the disjunction of conjunctive expressions in PERMIT rules, and similarly, \(B\) for DENY rules. Then, we construct \(A \land \neg B\) 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.

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.
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

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 \( m_1 \perp m_2 \perp m_3 \) where \( m_1 \perp m_2 \perp m_3 \). We can verify the correctness of the certificate in polynomial time. The verification will take \( O(|p_N|) \) 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 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 \( m_1 \perp m_2 \perp m_3 \) where \( m_1 \perp m_2 \perp m_3 \). We can verify the correctness of the certificate in polynomial time. The verification will take \( O(|p_N|) \) 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 to DDDO convertibility. The validity problem is to determine whether a given minterm \( m \) 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 \( m_1 \perp m_2 \perp m_3 \) where \( m_1 \perp m_2 \perp m_3 \). We can verify the correctness of the certificate in polynomial time. The verification will take \( O(|p_N|) \) 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 to DDDO convertibility. The validity problem is to determine whether a given minterm \( m \) 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 \( m_1 \perp m_2 \perp m_3 \) where \( m_1 \perp m_2 \perp m_3 \). We can verify the correctness of the certificate in polynomial time. The verification will take \( O(|p_N|) \) 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 to DDDO convertibility. The validity problem is to determine whether a given minterm \( m \) is computationally hard, and more specifically, co-NP-complete.
conjunctive form: \((\Pi_i, \text{PERMIT})\). 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 \(\Psi\) is valid. Let us consider the case that \(\Psi\) is valid (always equivalent to true). In this case, the corresponding rule in \(p_n\) always results in \text{PERMIT}. In other words, \(\mu(p_n) = M_X\) (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 \(\Psi\) is not valid. In this case, there will exist a minterm \(m \in M_Y\) that does not belong to \(\mu(\Psi)\), i.e., \(\Psi\) evaluates to false on the truth assignment corresponding to \(m\). Given the abovementioned first and second rules in \(p_n\), we will have: \(\overline{x_1 x_2 m} \in \mu(p_n)\) 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, \(\overline{x_1 x_2 m} \leq x_1 \overline{x_2 m} \leq x_1 x_2 m\), 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.