Efficient Extended ABAC Evaluation

Charles Morisset
Newcastle University
charles.morisset@newcastle.ac.uk

Tim A. C. Willemse
Eindhoven University of Technology
t.a.c.willemse@tue.nl

Nicola Zannone
Eindhoven University of Technology
n.zannone@tue.nl

ABSTRACT

A main challenge of attribute-based access control (ABAC) is the handling of missing information. Several studies show that the way standard ABAC mechanisms (*e.g.*, XACML) handle missing information is flawed, making ABAC policies vulnerable to attribute-hiding attacks. Recent work addressed the problem of missing information in ABAC by introducing the notion of extended evaluation, where the evaluation of a query considers all possible ways of extending that query. This method counters attribute-hiding attacks, but a naïve implementation is intractable, as it requires an evaluation of the whole query space. In this paper, we present an efficient extended ABAC evaluation method that relies on the encoding of ABAC policies as multiple Binary Decision Diagrams (BDDs), and on the specification of query constraints to avoid including the evaluation of queries that do not represent a valid state of the system. We illustrate our approach on two real-world case studies, which would be intractable with the original method and are analyzed in seconds with our method.

CCS CONCEPTS

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

KEYWORDS

Policy evaluation, ABAC; missing attributes; PTaCL; BDD

ACM Reference Format:
Charles Morisset, Tim A. C. Willemse, and Nicola Zannone. 2018. Efficient Extended ABAC Evaluation. In *SACMAT '18: The 23rd ACM Symposium on Access Control Models & Technologies (SACMAT), June 13–15, 2018, Indianapolis, IN, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3205977.3205980

1 INTRODUCTION

Attribute-Based Access Control (ABAC) is emerging as the de facto paradigm for the specification and enforcement of access control policies. In ABAC, policies and access requests are defined in terms of attribute name-value pairs. This provides an expressive, flexible and scalable paradigm that is able to capture and manage authorizations in complex environments.

Despite providing a powerful paradigm for the specification and evaluation of access control policies, ABAC has some intrinsic drawbacks related to the handling of missing information [3, 16]. As shown in [4], these drawbacks make ABAC policies vulnerable to attribute hiding attacks where users can obtain a more favorable decision by hiding some of their attributes. The main problem lies in the fact that all the information necessary for policy evaluation should be available to the policy decision point.

Recent years have seen the emergence of authorization mechanisms that go beyond the view of a centralized monitor with full knowledge of the system. Authorization mechanisms increasingly rely on external services to gather the information necessary for access decision making (*e.g.*, Amazon Web Services rely on third-party identity providers and federated identity systems, the OAuth 2.0 protocol enables delegation of authorization). The use of external services makes it difficult to guarantee and, in some cases, even to check that all necessary information has been provided. Moreover, in some domains like IoT, it might be difficult and costly to gather (accurate) information needed for policy evaluation. Missing information can significantly influence query evaluation and pose significant risks to a large range of modern systems.

eXtensible Access Control Markup Language (XACML) [14], the de-facto standard for attribute-based access control, provides a mechanism to deal with missing attributes, but Crampton et al. [5] showed that the evaluation of a query can yield a decision that does not necessarily provide an intuitive interpretation on whether access should be granted or not due to the fact that some information needed for the evaluation might be missing. They subsequently proposed a novel approach that allows for an extended evaluation of ABAC policies. In a nutshell, they suggest that the evaluation of a given query is calculated using the evaluation of all queries that can be constructed from that query. However, their approach requires exploring the state space for all possible queries, which is exponential in the number of attribute values, and therefore not particularly efficient.

In this paper, we present a new approach for the extended evaluation of ABAC policies that addresses this drawback. We first encode ABAC policies using binary decision diagram (BDD)-based data structures, which provide a compact encoding for storing the decisions for each query and for efficient policy evaluation [1, 8, 9]. We then propose a new method to compute the extended evaluation directly on the BDD structure, taking into account *query constraints*, which are used to exclude those queries that are not possible within the system from the query space. We demonstrate our approach on two complex case studies, where a naïve approach would deal with a query space comprising several millions of states, whereas our approach compiles in a few seconds a compact decision diagram.

The remainder of this paper is structured as follows. The next section presents preliminaries on ABAC and the notion of extended evaluation. Section 3 introduces a motivating example and provides a formulation of the problem. Section 4 presents the notion of query constraints. Section 5 presents a novel algorithm to compute the extended query evaluation. Section 6 provides a validation of our approach on two real-world case studies. Finally, Section 7 discusses related work and Section 8 concludes the paper. We provide the proofs of the theorems in appendix.

2 PRELIMINARIES

This section presents a general view of how Attribute-Based Access Control (ABAC) policies and queries are evaluated using the PTaCL language [4], which is an abstraction of the XACML standard [14]. We first present the syntax of PTaCL, which is based on two different languages: one for targets, which is used to decide the applicability of a policy to a query, and another for policies, which is used to specify how policies are combined together. We then present the different evaluation functions for PTaCL: the standard one, introduced in [4] and the extended one, introduced in [5].

2.1 ABAC Syntax

We consider here the ABAC paradigm, where queries are defined as sets of attribute values (instead of the traditional triple subject, object, access mode). More precisely, let $\mathcal{A} = \{a_1, \ldots, a_n\}$ be a finite set of attributes, and given an attribute $a \in \mathcal{A}$, let $\mathcal{V}_a$ be the domain of a. Given a set of attributes $\mathcal{A}$, we write $\mathcal{V}_\mathcal{A}$ for the union of all attribute domains, *i.e.* $\mathcal{V}_\mathcal{A} = \bigcup_{a\in\mathcal{A}} \mathcal{V}_a$. The set of queries $Q_\mathcal{A}$ is then defined as $\wp(\bigcup_{i=1}^n a_i \times \mathcal{V}_{a_i})$, and a *query* $q = \{(a_1, v_1), \ldots, (a_k, v_k)\}$ is a set of attribute name-value pairs (a_i, v_i) such that $a_i \in \mathcal{A}$ and $v_i \in \mathcal{V}_{a_i}$. A query encompasses both a specific request for access, and a current view of the world relative to the different entities concerned by that request.

The PTaCL language is *tree-based*, *i.e.* policies are either atomic or constructed by composing other policies. This vision follows the traditional "separation of concerns" principle: each policy might regulate accesses to a specific sub-domain of an organization, or regulate accesses done by a specific category of users or in specific contexts. In order to identify which policies are applicable to which targets, PTaCL introduce a target language $\mathcal{T}_\mathcal{A}$, such that a target $t \in \mathcal{T}_\mathcal{A}$ is defined as:

$$t = (a, v) \mid \mathrm{op}(t_1, \ldots, t_n)$$

where (a, v) is an attribute name-value pair, and op is an n-ary operator, defined over the set $\mathcal{D}_3 = \{1, 0, \perp\}$, indicating that the target matches the query, that the target does not match the query, and that it is indeterminate whether the target matches the query or not (the semantical evaluation of targets is described below). Crampton and Williams showed that the set of operators $\{\neg, E_1, \tilde{\sqcup}\}$ is canonically complete [6], *i.e.* any 3-valued operator can be constructed using these three operators. Table 1 presents these operators, as well as operators commonly used in ABAC languages.

PTaCL also defines a policy language $\mathcal{P}_\mathcal{A}$, where a policy $p \in \mathcal{P}_\mathcal{A}$ is defined as:

$$p = 1 \mid 0 \mid (t, p) \mid \mathrm{op}(p_1, \ldots, p_n)$$

where 1 and 0 represent the *allow* and *deny* decisions respectively, (t, p) is a target policy and op is a n-ary operator, also defined on the three-valued set $\{1, 0, \perp\}$, where $\perp$ represents the *not-applicable* decision. Although this set is syntactically equivalent to the one used for targets, the meaning of the values in the set depends on whether it is used as a target or as a policy. This should always be clear from the context in the remainder of this paper.

d_1	d_2	$\neg d_1$	$\sim d_1$	$E_1(d_1)$	$d_1 \tilde{\sqcap} d_2$	$d_1 \sqcap d_2$	$d_1 \vartriangle d_2$	$d_1 \tilde{\sqcup} d_2$	$d_1 \sqcup d_2$	$d_1 \triangledown d_2$
1	1	0	1	$\perp$	1	1	1	1	1	1
1	0	0	1	$\perp$	0	0	0	1	1	1
1	$\perp$	0	1	$\perp$	$\perp$	$\perp$	1	1	$\perp$	1
0	1	1	0	0	0	0	0	1	1	1
0	0	1	0	0	0	0	0	0	0	0
0	$\perp$	1	0	0	0	$\perp$	0	$\perp$	$\perp$	0
$\perp$	1	$\perp$	0	1	$\perp$	$\perp$	1	1	$\perp$	1
$\perp$	0	$\perp$	0	1	0	$\perp$	0	$\perp$	$\perp$	0
$\perp$	$\perp$	$\perp$	0	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Table 1: Operators on the set $\mathcal{D}_3 = \{1, 0, \perp\}$

2.2 ABAC Evaluation

Given the set of policies $\mathcal{P}_\mathcal{A}$, the set of queries $Q_\mathcal{A}$ and a set of decisions $\mathcal{D}$, an evaluation function is a function $[\![\cdot]\!] : \mathcal{P}_\mathcal{A} \times Q_\mathcal{A} \to \mathcal{D}$ such that, given a query q and a policy p, $[\![p]\!](q)$ represents the decision of evaluating p against q. PTaCL has three main policy evaluation functions, which handle missing attributes in a different way. For the sake of uniformity, hereafter we might use different notations than those in the original publications.

2.2.1 Standard Evaluation $[\![\cdot]\!]_\mathrm{P}$: The standard evaluation consists in evaluating a target to $\perp$ when the attribute is completely missing from the query, to 0 if the attribute is present in the query, but without the appropriate value, and to 1 otherwise. A policy then evaluates to a set of decisions within $\mathcal{D}_7 = \wp(\{1, 0, \perp\}) \setminus \emptyset$ where 1 and 0 indicate that access should be granted or denied respectively, and $\perp$ that the policy is not applicable to a given query. Non-singleton decisions are returned when the query does not provide the information necessary to evaluate a target (*i.e.*, the target evaluates to $\perp$). Intuitively, non-singleton decisions correspond to the Indeterminate decision in XACML [12].

More formally, the semantics of a target t is given by the function:

$$[\![\cdot]\!]_\mathrm{T} : \mathcal{T}_\mathcal{A} \times Q_\mathcal{A} \to \mathcal{D}_3$$

$$[\![(a, v)]\!]_\mathrm{T}(q) = \begin{cases} 1 & \text{if } (a, v) \in q \\ \perp & \text{if } \forall v' \in \mathcal{V}_a : (a, v') \notin q \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\mathrm{op}(t_1, \ldots, t_n)]\!]_\mathrm{T}(q) = \mathrm{op}([\![t_1]\!]_\mathrm{T}(q), \ldots, [\![t_n]\!]_\mathrm{T}(q))$$

The standard semantics of a policy p is given by the function:

$$[\![\cdot]\!]_\mathrm{P} : \mathcal{P}_\mathcal{A} \times Q_\mathcal{A} \to \mathcal{D}_7$$

$$[\![1]\!]_\mathrm{P}(q) = \{1\}$$

$$[\![0]\!]_\mathrm{P}(q) = \{0\}$$

$$[\![(t, p)]\!]_\mathrm{P}(q) = \begin{cases} [\![p]\!]_\mathrm{P}(q) & \text{if } [\![t]\!]_\mathrm{T}(q) = 1 \\ \{\perp\} & \text{if } [\![t]\!]_\mathrm{T}(q) = 0 \\ \{\perp\} \cup [\![p]\!]_\mathrm{P}(q) & \text{otherwise} \end{cases}$$

$$[\![\mathrm{op}(p_1, \ldots, p_n)]\!]_\mathrm{P}(q) = \mathrm{op}^\uparrow([\![p_1]\!]_\mathrm{P}(q), \ldots, [\![p_n]\!]_\mathrm{P}(q))$$

where, given an operator op : $\mathcal{D}_3 \times \mathcal{D}_3 \to \mathcal{D}_3$ and any non-empty sets $X, Y \subseteq \mathcal{D}_3$, $\mathrm{op}^\uparrow : \mathcal{D}_7 \times \mathcal{D}_7 \to \mathcal{D}_7$ is defined as

$\mathrm{op}^{\uparrow}(X, Y) = \{\mathrm{op}(x, y) \mid x \in X \wedge y \in Y\}$. Intuitively, $\mathrm{op}^{\uparrow}$ corresponds to operator op extended in a point-wise way to sets of decisions.

2.2.2 Simplified Evaluation $[\![\cdot]\!]_{\mathrm{B}}$*:* Crampton et al. [5] have shown that the standard evaluation, which is the one used by XACML, might yield a set of decisions that does not necessarily correspond to an intuitive interpretation of what those decisions mean due to missing attributes. In other words, given a policy, it is possible for a query to evaluate to a set of decisions D such that there exists a decision $d \in D$ for which the query extended with additional attribute values would not evaluate d, while there could be some decision $d' \notin D$ for which the query extended with some additional attribute values would evaluate to d'.

The authors have addressed the problem of missing attributes by proposing a novel evaluation mechanism based on non-deterministic attribute retrieval[1]. The fundamental intuition behind non-deterministic attribute retrieval is to model the fact that a query might represent a partial view of the world, whereby some attribute values are missing. In order to define the extended evaluation function, they first introduce the simplified evaluation function:

$$\begin{aligned}
[\![\cdot]\!]_{\mathrm{B}} &: \mathcal{P}_{\mathcal{A}} \times Q_{\mathcal{A}} \to \mathcal{D}_3 \\
[\![1]\!]_{\mathrm{B}}(q) &= 1 \\
[\![0]\!]_{\mathrm{B}}(q) &= 0 \\
[\![(t,p)]\!]_{\mathrm{B}}(q) &= \begin{cases} [\![p]\!]_{\mathrm{B}}(q) & \text{if } [\![t]\!]_{\mathrm{T}}(q) = 1 \\ \bot & \text{otherwise} \end{cases} \\
[\![\mathrm{op}(p_1, \ldots, p_n)]\!]_{\mathrm{B}}(q) &= \mathrm{op}([\![p_1]\!]_{\mathrm{B}}(q), \ldots, [\![p_n]\!]_{\mathrm{B}}(q))
\end{aligned}$$

The simplified evaluation function ignores missing attributes[2], and therefore always returns a single decision.

2.2.3 Extended Evaluation $[\![\cdot]\!]_{\mathrm{E}}$*:* The intuition behind the extended ABAC evaluation is that a query should evaluate to all possible decisions that can be obtained by adding possibly missing attributes. Hereafter, we represent a query space as a directed acyclic graph (save for self-loops) $(Q_{\mathcal{A}}, \to)$, where $Q_{\mathcal{A}}$ is a set of queries, and $\to \subseteq Q_{\mathcal{A}} \times Q_{\mathcal{A}}$ is a relation such that, given two queries $q, q' \in Q_{\mathcal{A}}$, $q \to q'$ if, and only if $q' = q \cup \{(a, v)\}$ for some attribute a and some value v.

However, it was identified that some extensions of q may not be possible. For instance, for a given Boolean attribute, it might not make sense to have in the same query both *true* and *false* for that attribute. Hence, Crampton et al. introduce in [5] the notion of negative attribute value to explicitly indicate that an attribute cannot have a certain value in a given context and the notion of well-formed predicate $\mathrm{wf} : Q_{\mathcal{A}} \to \mathbb{B}$ over queries to ensure that a query does not contain both an attribute value and its negation. Based on these notions, they define the extended evaluation function as:

$$\begin{aligned}
[\![\cdot]\!]_{\mathrm{E}} &: \mathcal{P}_{\mathcal{A}} \times Q_{\mathcal{A}} \to \mathcal{D}_8 = \wp(\{1, 0, \bot\}) \\
[\![p]\!]_{\mathrm{E}}(q) &= \{[\![p]\!]_{\mathrm{B}}(q') \mid q \to^* q' \wedge \mathrm{wf}(q')\}
\end{aligned}$$

where $\to^*$ denotes the reflexive-transitive closure of $\to$. With the restrictions imposed on $\to$, the relation $\to^*$ reduces to the subset relation on queries. It is worth observing that $[\![\cdot]\!]_{\mathrm{E}}$ returns the empty set for any query that does not satisfy wf.

3 PROBLEM STATEMENT

As discussed before, the standard ABAC evaluation function $[\![\cdot]\!]_{\mathrm{P}}$ can yield a decision that does not necessarily provide an intuitive interpretation on whether access should be granted or not due to the fact that some information needed for the evaluation might be missing. To overcome the drawbacks of function $[\![\cdot]\!]_{\mathrm{P}}$, given a policy p and a query q, a policy enforcement point (*i.e.*, the point in the system in charge of enabling an access query or not) can decide to evaluate $[\![p]\!]_{\mathrm{E}}(q)$ in addition to $[\![p]\!]_{\mathrm{B}}(q)$, to see whether any missing attribute could change the evaluation. However, the extended evaluation $[\![\cdot]\!]_{\mathrm{E}}$ can lead to a very large query space to be explored, making policy evaluation inefficient. We exemplify these issues in the following example.

Consider a system wherein access is based on the nationality of users. In particular, the system allows Belgians to access system resources, whereas the Dutch are not. This policy can be represented as follows:

$$p = ((\mathbf{nat}, \mathsf{BE}), 1) \vartriangle ((\mathbf{nat}, \mathsf{NL}), 0)$$

Consider now a user submitting the query $q = \{(\mathbf{nat}, \mathsf{BE})\}$: this query evaluates to $[\![p]\!]_{\mathrm{P}}(\{(\mathbf{nat}, \mathsf{BE})\}) = \{1\}$, *i.e.* the access would be granted. However, it is possible for a user to have multiple nationalities, and in some cases, it might be possible for a user to hide some nationalities[3]. In our case, the user might be hiding that she also has a Dutch nationality, in which case the access should have been denied since $[\![p]\!]_{\mathrm{P}}(\{(\mathbf{nat}, \mathsf{BE}), (\mathbf{nat}, \mathsf{NL})\}) = \{0\}$.

To address this issue, we can use the extended evaluation function $[\![\cdot]\!]_{\mathrm{E}}$. In particular, we obtain $[\![p]\!]_{\mathrm{E}}(\{(\mathbf{nat}, \mathsf{BE})\}) = \{1, 0\}$ indicating that there exists a query (*i.e.*, a view of the world) reachable from q that should be denied. Computing $[\![p]\!]_{\mathrm{E}}(q)$, however, requires evaluating all the queries that can be constructed from the initial query q. There are 206 sovereign states recognized by the United Nations,[4] and users can have more than one nationality. This leads to two main problems:

- A naïve implementation of $[\![\cdot]\!]_{\mathrm{E}}$ requires evaluating 2^{205} queries (*i.e.*, the queries that can be constructed from the initial query $\{(\mathbf{nat}, \mathsf{BE})\}$), which is clearly infeasible.
- Some combination of nationalities are not possible, due to specific country restrictions on dual nationality, which can lead to misleading decisions.

Concerning the first problem, it is worth observing that, although there is no limit on the number of nationalities individuals can hold according to international laws, this number is usually limited. For the sake of exemplification, we assume here that individuals cannot hold more than three nationalities. Accordingly, the number of queries to be constructed from the initial query $\{(\mathbf{nat}, \mathsf{BE})\}$ and evaluated is $21,116$.[5] Although this is still a large number, it is

[1]They also consider probabilistic attribute retrieval, which is beyond the scope of this paper.
[2]In XACML, this would correspond to no attribute indicated as *must-be-present.*

[3]For instance illustrated in 2017 with the Australian parliament, where seven members of parliament were revealed to hold dual nationalities and therefore were not eligible.
[4]https://en.wikipedia.org/wiki/List_of_sovereign_states
[5]The number of queries to be evaluated is the number of combinations of at most three nationalities where one nationality is Belgian, *i.e.* $\sum_{k=0}^{2} \binom{205}{k}$.

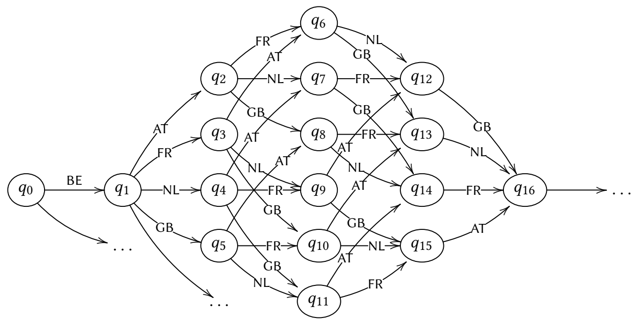

Figure 1: Portion of the query space

negligible with respect to the number of requests that have to be constructed without considering this domain constraint.

To visually represent the problem, consider (the portion of) the query space in Fig. 1, which is explored in the evaluation of p. In the figure, nodes represent queries (with q_0 the empty query) and edges are annotated with a label indicating how a query has been extended, *i.e.* $q_i \xrightarrow{\mathsf{NL}} q_j$ denotes $q_j = q_i \cup \{(\mathbf{nat}, \mathsf{NL})\}$. If we consider that no user has more than three nationalities, then all queries formed by four or more attribute name-value pairs are not reachable from the initial state (*i.e.*, queries q_{12} to q_{16} in Fig. 1) as they are not plausible views of the world.

More importantly, ignoring domain constraints can result in decisions that cannot be reached in practice, thus providing misleading information for decision making. Suppose for instance that Austria does not allow dual nationality with the Netherlands[6]. In this case, we should exclude from the state space any query containing $\{(\mathbf{nat}, \mathsf{NL}), (\mathbf{nat}, \mathsf{AT})\}$, in which case, given the query $\{(\mathbf{nat}, \mathsf{AT})\}$, we have $[\![p]\!]_{\mathrm{E}}(\{(\mathbf{nat}, \mathsf{AT})\}) = \{1, \bot\}$. In other words, it is impossible for this request to be denied, even if some attribute is missing. Note that, according to the standard evaluation, $[\![p]\!]_{\mathrm{P}}(\{(\mathbf{nat}, \mathsf{AT})\}) = \{\bot\}$ regardless the existence of the domain constraint.

In the remainder of the paper, we will exploit these observations to establish the foundations for the design of practical policy engines supporting the extended evaluation of attribute-based access control policies. In particular:

- We introduce the notion of *query constraints* to identify which views of the world are plausible based on domain specific requirements and assumptions, thus constructing a realistic query space (Section 4).
- We investigate practical approaches for the computation and representation of the extended evaluation $[\![\cdot]\!]_{\mathrm{E}}$ of a given policy (Section 5).

4 QUERY CONSTRAINTS

As shown above, a non-deterministic evaluation of ABAC policies requires the construction of all possible views of the world from a given query, leading to a huge number of queries to be evaluated. However, many of these views may not be possible in practice. In fact, a system can be characterized by domain requirements and assumptions that determine which views of the world are plausible and which are not. The main problem lies in the fact that domain requirements and assumptions are typically defined outside the authorization mechanism and, thus, not available for policy evaluation.

It is worth emphasizing here that there is a fundamental distinction between queries that are not possible and queries that should be denied. In the previous section, a query including both Austrian and Dutch nationalities is neither denied or granted, but considered instead as not possible.

To address this issue, we introduce the notion of *query constraint*. Intuitively, query constraints encode domain requirements and are used to reduce the portion of the query space to be explored (*i.e.*, the number of queries that are reachable from the initial query).

Syntactically speaking, the language for constraints $\mathcal{C}_{\mathcal{A}}$ is such that a constraint $c \in \mathcal{C}_{\mathcal{A}}$ is defined as:

$$c = (a, v) \mid \mathrm{op}(c_1, \ldots, c_n)$$

where a is attribute, v in an attribute value, and op is a Boolean operator. The only difference between $\mathcal{C}_{\mathcal{A}}$ and $\mathcal{T}_{\mathcal{A}}$ (defined in Section 2.1) is that we do not consider three-valued operators for constraints. We therefore have $\mathcal{C}_{\mathcal{A}} \subseteq \mathcal{T}_{\mathcal{A}}$, since any Boolean operator trivially corresponds to a three-valued operator. The evaluation of a constraint is given by the function:

$$[\![\cdot]\!]_{\mathrm{C}} : \mathcal{C}_{\mathcal{A}} \times \mathcal{Q}_{\mathcal{A}} \to \mathbb{B}$$

$$[\![(a, v)]\!]_{\mathrm{C}}(q) = \begin{cases} 1 & \text{if } (a, v) \in q \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\mathrm{op}(t_1, \ldots, t_n)]\!]_{\mathrm{C}}(q) = \mathrm{op}([\![t_1]\!]_{\mathrm{C}}(q), \ldots, [\![t_n]\!]_{\mathrm{C}}(q))$$

Example 1. Some countries such as Singapore, Austria and India, do not allow dual nationality, leading to automatic loss of citizenship upon acquiring another nationality. Other countries restrict dual nationality to certain countries. For instance, Pakistan allows double nationality only with 16 countries and Spain allows only with certain Latin American countries, Andorra, Portugal, the Philippines and Equatorial Guinea. These requirements can be modeled using query constraints. For instance, the following constraint indicates that it is not possible to have both Austrian and Dutch citizenships: $\neg((\mathbf{nat}, \mathsf{AT}) \land (\mathbf{nat}, \mathsf{NL}))$.

Some constraints might be more complex to build. For instance, we might want to have cardinality constraints specifying the maximum number of values a particular attribute can take. However, there is no Boolean operator expressing directly such constraints. Instead, given an attribute a and a number k, we can generate the corresponding constraint enumerating all possible cases. We first write $\mathcal{A}_{|a} = \{(a, v) \mid v \in \mathcal{V}_a\}$ for the set of all attribute values for a. We then write $C_{a,k} = \{s \subseteq \mathcal{A}_{|a} \mid |s| = k + 1\}$ for the set of subsets of $\mathcal{A}_{|a}$ with a cardinality equal to $k + 1$. Finally, the *cardinality constraint* expressing that an attribute can have at most k values can be expressed as:

$$card_{a,k} = \bigwedge_{s \in C_{a,k}} \neg \bigwedge_{(a,v) \in s} (a, v)$$

Any query containing more than k values for attribute a would have at least one set $s \in C_{a,k}$ for which the conjunction of the attribute values would be true, rendering the whole conjunction $card_{a,k}$ false.

[6]Actual rules for dual-nationality tend to be very complex, and we do not go into any detail here.

Example 2. Consider the scenario in Section 3, such that, for the sake of exposition, we only consider six possible nationalities: FR, AT, GB, DE, BE and NL. The constraint that individuals cannot hold more than three nationalities can be expressed by the constraint $card_{\mathbf{nat},3}$, which consists of 30 conjunctions of conjunctions:

$$\begin{aligned} card_{\mathbf{nat},3} = \; & \neg((\mathbf{nat}, \mathsf{FR}) \wedge (\mathbf{nat}, \mathsf{AT}) \wedge (\mathbf{nat}, \mathsf{GB}) \wedge (\mathbf{nat}, \mathsf{DE})) \\ & \wedge \neg((\mathbf{nat}, \mathsf{FR}) \wedge (\mathbf{nat}, \mathsf{AT}) \wedge (\mathbf{nat}, \mathsf{GB}) \wedge (\mathbf{nat}, \mathsf{BE})) \\ & \wedge \ldots \\ & \wedge \neg((\mathbf{nat}, \mathsf{GB}) \wedge (\mathbf{nat}, \mathsf{DE}) \wedge (\mathbf{nat}, \mathsf{BE}) \wedge (\mathbf{nat}, \mathsf{NL})) \end{aligned}$$

As demonstrated by the example above, the cardinality constraint for an attribute a can only be constructed in this form when the attribute domain $\mathcal{V}_a$ is finite. In this paper, our encoding of ABAC policies requires anyway finite domains for attributes, and we leave the investigation of infinite attribute domains for future work. Hereafter, given a set of constraints C, we write $Q_{\mathcal{A}|C}$ for the set $\{q \in Q_{\mathcal{A}} \mid \forall c \in C \; [\![c]\!]_{\mathrm{C}}(q) = 1\}$.

5 EFFICIENT EXTENDED EVALUATION COMPUTATION

In the previous section, we introduced the notion of query constraint. The use of query constraints has the advantage to remove queries that are not possible. We next propose an algorithm for computing the extended evaluation function $[\![\cdot]\!]_{\mathrm{E}}$. Along the lines of previous work (*e.g.*, [1, 8, 9]), our algorithm relies on the use of *binary decision diagram* (BDD) for the representation of ABAC policies, query constraints and the query space. We evaluate this approach in the following section.

We first briefly review the essential concepts behind BDDs and how we use them to represent the aforementioned artifacts. For a more in-depth treatment of the underlying algorithmics for constructing and manipulating BDDs, we refer to [2].

Let *Vars* be a finite set of Boolean variables. A propositional formula over *Vars* can be efficiently represented by a BDD. Formally, a BDD is a graph-based data structure defined as follows:

Definition 1. A binary decision diagram (BDD) is a rooted directed acyclic graph with vertex set V containing the terminal vertices 0 and 1, and non-terminal vertices that are labelled (using a function L) with variables from *Vars*. Non-terminal vertices have exactly one outgoing high edge (denoted *hi*) and one outgoing low edge (denoted *lo*). Terminal vertices have no outgoing edges.

A BDD is said to be *reduced* if it contains no vertex v with $lo(v) = hi(v)$, nor does it contain two distinct vertices v and v' whose subgraphs (*i.e.*, the BDDs rooted in v and v') are isomorphic. In this work, we are only concerned with reduced BDDs.

We assume a fixed ordering $<$ on the Boolean variables *Vars*. A propositional formula can be represented uniquely (up-to-isomorphism) by a (reduced) BDD by labeling each non-terminal vertex v with a Boolean variable $L(v)$, ensuring that each successor vertex v' of v is either a terminal vertex or a vertex labeled with a Boolean variable $L(v') < L(v)$. The formula $F(v)$, represented by a BDD with root v, is obtained as follows:

$$F(v) = \begin{cases} \mathit{false} & \text{if } v = 0 \\ \mathit{true} & \text{if } v = 1 \\ (L(v) \implies F(hi(v))) \wedge (\neg L(v) \implies F(lo(v))) & \text{otherwise} \end{cases}$$

Checking whether a concrete truth-assignment to the Boolean variables is such that the propositional formula represented by the BDD holds reduces to checking whether in the BDD, the path associated with the variable assignment leads to terminal vertex 1. That is, the runtime complexity for evaluating whether such a truth-assignment makes a formula true is linear in the depth of the BDD, which, in turn, is limited by the size of *Vars*. BDDs can be used effectively for representing and computing the extended evaluation; we explain how this is done in the remainder of this section.

Henceforward, let $(Q_{\mathcal{A}|C}, \rightarrow)$ be a fixed constrained query space ranging over a set of attribute names $\mathcal{A}$ and attribute domains $\mathcal{V}_{\mathcal{A}}$. We represent each attribute name-value pair $(a, v) \in \mathcal{A} \times \mathcal{V}_{\mathcal{A}}$ by a Boolean variable a_v. The set of all Boolean variables is denoted $Vars_{\mathcal{A}}$. A truth-assignment to all Boolean variables represents a single query. A set of queries can be represented as a propositional formula over these variables. For instance, the propositional formula $\neg(\mathbf{nat}_{\mathsf{AT}} \wedge \mathbf{nat}_{\mathsf{NL}})$ encodes the set of all queries except those queries that contain both attribute name-value pairs $(\mathbf{nat}, \mathsf{AT})$ and $(\mathbf{nat}, \mathsf{NL})$. A query q induces an *interpretation* $I(q)$ which is defined as $I(q)(a_v) = \mathit{true}$ iff $(a, v) \in q$. Given an interpretation $I(q)$ and a propositional formula ϕ, we write $I(q) \models \phi$ iff the formula evaluates to *true* under interpretation $I(q)$.

The main idea behind our approach is as follows. Given a fixed policy p, we construct a triple $(e_1, e_0, e_\perp)$ of propositional formulae representing sets of queries Q_1, Q_0 and $Q_\perp$ such that $d \in [\![p]\!]_{\mathrm{E}}(q)$ exactly when $q \in Q_d$. We represent these propositional formulae using (reduced) BDDs. For computing the triple of propositional formulae $(e_1, e_0, e_\perp)$, we use the following formulae:

(1) a triple of propositional formulae $(b_1, b_0, b_\perp)$ representing $[\![p]\!]_{\mathrm{B}}$,
(2) a propositional formula S encoding the constrained query space $Q_{\mathcal{A}|C}$,
(3) a propositional formula R encoding relation $\rightarrow^*$ on $Q_{\mathcal{A}|C}$.

The triple of propositional formulae $(b_1, b_0, b_\perp)$ representing $[\![p]\!]_{\mathrm{B}}$ is computed recursively using transformations τ and π employing the inductive definition of the policy language. More specifically, each b_d is a propositional formula representing a set of queries $Q_d \subseteq Q_{\mathcal{A}}$ satisfying $d = [\![p]\!]_{\mathrm{B}}(q)$ whenever $q \in Q_d$. Hereafter, we write $\tau_d(t)$ and $\pi_i(p)$ for the formulae representing decision d in the transformation $\tau(t)$ and $\pi(p)$, respectively. The transformation rules for τ (for targets) and π (for policies) given in Table 2 explain the construction of the propositional formula for decision 1 for all targets, (policy) constants and all (policy and target) operators of Table 1. The rules for constructing the propositional formulae for decisions 0 and $\perp$, given by $\tau_0(t)$, $\pi_0(p)$, $\tau_\perp(t)$ and $\pi_\perp(p)$, are similar (see Appendix). The correctness of the propositional formulae $\tau_d(t)$ and $\pi_d(p)$ is stated by the following lemma.

Lemma 1. *For all* $q \in Q_{\mathcal{A}}$*:*
(a) $I(q) \models \tau_d(t)$ *iff* $d = [\![t]\!]_{\mathrm{T}}(q)$,
(b) $I(q) \models \pi_d(p)$ *iff* $d = [\![p]\!]_{\mathrm{B}}(q)$.

Example 3. Let us reconsider policy p introduced in Section 3. By applying transformations τ_1 and π_1 in Table 2 to p, we obtain (after minor simplification) the following propositional formula:

$$\pi_1(p) = \mathbf{nat}_{\mathsf{BE}} \wedge \neg\mathbf{nat}_{\mathsf{NL}}$$

$\tau_1((a, v))$	$=$	a_v
$\tau_1(\neg t_1)$	$=$	$\tau_0(t_1)$
$\tau_1(\sim t_1)$	$=$	$\tau_1(t_1)$
$\tau_1(E_1(t_1))$	$=$	$\tau_\perp(t_1)$
$\tau_1(t_1 \tilde{\sqcap} t_2)$	$=$	$\tau_1(t_1) \wedge \tau_1(t_2)$
$\tau_1(t_1 \sqcap t_2)$	$=$	$\tau_1(t_1) \wedge \tau_1(t_2)$
$\tau_1(t_1 \vartriangle t_2)$	$=$	$(\tau_1(t_1) \wedge \neg\tau_0(t_2)) \vee (\tau_1(t_2) \wedge \neg\tau_0(t_1))$
$\tau_1(t_1 \tilde{\sqcup} t_2)$	$=$	$\tau_1(t_1) \vee \tau_1(t_2)$
$\tau_1(t_1 \sqcup t_2)$	$=$	$(\tau_1(t_1) \wedge \neg\tau_\perp(t_2)) \vee (\tau_1(t_2) \wedge \neg\tau_\perp(t_1))$
$\tau_1(t_1 \triangledown t_2)$	$=$	$\tau_1(t_1) \vee \tau_1(t_2)$
$\pi_1(1)$	$=$	*true*
$\pi_1(0)$	$=$	*false*
$\pi_1((t, p_1))$	$=$	$\tau_1(t) \wedge \pi_1(p_1)$
$\pi_1(\neg p_1)$	$=$	$\pi_0(p_1)$
$\pi_1(\sim p_1)$	$=$	$\pi_1(p_1)$
$\pi_1(E_1(p_1))$	$=$	$\pi_\perp(p_1)$
$\pi_1(p_1 \tilde{\sqcap} p_2)$	$=$	$\pi_1(p_1) \wedge \pi_1(p_2)$
$\pi_1(p_1 \sqcap p_2)$	$=$	$\pi_1(p_1) \wedge \pi_1(p_2)$
$\pi_1(p_1 \vartriangle p_2)$	$=$	$(\pi_1(p_1) \wedge \neg\pi_0(p_2)) \vee (\pi_1(p_2) \wedge \neg\pi_0(p_1))$
$\pi_1(p_1 \tilde{\sqcup} p_2)$	$=$	$\pi_1(p_1) \vee \pi_1(p_2)$
$\pi_1(p_1 \sqcup p_2)$	$=$	$(\pi_1(p_1) \wedge \neg\pi_\perp(p_2)) \vee (\pi_1(p_2) \wedge \neg\pi_\perp(p_1))$
$\pi_1(p_1 \triangledown p_2)$	$=$	$\pi_1(p_1) \vee \pi_1(p_2)$

Table 2: Transformation rules for τ (for targets) and π (for policies) for the case 1; the rules for the cases 0 and $\perp$ are similar and can be deduced from the semantics of $[\![\_]\!]_T$ and $[\![\_]\!]_B$.

Fig. 2a shows the corresponding BDD. The BDDs corresponding to $\pi_0(p)$ (Fig. 2b) and $\pi_\perp(p)$ (Fig. 2c) can be obtained in a similar way. In the figures, solid arrows indicate that the path of the BDD in case a given attribute value is present in the query (*i.e.* the *hi*-edge), whereas dashed arrows indicate that the attribute value is not present (*i.e.* the *lo*-edge); terminal nodes are represented using a double line rectangle. These BDDs show that any query including attribute name-value pair (**nat**, NL) evaluates to 0 and any query including attribute name-value pair (**nat**, BE) (and not (**nat**, NL)) evaluates to 1; if both (**nat**, BE) and (**nat**, NL) are not present, the query evaluates to $\perp$.

The propositional formula S representing the constrained query space $Q_{\mathcal{A}|C}$ can be readily constructed by reusing transformation τ, since the constraint language $C_{\mathcal{A}}$ is essentially a subset of the target language $\mathcal{T}_{\mathcal{A}}$. The constrained query space can therefore be represented by the following propositional formula for S:

$$\bigwedge\{\tau_1(c) \mid c \in C\}$$

Example 4. Fig. 2d shows a BDD encoding a constrained query space. It is obtained by applying transformation τ to the cardinality constraint in Example 2 (*i.e.*, $card_{\mathbf{nat},3}$) in conjunction with a query constraint imposing that individuals having an Austrian nationality cannot have dual nationality. It is easy to observe in the BDD that queries including attribute name-value pair (**nat**, AT) and any other nationalities are invalid (left part of Fig. 2d); queries that contain four nationalities are invalid as well and thus all map to terminal vertex 0.

For representing the relation $\rightarrow^*$, we introduce a set of copies of all Boolean variables; that is, for each variable a_v, we introduce a unique copy a'_v representing the value of a_v in a reachable query. We denote the set of variables consisting of these copies by $Vars'_{\mathcal{A}}$. Since $\rightarrow^*$ is in essence the subset relation, the proposition R encoding this relation is constructed by conjunctively composing the propositional formulae $a_v \implies a'_v$. The correctness of this encoding is given by the following lemma.

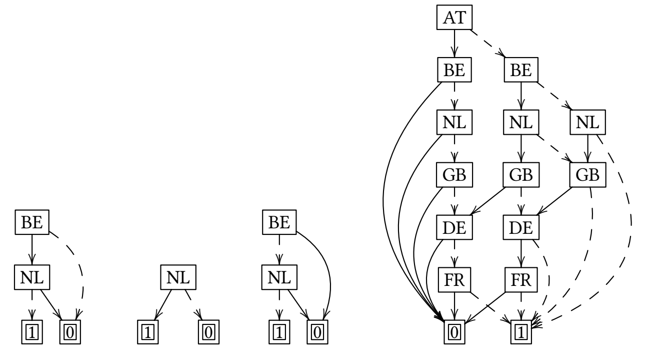

Figure 2: BDDs $(b_1, b_0, b_\perp)$ and constrained query space for the policy in Section 3

LEMMA 2. *Let $q, q' \in Q_{\mathcal{A}}$. Let $I(q)$ denote the interpretation for Vars and $I'(q')$ the interpretation for Vars′, defined as $I'(q')(a'_v) =$ true iff $(a, v) \in q'$. We then have $I(q) \cup I'(q') \models \bigwedge\{a_v \implies a'_v \mid a_v \in Vars\}$ iff $(q, q') \in \rightarrow^*$.*

Note that we also need to ensure that only queries from the set represented by S are considered. We achieve this by strengthening the transition relation using the propositional formula S. This leads to the following propositional formula for the transition relation R:

$$S \wedge S[Vars_{\mathcal{A}} := Vars'_{\mathcal{A}}] \wedge \bigwedge\{a_v \implies a'_v \mid a_v \in Vars_{\mathcal{A}}\}$$

The substitution notation we use in this formula is short-hand for replacing each unprimed variable by its primed counterpart in the propositional formula.

Using the triple $(b_1, b_0, b_\perp)$, the constrained query space encoded by S and the transition relation encoded by R, we can compute a triple of propositional formulae $(e_1, e_0, e_\perp)$ representing $[\![p]\!]_E$ using a backwards reachability analysis. Since our transition relation R is transitively closed, it essentially suffices to use R to compute all immediate predecessors of b_1, b_0 and $b_\perp$. The computation of predecessors can be performed effectively on the level of BDDs using a standard encoding of the existential quantification over all primed variables. For e_1, this boils down to computing the (reduced) BDD for the following formula:

$$(S \wedge b_1) \vee \exists Vars'_{\mathcal{A}}.(R \wedge (b_1[Vars_{\mathcal{A}} := Vars'_{\mathcal{A}}]))$$

The computation of e_0 and $e_\perp$ proceeds analogously. We summarize the steps we take to compute the extended evaluation in Algorithm 1. The correctness of the algorithm is stated in the following theorem.

THEOREM 3. *Procedure COMPUTEEXTENDEDEVALUATION computes, for a given policy p and a constrained query space $(Q_{\mathcal{A}|C}, \rightarrow)$, a triple $(e_1, e_0, e_\perp)$ of BDDs representing sets $(Q_1, Q_0, Q_\perp)$ satisfying, for each $q \in Q_{\mathcal{A}}$, $q \in Q_d$ iff $q \in Q_{\mathcal{A}|C} \wedge d \in [\![p]\!]_E(q)$.*

As we explained above, testing whether a truth-assignment to all variables makes a propositional formula true can be done in worst-case time $O(|Vars|)$. As a consequence, the BDDs $(e_1, e_0, e_\perp)$ that are computed by Algorithm 1 can be used to simply and efficiently evaluate a policy p for a concrete query q using the extended evaluation $[\![\cdot]\!]_E$: for each $d \in \{1, 0, \perp\}$, one evaluates at run-time

Algorithm 1 Pseudo code for computing the extended evaluation for a policy p and constrained query space $(Q_{\mathcal{A}|C}, \rightarrow)$.

```
1: procedure ComputeExtendedEvaluation
2:     $(b_1, b_0, b_\perp) := (\pi_1(p), \pi_0(p), \pi_\perp(p))$
3:     $S := \bigwedge\{\tau_1(c) \mid c \in C\}$
4:     $R := S \wedge S[Vars_{\mathcal{A}} := Vars'_{\mathcal{A}}] \wedge \bigwedge\{a_v \implies a'_v \mid a_v \in Vars_{\mathcal{A}}\}$
5:     for $d \in \{1, 0, \perp\}$ do
6:         $e_d := (S \wedge b_d) \vee \exists Vars'_{\mathcal{A}}.(R \wedge (b_d[Vars_{\mathcal{A}} := Vars'_{\mathcal{A}}]))$
7:     end for
8:     return $(e_1, e_0, e_\perp)$
9: end procedure
```

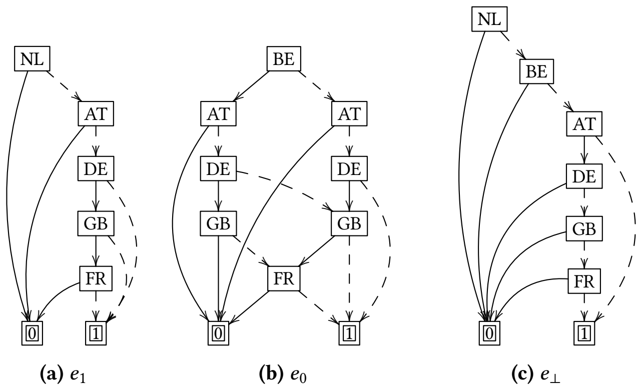

Figure 3: BDDs $(e_1, e_0, e_\perp)$ for the policy in Section 3

whether $d \in [\![p]\!]_E(q)$ by inspecting BDD e_d, in worst-case time $O(|Vars|)$.

Example 5. Fig. 3 illustrates the BDDs $(e_1, e_0, e_\perp)$ encoding the extended evaluation of the policy in Example 4 augmented with the constrained query space in Fig. 2d. The BDD in Fig. 3a shows that a query will never be evaluated to 1 if it contains attribute name-value pairs (**nat**, NL) and (**nat**, AT). In fact, any query containing (**nat**, NL) is always evaluated to 0 as shown in Fig. 2b and a query containing (**nat**, AT) cannot be extended as imposed by query constraints. The other paths in the BDD indicate that a query not including those attribute name-value pairs can potentially be evaluated to 1 as the query can be extended with attribute name-value pair (**nat**, BE) unless the query already includes three nationalities (the maximum number of nationalities allowed in our scenario). Similar observations holds for the other BDDs in Fig. 3.

6 CASE STUDIES

In this section, we demonstrate our approach to compute BDDs encoding the extended evaluation of ABAC policies using two real-world policies, namely the Continue policy and the SAFAX policy. The approach has been implemented in Python using the dd library[7] (v. 0.5.2). In our experiments, we also compared the standard evaluation function $[\![\cdot]\!]_P$ and the extended evaluation function $[\![\cdot]\!]_E$. To this end, we have also implemented the construction of BDDs encoding function $[\![\cdot]\!]_P$ using the dd library by extending the transformation rules τ and π presented in Section 5. The experiments were performed using a machine with 2.30GHz Intel Xeon processor and 16 GB of RAM.

[7]https://github.com/johnyf/dd

	Policy Size			#Var	#Value Constraints	#Cardinality Constraints
	#PS	#P	#R			
Continue	111	266	298	47	10	2
SAFAX (10)	5	18	35	54	36	5
SAFAX (20)	5	18	35	84	36	5
SAFAX (50)	5	18	35	174	36	5

Table 3: Overview of the datasets used for the experiments

6.1 Datasets

This section provides an overview of the two real-world policies used for our demonstration. Both policies are specified in XACML v2 [13]. XACML has several commonalities with PTaCL; in particular, it has been shown in previous work [12] that XACML policies can be encoded in PTaCL. For the sake of space, we refer to [12] for the details of the encoding. A summary of the policies and datasets constructed from them is given in Table 3. In the table, we report the size of the policies (in terms of number of policysets (**#PS**), policies (**#P**) and rules (**#R**)), the number of variables used to encode the policy (**#Var**) and the number of cardinality and value constraints (*i.e.*, query constraints excluding that two different attribute values can be present at the same time).

Continue*:* Continue is a conference manager system that supports the submission, review, discussion and notification phases of conferences. The Continue policy[8] consists of 111 policysets that, in turn, consist of 266 policies comprising 298 rules. The target of policysets, policies and rules are defined over 14 attributes ranging from the role of users (role) within the conference management system, the type of resource accessed (resource_class) and the action for which access is requested (action_type) to attributes used to characterize the existence of conflicts of interest (isConflicted) and the status of the review process (isReviewContentInPlace, isPending, etc.). Some of these attributes are Boolean, whereas others, such as role and resource_class, take values from a more complex domain. In total, the union of the attribute domains for the Continue policy consists of 47 attribute values.

Together with the policy we specified 10 value constraints. In particular, 9 constraints were used to enforce that Boolean attributes can be either *true* or *false*. The other value constraint was used to impose that subreviewers cannot be PC members as required by Continue [8]. Moreover, we defined two cardinality constraints to restrict the values that attributes resource_class and action_type can take as suggested in [8].

SAFAX: SAFAX [10] is an XACML-based framework that offers authorization as a service. SAFAX provides a web interface through which users can create, manage and configure their authorization services. The SAFAX policy is used to regulate the action users can perform on the web interface. The SAFAX policy consists of 5 policysets, 18 policies and 35 rules. The target of these policy elements are built over 8 attributes ranging from the group(s) a user belongs to (group), the type of object to be accessed (type) and

[8]http://www.margrave-tool.org/v1+v2/margrave/versions/01-01/examples/continue/

the action to be performed on the object (action) to the number of objects a user has already created (count-project, count-demo, count-ppdp) and the relation of the user with the object (isowner, match_project). The last two attributes are Boolean, whereas the others have a more complex domain. In particular, three attributes range over integer numbers. To test the scalability of our approach, we varied the size of the domain of these attributes. In particular, we generated three datasets – SAFAX (10), SAFAX (20) and SAFAX (50) – where the number in parentheses represents the size of the domain of numerical attributes.

We also defined a number of query constraints that reflect the functioning of the system. Besides introducing constraints for Boolean attributes and cardinality constraints for numerical attributes, we restricted the number of object types and actions that can occur in a request. This is motivated by the fact that, in SAFAX, an object can have only one type and access requests are triggered to determine whether a user is allowed to perform a certain action. Moreover, certain actions can be performed only on certain types of objects. We modeled these domain requirements using value constraints. We also defined constraints to restrict the groups a user can belong to simultaneously. Users should register to SAFAX to use the web application. Nonetheless, SAFAX also provides a guest account (with limited functionalities) that allows the use of the application without registration. Guest users are assigned to a special group that is incompatible with other groups. We capture this requirement using value constraints. In total, we complemented the policy with 36 value constraints and 5 cardinality constraints.

6.2 Results

This section presents the results of our experiments. First, we analyze the BDDs obtained using the extended evaluation $[\![\cdot]\!]_E$ and its feasibility in real scenarios. Then, we compare $[\![\cdot]\!]_E$ with the standard evaluation $[\![\cdot]\!]_P$. Finally, we present lessons learned from our experiments and discuss the limitations of the approach.

Analysis of extended evaluation function $[\![\cdot]\!]_E$. For each dataset, Table 4 shows the size of the BDDs obtained using the simplified evaluation function $[\![\cdot]\!]_B$ presented in Section 2.2, and the size of the BDDs obtained using the extended evaluation function $[\![\cdot]\!]_E$ with and without constraints. In particular, for each BDD, the table reports the number of vertices and the depth of the BDD. The depth of BDDs is particularly important as it affects policy evaluation. Table 5 reports the size of the BDDs encoding the constrained query space for the datasets, which represent the set of valid queries (*i.e.*, the queries that satisfy the constraints) along with the number of valid queries. This latter information provides an indication of the size of the constrained query space. Moreover, the table reports the percentage of queries that evaluate to 1, 0 and $\perp$ for $[\![\cdot]\!]_B$ and $[\![\cdot]\!]_E$. One can observe that, for $[\![\cdot]\!]_E$, the sum of percentages is greater than 100%. Recall from Section 2 that $[\![\cdot]\!]_E$ is defined over $\mathcal{D}_8 = \wp(\{1, 0, \perp\})$.

The reported statistics were obtained after applying the garbage collection and reordering functions provided by the dd library. The garbage collector function deletes unreferenced nodes. Reordering is used to change the variable order to reduce the size of the BDD representation. In particular, it uses Rudell's sifting algorithm [15], a widely used heuristics for dynamic reordering, to search for a better (fixed) order of variables compared the one currently used. Moreover, the reordering function is nondeterministic in the sense that it can return different orders of variables for the same input set of BDDs. This explains the differences in the number of nodes between the BDDs encoding the simplified evaluation of the SAFAX policy (top-left block of Table 4).[9]

In Table 4 (top-right block), we can observe that the BDDs encoding the extended evaluation without constraints for decision 1 consist of only one vertex. This vertex is the terminal vertex *true*, indicating that all queries can be potentially evaluated to 1. This is due to how the Continue and SAFAX policies are defined. For instance, in the Continue policy positive authorizations have a higher priority than negative authorizations, *i.e.* all XACML policy elements are combined using the first-applicable combining algorithm and *Permit* rules always occur at the top, thus yielding *Permit* whenever they are applicable. Similarly, the SAFAX policy employs a number of *Deny* rules but they are only used as default rules. Thus, if all attribute values are provided in the query, both policies evaluate 1. This demonstrates the importance of constraints. By looking at Table 5, we can observe that only 59% of queries could actually yield decision 1 for the Continue policy and 97% for the SAFAX policy. Thus, neglecting constraints can result in misleading decisions.

We can also observe from Table 4 that the BDDs encoding the simplified evaluation and the extended evaluation without constraints (top-left and top-right blocks, resp.) for $\perp$ are the same. This is expected as the applicability of both the Continue and SAFAX policies is monotonic; if they apply to a query, they also apply to all queries that can be constructed from it. Thus, it is not possible that a query evaluates to $\perp$ according to $[\![\cdot]\!]_E$ but not according to $[\![\cdot]\!]_B$. We can also observe that, for the SAFAX policy, these BDDs are relative small (7 nodes and depth equal to 6) and, from Table 5, that they cover about 16% of the query space. This is due to the use of default *Deny* rules mentioned above. As a matter of fact, these rules map most of the queries for which a positive authorization is not specified to 0.

As discussed in Section 5, the depth of a BDD is upper bounded by the number of variables. We can observe in Table 4 (bottom-left and bottom-right blocks) that, for the SAFAX policy with constraints, the depth of BDDs is exactly equal to the number of variables. This is due to the fact that the constraints defined for this policy involve all attribute values. This is also visible by observing in Table 3 that the depth of the BDDs representing the constrained query space is equal to the number of variables, indicating that all variables are used to determine the validity of queries.

Feasibility of extended evaluation function $[\![\cdot]\!]_E$. To assess the feasibility of the approach, we considered the time needed to generate the BDDs encoding the extended evaluation of the Continue and SAFAX policies along with the corresponding query constraints and the memory required to store the generated BDDs. Table 6 reports the time required to generate the BDDs encoding the extended evaluation on the constrained query space. From the table, we can observe that the construction of BDDs for the Continue policies required about 1.5s, whereas less than one

[9] Recall that these BDDs only encode the evaluation of the given policy and, thus, only constrain the values occurring in the policy, which are the same in all three datasets.

		Simplified $[\![\cdot]\!]_{\mathrm{B}}$						Extended $[\![\cdot]\!]_{\mathrm{E}}$					
		BDD_1		BDD_0		$BDD_\perp$		BDD_1		BDD_0		$BDD_\perp$	
		#Vertex	**Depth**	**#Vertex**	**Depth**	**#Vertex**	**Depth**	**#Vertex**	**Depth**	**#Vertex**	**Depth**	**#Vertex**	**Depth**
no constraints	Continue	1085	31	496	29	579	29	1	0	147	24	579	29
	SAFAX (10)	347	24	370	24	7	6	1	0	430	24	7	6
	SAFAX (20)	369	24	407	24	7	6	1	0	450	24	7	6
	SAFAX (50)	343	24	366	24	7	6	1	0	427	24	7	6
constraints	Continue	1156	46	510	46	846	46	594	44	672	46	830	46
	SAFAX (10)	513	54	455	54	108	54	255	54	497	54	108	54
	SAFAX (20)	949	84	920	84	188	84	375	84	762	84	188	84
	SAFAX (50)	1587	174	1551	174	428	174	735	174	1540	174	428	174

Table 4: Overview of the BDDs encoding the simplified $[\![\cdot]\!]_{\mathrm{B}}$ and extended $[\![\cdot]\!]_{\mathrm{E}}$ evaluation with/without constraints

	#Vertex	**Depth**	**#Queries**	**Simplified $[\![\cdot]\!]_{\mathrm{B}}$**			**Extended $[\![\cdot]\!]_{\mathrm{E}}$**		
				BDD_1	BDD_0	$BDD_\perp$	BDD_1	BDD_0	$BDD_\perp$
Continue	63	44	134,631,720	20.09%	32.28%	47.52%	59.10%	41.48%	47.52%
SAFAX (10)	128	54	7,331,148	55.43%	28.39%	16.18%	97.10%	41.87%	16.18%
SAFAX (20)	188	84	51,009,588	55.36%	28.49%	16.18%	97.06%	43.03%	16.18%
SAFAX (50)	368	174	730,641,708	55.27%	28.55%	16.18%	97.04%	42.14%	16.18%

Table 5: BDD encoding constrained query space and percentage of queries that evaluate $1, 0, \perp$ for $[\![\cdot]\!]_{\mathrm{B}}$ and $[\![\cdot]\!]_{\mathrm{E}}$

	Continue	SAFAX (10)	SAFAX (20)	SAFAX (50)
Time (sec)	1.506	0.673	0.985	2.957
Avg. BDD size (KB)	20.33	7.33	12.33	34.33

Table 6: Time needed to construct the BDDs encoding the extended evaluation on the constrained query space and average BDD size

second was required for the SAFAX (10) and SAFAX (20) datasets; SAFAX (50) required slightly less than 3 seconds.

To estimate the memory required to store the generated BDDs, we exploited the functionalities of the dd library. In particular, the dd library makes it possible to dump a BDD to a pickle file. The average size of the dump files is reported in the last row of Table 6. These results suggest that the precomputed BDDs can be stored and evaluated in resource-constrained devices, like IoT devices, to determine whether a user is allowed to access a device's resources.

Extended evaluation function $[\![\cdot]\!]_{\mathrm{E}}$ vs. standard evaluation function $[\![\cdot]\!]_{\mathrm{P}}$. We also evaluated and compared the outcome yielded by evaluation functions $[\![\cdot]\!]_{\mathrm{E}}$ and $[\![\cdot]\!]_{\mathrm{P}}$ (see Section 2). Tables 7 and 8 report the results of this comparison for Continue and SAFAX (10) datasets, respectively. In particular, for each decision, we report the number of queries returned by each evaluation function and how these queries are evaluated by the other evaluation function. From Table 7, we can observe that the Continue policy is vulnerable to attribute hiding attacks that are not captured when evaluating the policy using $[\![\cdot]\!]_{\mathrm{P}}$. In particular, there exist a number of queries for which $[\![\cdot]\!]_{\mathrm{P}}$ grants access while some extensions of those queries should be denied. On the other hand, $[\![\cdot]\!]_{\mathrm{E}}$ is able to identify the risks of attribute hiding attacks by indicating that some extensions of these queries should be denied. Moreover, from the tables, we can observe that, for both policies, the standard evaluation returns decisions than actually cannot be reached. For instance, the evaluation of the Continue policy (Table 7) shows that 972 queries evaluate to $\{1, 0\}$ according to $[\![\cdot]\!]_{\mathrm{P}}$ due to missing information, whereas decision 0 can never be reached for any non-deterministic extension of those queries (which follows from the $\{1, 0\}$ entry in the $\{0\}$ row of the $[\![\cdot]\!]_{\mathrm{E}}$ evaluation). A similar situation can be observed for 106920 queries evaluating to $\{1, \perp\}$ according to $[\![\cdot]\!]_{\mathrm{E}}$.

On the other hand, the differences between $[\![\cdot]\!]_{\mathrm{P}}$ and $[\![\cdot]\!]_{\mathrm{E}}$ are less prominent for the SAFAX policy (Table 8). For instance, the SAFAX policy is not vulnerable to attribute hiding attacks, although we can observe from the table that some queries for which access is denied could be eventually permitted when more information is provided. This result can be explained by recalling that the SAFAX policy employs a number of default *Deny* rules that apply when positive authorizations do not apply. In general, the closeness between for $[\![\cdot]\!]_{\mathrm{P}}$ and $[\![\cdot]\!]_{\mathrm{E}}$ for the SAFAX policy is due to the extensive use of constraints (see Table 3). Specifically, these constraints make several queries that would have resulted in an inconclusive decision (*i.e.*, decisions different from $\{1\}$ or $\{0\}$) invalid. From Section 2, it is easy to observe that $[\![\cdot]\!]_{\mathrm{P}}$ and $[\![\cdot]\!]_{\mathrm{E}}$ have the same behavior over singleton decision.

Discussion. The experiment results show the feasibility and applicability of our approach in real scenarios as well as that the extended evaluation function $[\![\cdot]\!]_{\mathrm{E}}$ provides a more accurate evaluation of ABAC policies compared to standard evaluation function $[\![\cdot]\!]_{\mathrm{P}}$.

Nonetheless, the experiments reveal that query constraints have a significant impact on the extended evaluation function $[\![\cdot]\!]_{\mathrm{E}}$. On the one hand, query constraints improve the accuracy of policy evaluation by removing queries that cannot occur in practice, also reducing the query space. On the other hand, they affect the size of the BDDs representing policy evaluation because invalid queries have to be explicitly encoded in the BDDs.

Another factor that largely influences BDD size is the size of attributes' domains (in combination with query constraints). This

			Standard $[\![\cdot]\!]_P$						
			$\{1\}$	$\{0\}$	$\{\perp\}$	$\{1,\perp\}$	$\{0,\perp\}$	$\{1,0\}$	$\{1,0,\perp\}$
		#Queries	26892810	33023700	41504886	11091006	2519424	10732338	8867556
Extended $[\![\cdot]\!]_E$	$\{1\}$	26578854	26577882	0	0	0	0	972	0
	$\{0\}$	24249456	0	24249456	0	0	0	0	0
	$\{\perp\}$	29865672	0	0	29865672	0	0	0	0
	$\{1,\perp\}$	22336560	0	0	11304846	10924794	0	0	106920
	$\{0,\perp\}$	944784	0	0	0	0	944784	0	0
	$\{1,0\}$	19820538	314928	8774244	0	0	0	10731366	0
	$\{1,0,\perp\}$	10835856	0	0	334368	166212	1574640	0	8760636

Table 7: Comparison between extended evaluation and standard evaluation for the CONTINUE policy.

			Standard $[\![\cdot]\!]_P$						
			$\{1\}$	$\{0\}$	$\{\perp\}$	$\{1,\perp\}$	$\{0,\perp\}$	$\{1,0\}$	$\{1,0,\perp\}$
		#Queries	4063785	1015916	0	0	390104	1065526	795817
Extended $[\![\cdot]\!]_E$	$\{1\}$	4063785	4063785	0	0	0	0	0	0
	$\{0\}$	147499	0	147499	0	0	0	0	0
	$\{\perp\}$	0	0	0	0	0	0	0	0
	$\{1,\perp\}$	197934	0	0	0	0	0	0	197934
	$\{0,\perp\}$	64977	0	0	0	0	64977	0	0
	$\{1,0\}$	1933943	0	868417	0	0	0	1065526	0
	$\{1,0,\perp\}$	923010	0	0	0	0	325127	0	597883

Table 8: Comparison between extended evaluation and standard evaluation for SAFAX (10).

is particularly evident for the SAFAX policy, which contains numerical attributes. In particular, we observe that, in this policy, the number of vertices forming the BDDs increases with the size of attributes' domains and the depth of BDDs is equal to the overall number of attribute values, thus representing the worst case scenario. Nevertheless, the experiments show that our approach remains tractable and it is able to handle such types of policies.

Although in the worst case the number of vertices in a BDD is exponential in the number of variables, in practice the number of vertices is often polynomial [8]. In this respect, the BDD representation used has an impact on the BDD size. The dd library uses a fixed order of variables that is common for all BDDs. This BDD representation can affect the size of the generated BDDs. To reduce the size of BDDs, we used the optimizations offered by the library, namely garbage collection and reordering. Although the use of these optimization provides some benefits in terms of BDD size, we believe that the size of BDDs can be further reduced using different representations (or variants of BDDs), which for instance use a variable order of variables, or by optimizing the order of variables for each BDD independently.

7 RELATED WORK

Attribute-based access control has gained increasing popularity in the last years due to its flexibility and expressiveness. Several mechanisms for the evaluation and enforcement of ABAC policies have been proposed in both academia and industry, especially for XACML [13, 14]. Examples of these mechanisms are SUN-XACML[10], HERAS-AF [7], XEngine [11], enterprise-java-xacml[11] and WSO2 Balana[12]. These mechanisms evaluate policies according to the standard evaluation function. As discussed in Section 2, this function can yield decisions that do not correspond to an intuitive interpretation of what these decisions means due to missing information.

Tschantz and Krishnamurthi introduced in [16] the problem of missing information, and Crampton and Morisset developed in [4] the notion of attribute-hiding attacks for PTaCL and proposed different restrictions on the definition of a target to prevent such attacks. A different approach to address the problem of missing information is presented in [5], where all queries that can be constructed from the initial query are evaluated to account that attributes could have been hidden, using the PRISM model-checker. Model-checking has been used in the past for access control, for instance Zhang et al. [18] propose a tool checking whether a particular goal can be reached within an access control policy, but not in the context of missing information for ABAC. However, the query space could potentially consist of a huge number of states and its exploration at evaluation time is not practical in real settings. In this work, we improve on [5] by studying how to efficiently compute the extended evaluation of policies while considering more expressive domain constraints.

Recently, Turkmen et al. [17] have proposed a policy analysis framework for XACML policies based on SMT. The framework supports the verification of a large range of properties including the robustness of XACML policies against two types of attribute hiding attacks, namely partial attribute hiding and general attribute hiding. Partial attribute hiding analyzes the case where a user hides a single attribute name-value pair, whereas general attribute hiding extends partial attribute hiding by assuming that a user completely suppresses information about one attribute. However, this work only allows verifying whether a policy is vulnerable to attribute hiding attacks.

In this work, we have proposed the use of binary decision diagram (BDD)-based data structures for the representation of ABAC policies. We are not the first that use such data structures in the context of ABAC. For instance, Hu et al. [9] use BDDs to determine the applicability of policies, whereas other researchers [1, 8] propose an

[10]http://sunxacml.sourceforge.net
[11]http://code.google.com/p/enterprise-java-xacml
[12]http://xacmlinfo.org/category/balana

encoding of ABAC policies using Multi-Terminal BDDs (MTBDDs). Although the use of BDD-based data structures presented in our work shares several similarities with these works, there also several differences. Similarly to our work, these proposals construct BDDs (or MTBDDs) from the policy specification. However, they encode policy evaluation according to the standard evaluation function, which, as discussed in Section 2, is not able to handle missing information properly. Moreover, these approaches typically neglect domain constraints. As shown in Section 6, this can result in misleading decisions. To the best of our knowledge, the only approach that address this issue is Margrave [8], a formal framework for the analysis of XACML policies. In Margrave, domain constraints are incorporated by introducing a terminal node representing queries that do not satisfy the constraints. In our work, we encoded constraints in a separated BDD, which is combined with the BDDs encoding the simplified evaluation of a policy when computing the extended evaluation of the policy.

8 CONCLUSION

The ABAC paradigm is gaining more and more attention due to its flexibility and expressiveness. However, it has been shown that the current way standard ABAC mechanisms (*e.g.*, XACML) handle missing information is flawed, making ABAC policies vulnerable to attribute hiding attacks. Previous work [5] has addressed this issue by providing a novel approach to the evaluation of ABAC policies. However, a naïve implementation of this approach would require exploring the state space for all possible queries, which is exponential in the number of attribute values, and therefore not feasible in practice. In this work, we have presented an efficient method for the computation of the extended evaluation of ABAC policies. The method uses the BDD representation of the policies to compute the extended evaluation directly on the BDD structure. Moreover, we have investigated the use of query constraints to obtain more accurate decisions. We have demonstrated our approach using two real-world policies. The results show that the extended evaluation can be computed in a few seconds and the corresponding BDDs do not require considerable memory for storage. Moreover, the results show that the extended evaluation is able to identify the risk of attribute hiding attacks and provides more accurate decisions with respect to the standard evaluation.

As future work, we plan to extend our approach to support a probabilistic evaluation of ABAC policies. Intuitively, we would like to determine the probability that a certain decision can be reached through the exploration of the (constrained) query space. Moreover, we want to explore the use of other variants of BDD for the representation of the extended evaluation of ABAC policies. In particular, Multi-valued Decision Diagrams (MDDs) and Multi-Terminal BDDs (MTBDDs), as well as their combination, could be suitable alternatives for our purposes. On the one end, MTBDDs allow specifying a single decision diagram encoding all decisions, instead of three separated BDDs, one for each (singleton) decision. On the other hand, MDDs supports the encoding of multi-valued predicates. This would make it possible to bound the depth of decision diagrams to the number of attributes rather to the number of attribute values, thus reducing the time required for policy evaluation.

Acknowledgments. This work is partially funded by the ITEA projects M2MGrids (13011) and APPSTACLE (15017).

REFERENCES

[1] B. Bahrak, A. Deshpande, M. Whitaker, and J. Park. 2010. BRESAP: A Policy Reasoner for Processing Spectrum Access Policies Represented by Binary Decision Diagrams. In *Proc. of Symp. on New Frontiers in Dynamic Spectrum.* IEEE, 1–12.

[2] R. E. Bryant. 1992. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.* 24, 3 (1992), 293–318.

[3] J. Crampton and M. Huth. 2010. An Authorization Framework Resilient to Policy Evaluation Failures. In *Computer Security.* Springer, 472–487.

[4] J. Crampton and C. Morisset. 2012. PTaCL: A Language for Attribute-Based Access Control in Open Systems. In *Principles of Security and Trust (LNCS 7215).* Springer, 390–409.

[5] J. Crampton, C. Morisset, and N. Zannone. 2015. On Missing Attributes in Access Control: Non-deterministic and Probabilistic Attribute Retrieval. In *Proc. of Symposium on Access Control Models and Technologies.* ACM, 99–109.

[6] J. Crampton and C. Williams. 2016. On Completeness in Languages for Attribute-Based Access Control. In *Proc. of Symposium on Access Control Models and Technologies.* ACM, 149–160.

[7] S. Dolski, F. Huonder, and S. Oberholzer. 2007. *HERAS-AF: XACML 2.0 Implementation.* Tech. Rep. University of Applied Sciences Rapperswil.

[8] K. Fisler, S. Krishnamurthi, L. Meyerovich, and M. Tschantz. 2005. Verification and Change-impact Analysis of Access-control Policies. In *Proc. of Int. Conference on Software Engineering.* ACM, 196–205.

[9] H. Hu, G. Ahn, and K. Kulkarni. 2013. Discovery and Resolution of Anomalies in Web Access Control Policies. *TDSC* 10, 6 (2013), 341–354.

[10] S. P. Kaluvuri, A. I. Egner, J. den Hartog, and N. Zannone. 2015. SAFAX - An Extensible Authorization Service for Cloud Environments. *Front. ICT* (2015).

[11] A. Liu, F. Chen, J. Hwang, and T. Xie. 2011. Designing Fast and Scalable XACML Policy Evaluation Engines. *IEEE Trans. Computers* 60, 12 (2011), 1802–1817.

[12] C. Morisset and N. Zannone. 2014. Reduction of access control decisions. In *Proc. of Symposium on Access Control Models and Technologies.* ACM, 53–62.

[13] OASIS. 2005. *eXtensible Access Control Markup Language (XACML) Version 2.0.* OASIS Standard.

[14] OASIS. 2013. *eXtensible Access Control Markup Language (XACML) Version 3.0.* OASIS Standard.

[15] R. Rudell. 1993. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. of International Conference on Computer Aided Design.* IEEE, 42–47.

[16] M. Tschantz and S. Krishnamurthi. 2006. Towards reasonability properties for access-control policy languages. In *Proc. of Symposium on Access Control Models and Technologies.* ACM, 160–169.

[17] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone. 2017. Formal analysis of XACML policies using SMT. *Computers & Security* 66 (2017), 185–203.

[18] N. Zhang, M. Ryan, and D. Guelev. 2005. Evaluating Access Control Policies Through Model Checking. In *Information Security.* Springer, 446–460.

A TRANSFORMATION RULES FOR DECISIONS 0 AND $\perp$

Tables 9 and 10 present the transformation rules for τ (for targets) and π (for policies) for decisions 0 and $\perp$, respectively. We note that using a simple structural induction, one can show that both $\tau_\perp(t) = \neg(\tau_1(t) \vee \tau_0(t))$ and $\pi_\perp(p) = \neg(\pi_1(p) \vee \pi_0(p))$. Thus, in our proofs, we can focus on the cases $d = 1$ and $d = 0$.

B PROOF OF THEOREM 3

Note that the semantics of a propositional formula is given in the context of an interpretation $\eta : Vars \to \mathbb{B}$, assigning meaning to variables. Let $\eta : Vars \to \mathbb{B}$ be such an interpretation. We write $\eta \models \phi$ for propositional formula ϕ ranging over $Vars$ iff ϕ holds under interpretation η. A query induces an interpretation $I : Q_{\mathcal{A}} \to (Vars \to \mathbb{B})$, given by $I(q)(a_v) = true$ iff $(a, v) \in q$.

The correctness of our algorithm essentially hinges on two lemmata, which we present next. The first one states that transformation τ faithfully characterizes sets of queries, whereas the second one states that transformation τ correctly encodes the simplified evaluation of the policy language.

$\tau_0((a,v))$	$=$	$\neg a_v \wedge \bigvee\{a_{v'} \mid v' \in \mathcal{V}_{\mathcal{A}}\}$
$\tau_0(\neg t_1)$	$=$	$\tau_1(t_1)$
$\tau_0(\sim t_1)$	$=$	$\tau_0(t_1) \vee \tau_\perp(t_1)$
$\tau_0(E_1(t_1))$	$=$	$\tau_0(t_1)$
$\tau_0(t_1 \tilde{\sqcap}\, t_2)$	$=$	$\tau_0(t_1) \vee \tau_0(t_2)$
$\tau_0(t_1 \sqcap t_2)$	$=$	$(\tau_0(t_1) \wedge \neg\tau_\perp(t_2)) \vee (\tau_0(t_2) \wedge \neg\tau_\perp(t_1))$
$\tau_0(t_1 \vartriangle t_2)$	$=$	$\tau_0(t_1) \vee \tau_0(t_2)$
$\tau_0(t_1 \tilde{\sqcup}\, t_2)$	$=$	$\tau_0(t_1) \wedge \tau_0(t_2)$
$\tau_0(t_1 \sqcup t_2)$	$=$	$\tau_0(t_1) \wedge \tau_0(t_2)$
$\tau_0(t_1 \triangledown t_2)$	$=$	$(\tau_0(t_1) \wedge \neg\tau_1(t_2)) \vee (\tau_0(t_2) \wedge \neg\tau_1(t_1))$
$\pi_0(1)$	$=$	*false*
$\pi_0(0)$	$=$	*true*
$\pi_0((t,p_1))$	$=$	$\tau_1(t) \wedge \pi_0(p_1)$
$\pi_0(\neg p_1)$	$=$	$\pi_1(p_1)$
$\pi_0(\sim p_1)$	$=$	$\pi_0(p_1) \vee \pi_\perp(p_1)$
$\pi_0(E_1(p_1))$	$=$	$\pi_0(p_1)$
$\pi_0(p_1 \tilde{\sqcap}\, p_2)$	$=$	$\pi_0(p_1) \vee \pi_0(p_2)$
$\pi_0(p_1 \sqcap p_2)$	$=$	$(\pi_0(p_1) \wedge \neg\pi_\perp(p_2)) \vee (\pi_0(p_2) \wedge \neg\pi_\perp(p_1))$
$\pi_0(p_1 \vartriangle p_2)$	$=$	$\pi_0(p_1) \vee \pi_0(p_2)$
$\pi_0(p_1 \tilde{\sqcup}\, p_2)$	$=$	$\pi_0(p_1) \wedge \pi_0(p_2)$
$\pi_0(p_1 \sqcup p_2)$	$=$	$\pi_0(p_1) \wedge \pi_0(p_2)$
$\pi_0(p_1 \triangledown p_2)$	$=$	$(\pi_0(p_1) \wedge \neg\pi_1(p_2)) \vee (\pi_0(p_2) \wedge \neg\pi_1(p_1))$

Table 9: Transformation rules for τ_0 (for targets) and π_0 (for policies) for decision 0

$\tau_\perp((a,v))$	$=$	$\bigwedge\{\neg a_{v'} \mid v' \in \mathcal{V}_{\mathcal{A}}\}$
$\tau_\perp(\neg t_1)$	$=$	$\tau_\perp(t_1)$
$\tau_\perp(\sim t_1)$	$=$	*false*
$\tau_\perp(E_1(t_1))$	$=$	$\tau_1(t_1)$
$\tau_\perp(t_1 \tilde{\sqcap}\, t_2)$	$=$	$(\tau_\perp(t_1) \wedge \neg\tau_0(t_2)) \vee (\tau_\perp(t_2) \wedge \neg\tau_0(t_1))$
$\tau_\perp(t_1 \sqcap t_2)$	$=$	$\tau_\perp(t_1) \vee \tau_\perp(t_2)$
$\tau_\perp(t_1 \vartriangle t_2)$	$=$	$\tau_\perp(t_1) \wedge \tau_\perp(t_2)$
$\tau_\perp(t_1 \tilde{\sqcup}\, t_2)$	$=$	$(\tau_\perp(t_1) \wedge \neg\tau_1(t_2)) \vee (\tau_\perp(t_2) \wedge \neg\tau_1(t_1))$
$\tau_\perp(t_1 \sqcup t_2)$	$=$	$\tau_\perp(t_1) \vee \tau_\perp(t_2)$
$\tau_\perp(t_1 \triangledown t_2)$	$=$	$\tau_\perp(t_1) \wedge \tau_\perp(t_2)$
$\pi_\perp(1)$	$=$	*false*
$\pi_\perp(0)$	$=$	*false*
$\pi_\perp((t,p_1))$	$=$	$\tau_0(t) \vee \tau_\perp(t) \vee (\tau_1(t) \wedge \pi_\perp(p_1))$
$\pi_\perp(\neg p_1)$	$=$	$\pi_\perp(p_1)$
$\pi_\perp(\sim p_1)$	$=$	*false*
$\pi_\perp(E_1(p_1))$	$=$	$\pi_1(p_1)$
$\pi_\perp(p_1 \tilde{\sqcap}\, p_2)$	$=$	$(\pi_\perp(p_1) \wedge \neg\pi_0(p_2)) \vee (\pi_\perp(p_2) \wedge \neg\pi_0(p_1))$
$\pi_\perp(p_1 \sqcap p_2)$	$=$	$\pi_\perp(p_1) \vee \pi_\perp(p_2)$
$\pi_\perp(p_1 \vartriangle p_2)$	$=$	$\pi_\perp(p_1) \wedge \pi_\perp(p_2)$
$\pi_\perp(p_1 \tilde{\sqcup}\, p_2)$	$=$	$(\pi_\perp(p_1) \wedge \neg\pi_1(p_2)) \vee (\pi_\perp(p_2) \wedge \neg\pi_1(p_1))$
$\pi_\perp(p_1 \sqcup p_2)$	$=$	$\pi_\perp(p_1) \vee \pi_\perp(p_2)$
$\pi_\perp(p_1 \triangledown p_2)$	$=$	$\pi_\perp(p_1) \wedge \pi_\perp(p_2)$

Table 10: Transformation rules for $\tau_\perp$ (for targets) and $\pi_\perp$ (for policies) for decision $\perp$

LEMMA 1(A). *For all $q \in Q_{\mathcal{A}}$, $I(q) \models \tau_d(t)$ iff $d = [\![t]\!]_{\mathrm{T}}(q)$.*

PROOF. By structural induction on t. Let $q \in Q_{\mathcal{A}}$ be arbitrary.

- *Base case*: $t \equiv (a,v)$. We prove correctness for each $d \in \{1,0\}$ separately (Recall that case $d = \perp$ follows from $d=1$ and $d=0$).
  - Case $d = 1$. Suppose $I(q) \models \tau_1((a,v))$. By definition, $\tau_1((a,v)) = a_v$. From this, it follows that $I(q) \models a_v$ which, by definition means that $I(q)(a_v) = \mathit{true}$ and, thus, $(a,v) \in q$. By definition of $[\![\cdot]\!]_{\mathrm{T}}$ we also have $[\![(a,v)]\!]_{\mathrm{T}}(q) = 1$.
  - Case $d = 0$ follows identical reasoning using Table 9.
- *Induction hypothesis*: suppose that, for all d', $I(q) \models \tau_{d'}(t_i)$ iff $d' = [\![t_i]\!]_{\mathrm{T}}(q)$ with $i \in \{1,2\}$. We need to consider all unary and binary operators and prove each equivalence for all $d \in \{1,0\}$. We provide details for negation $\neg$ and strong conjunction $\tilde{\sqcap}$; the proofs for all remaining operators are analogous and therefore omitted.
  - Suppose $t \equiv \neg t_1$. We compute:

$I(q) \models \tau_1(\neg t_1)$	$I(q) \models \tau_0(\neg t_1)$
iff {by def.} $I(q) \models \tau_0(t_1)$	iff {by def.} $I(q) \models \tau_1(t_1)$
iff {by induction} $0 = [\![t_1]\!]_{\mathrm{T}}(q)$	iff {by induction} $1 = [\![t_1]\!]_{\mathrm{T}}(q)$
iff {by def. of $\neg$} $[\![\neg t_1]\!]_{\mathrm{T}} = 1$	iff {by def. of $\neg$} $[\![\neg t_1]\!]_{\mathrm{T}} = 0$

  - Suppose $t \equiv t_1 \tilde{\sqcap}\, t_2$. We compute for $d = 1$:

$$\begin{array}{ll} I(q) \models \tau_1(t_1 \tilde{\sqcap}\, t_2) & \text{iff \{by def.\}}\ I(q) \models \tau_1(t_1) \wedge \tau_1(t_2) \\ & \text{iff \{by def.\}}\ I(q) \models \tau_1(t_1) \text{ and } I(q) \models \tau_1(t_2) \\ & \text{iff \{by induction (2x)\}}\ 1 = [\![t_1]\!]_{\mathrm{T}}(q) \text{ and } 1 = [\![t_2]\!]_{\mathrm{T}}(q) \\ & \text{iff \{by def.\}}\ 1 = [\![t_1 \tilde{\sqcap}\, t_2]\!]_{\mathrm{T}}(q) \end{array}$$

  Case $d = 0$ follows the same reasoning, employing the encodings of Table 9. □

LEMMA 1(B). *For all $q \in Q_{\mathcal{A}}$, $I(q) \models \pi_d(p)$ iff $d = [\![p]\!]_{\mathrm{B}}(q)$.*

PROOF. The proof of this lemma proceeds by induction on the structure of the policy. Since the proof bears many similarities to that of the previous lemma, we only highlight the interesting case, which is the case $p \equiv (t, p_1)$. Assume, as our induction hypothesis, that for all d', $I(q) \models \pi_{d'}(p_1)$ iff $d' = [\![p_1]\!]_{\mathrm{B}}(q)$.

We separately prove the statement for $d \in \{1,0\}$. We reason as follows:

$I(q) \models \pi_1((t,p_1))$	$I(q) \models \pi_0((t,p_1))$
iff {by def.}	iff {by def.}
$I(q) \models \tau_1(t) \wedge \pi_1(p_1)$	$I(q) \models \tau_1(t) \wedge \pi_0(p_1)$
iff {by def.}	iff {by def.}
$I(q) \models \tau_1(t)$ and $I(q) \models \pi_1(p_1)$	$I(q) \models \tau_1(t)$ and $I(q) \models \pi_0(p_1)$
iff {by induction}	iff {by induction}
$I(q) \models \tau_1(t)$ and $1 = [\![p_1]\!]_{\mathrm{B}}(q)$	$I(q) \models \tau_1(t)$ and $0 = [\![p_1]\!]_{\mathrm{B}}(q)$
iff {by Lemma 1(a)}	iff {by Lemma 1(a)}
$[\![t]\!]_{\mathrm{T}}(q) = 1$ and $1 = [\![p_1]\!]_{\mathrm{B}}(q)$	$[\![t]\!]_{\mathrm{T}}(q) = 1$ and $0 = [\![p_1]\!]_{\mathrm{B}}(q)$
iff {by def.}	iff {by def.}
$1 = [\![(t,p_1)]\!]_{\mathrm{B}}(q)$	$0 = [\![(t,p_1)]\!]_{\mathrm{B}}(q)$ □

Finally, we observe that the proposition $\bar{R}$, defined as $\bigwedge\{a_v \Longrightarrow a'_v \mid a_v \in \mathit{Vars}_{\mathcal{A}}\}$, indeed encodes the subset relation on $Q_{\mathcal{A}}$. We introduce an interpretation $I' : Q_{\mathcal{A}} \to (\mathit{Vars}' \to \mathbb{B})$, which is given by $I'(q)(a'_v) = \mathit{true}$ iff $(a,v) \in q$. We write $\eta \cup \eta' \models \bar{R}$ iff $\bar{R}$ holds under interpretation $\eta : \mathit{Vars} \to \mathbb{B}$ for variables from Vars and $\eta' : \mathit{Vars}' \to \mathbb{B}$ for variables from Vars'.

LEMMA 2. *For all $q, q' \in Q_{\mathcal{A}}$, $I(q) \cup I'(q') \models \bar{R}$ iff $(q,q') \in \to^*$.*

PROOF. First, observe that $\to^*$ is in fact equivalent to $\subseteq$ on $Q_{\mathcal{A}}$.

- Implication from left to right. Suppose $I(q) \cup I'(q') \models \bar{R}$. Then, $I(q) \cup I'(q') \models \bigwedge\{a_v \Longrightarrow a'_v \mid a_v \in \mathit{Vars}_{\mathcal{A}}\}$, and, therefore, for all $a_v \in \mathit{Vars}_{\mathcal{A}}$, we find that $I(q) \cup I'(q') \models a_v \Longrightarrow a'_v$. But then if $I(q)(a_v)$ holds, then so does $I'(q')(a'_v)$. By definition, this means $(a,v) \in q$ implies $(a,v) \in q'$ for all $(a,v) \in Q_{\mathcal{A}}$. But then $q \subseteq q'$, or, equivalently $(q,q') \to^*$.
- Implication from right to left. Suppose $(q,q') \in \to^*$, or, equivalently, $q \subseteq q'$. Pick some arbitrary $(a,v) \in Q_{\mathcal{A}}$, and assume $(a,v) \in q$. By definition, we then have $I(q)(a_v)$ holds. Since $q \subseteq q'$, also $(a,v) \in q'$; but then also $I'(q)(a_v)$ holds. So we have $I(q)(a_v)$ implies $I'(q)(a'_v)$. But then $I(q) \cup I'(q') \models a_v \Longrightarrow a'_v$. Since we picked $(a,v) \in Q_{\mathcal{A}}$ arbitrary, we find that $I(q) \cup I'(q') \models \bigwedge\{a_v \Longrightarrow a'_v \mid a_v \in \mathit{Vars}_{\mathcal{A}}\}$. □

The correctness of procedure COMPUTEEXTENDEDEVALUATION (Theorem 3) directly follows from the next proposition, where $R = \bar{R} \wedge S \wedge S[\mathit{Vars}_{\mathcal{A}} := \mathit{Vars}'_{\mathcal{A}}]$ and $S = \bigwedge\{\tau_1(c) \mid c \in C\}$:

PROPOSITION 4. *For all $d \in \{1,0,\perp\}$, $q \in Q_{\mathcal{A}|C} \wedge d \in [\![p]\!]_{\mathrm{E}}(q)$ iff $I(q) \models (\pi_d(p) \wedge S) \vee \exists \mathit{Vars}'_{\mathcal{A}}.(R \wedge (\pi_d(p))[\mathit{Vars}_{\mathcal{A}} := \mathit{Vars}'_{\mathcal{A}}])$.*

PROOF. Follows from Lemmata 1(a), 1(b) and 2. □