TBA: A Hybrid of Logic and Extensional Access Control Systems

Extended Version

Timothy L. Hinrichs[1], William C. Garrison III[2], Adam J. Lee[2], Skip Saunders[3], and John C. Mitchell[4]

[1] University of Chicago
[2] University of Pittsburgh
[3] MITRE Corporation
[4] Stanford University

Abstract. Logical policy-based access control models are greatly expressive and thus provide the flexibility for administrators to represent a wide variety of authorization policies. Extensional access control models, on the other hand, utilize simple data structures to better enable a less trained and non-administrative workforce to participate in the day-to-day operations of the system. In this paper, we formally study a hybrid approach, *tag-based authorization* (TBA), which combines the ease of use of extensional systems while still maintaining a meaningful degree of the expressiveness of logical systems. TBA employs an extensional data structure to represent metadata tags associated with subjects and objects, as well as a logical language for defining the access control policy in terms of those tags. We formally define TBA and introduce variants that include tag ontologies and delegation. We evaluate the resulting system by comparing to well-known extensional and logical access control models.

1 Introduction

Logical access control systems, in which users write formal logic to express access control policies, are expressive and supremely flexible but are hard to use because they require fluency in formal logic. Extensional access control systems (*e.g.*, the access matrix, role-based access control, Bell-La Padula), in which users enter atomic values (*e.g.*, roles, rights, classifications) into simple data structures (*e.g.*, a matrix or a pair of binary relations), are in contrast easy to use but are far less flexible. Judging from the prominence of extensional approaches in real-world organizations, ease-of-use is more important than flexibility; nevertheless, the problems with extensional systems are well known and can be addressed to a large extent with the flexibility of logical access control systems. Thus, a hybrid approach to access control that achieves the flexibility of logic and the usability of extensional systems would serve the community well.

As a case in point, MITRE recently published a report outlining the problems the U.S. military has had with their extensional access control system in the context of dynamic coalitions [28]. The main problem is the frequency with which partner countries enter and leave coalitions, causing the U.S. to make massive, frequent changes

to its authorization policy. Logical access control systems are better suited to making large, frequent changes than extensional systems and so are an attractive alternative to the current system; however, it seems clear that the entire military cannot be trained to write formal logic in the near future. The right solution seems to be a combination of extensional and logical systems that allows relatively untrained personnel to create and contribute data while trained security experts write formal logic to express the desired access control policy.

Building a hybrid access control system that combines logic and extensionality is hard because the simplicity—and therefore usability—of extensional systems appears fundamentally at odds with logic's flexiblity. Extensionality's simplicity comes from its rigid commitment to a single representation of an access control policy, *e.g.*, RBAC grants subject s access to permission p when $\exists r.UR(s,r) \wedge PA(r,p)$. Logic's flexibility comes from its ability to represent a single policy in a myriad of ways—allowing security experts to choose the form best suited for supporting new and unforeseen demands. A hybrid system must therefore concede some of its flexibility by committing to a single representation for some component of the access control policy and must also concede some of its simplicity by allowing multiple representations of the policy.

In this paper, we formally study tag-based authorization (TBA), a hybrid access control system that combines the flexibility of logical access control systems and the usability of extensional systems. Relatively untrained people choose descriptive tags for the system's subjects and objects (similar to the tagging employed by many popular and successful web applications, such as Flickr and YouTube), and trained security experts write logical policies that define access permissions using combinations of subject and object tags (Section 2). One step we take to make TBA flexible yet easy to use is including delegation but separating delegation decisions from access control decisions. We replace delegation primitives inside the policy language with a scheme for combining policies outside the language (Section 3). We introduce a simple algorithm for TBA that is sound and complete and runs in polynomial time under certain conditions (Section 4). We evaluate TBA by demonstrating its ability to express a number of well-known access control paradigms (Section 5). Finally we discuss related work (Section 6) and conclude (Section 7).

2 Tag-based Authorization

Tag-based authorization combines the strengths of logical access control systems and extensional access control systems. Just as with logical access control, formal logic is used to describe the authorization policy. Just as with extensional access control, subjects and objects are ascribed a small set of simple properties when they are added to the system (*e.g.*, roles in Role-based Access Control or classifications and clearances in Bell-La Padula). The properties ascribed to subjects and objects are tags that capture all of the security-relevant properties of that subject or object. The authorization policy is defined in terms of tags: it dictates which subject tags are sufficient for which rights to which object tags. Because of the simplicity of tagging, relatively untrained users can tag subjects and objects, while a relatively small number of administrators write the logical authorization policy.

Formally, we use S to denote the set of subjects, O to denote the set of objects, and R to denote the set of rights. T denotes the set of possible tags, and tag denotes the function that maps subjects and objects to tag sets: $tag : S \cup O \rightarrow 2^T$. Tag denotes the set of all possible tag functions.

An authorization policy is written in some logical access control language $\langle \mathcal{P}, \mathcal{L}, \models \rangle$. $\mathcal{P}$ is the set of all authorization policies; $\mathcal{L}$ is the set of queries, which we assume always includes $allow(s, o, r)$ for all subjects s, objects o, and rights r; $\models$ dictates which queries are true given an authorization policy and a tag function.

Definition 1 (Tag-based authorization (TBA)). *For a logical language $\langle \mathcal{P}, \mathcal{L}, \models \rangle$, a policy $\Delta \in \mathcal{P}$, and a tag function tag where*

- *$\mathcal{P}$: the set of all authorization policies*
- *$\mathcal{L}$: the set of queries including $allow(s, o, r)$ for all subjects s, objects o, rights r*
- *$\models$: a subset of $\mathcal{P} \times Tag \times \mathcal{L}$*

$$auth_{TBA}(s, o, r) \text{ if and only if } \Delta, tag \models allow(s, o, r)$$

The following example illustrates TBA using DATALOG as the policy language.

Example 1 (Basic Tag-Based Authorization). Consider two subjects—s_1 and s_2—and two objects—o_1 and o_2—that are tagged as follows:

- $tag(s_1) = \{US, Army, enduring\_freedom, signals\}$
- $tag(s_2) = \{France, Navy\}$
- $tag(o_1) = \{submarine, radar\}$
- $tag(o_2) = \{Kandahar, sat\_732, high\_res\}$

Further, consider the following policy.

$allow(S, O, read) :- US \in tag(S), Navy \in tag(S), submarine \in tag(O)$
$allow(S, O, read) :- France \in tag(S), Navy \in tag(S), submarine \in tag(O)$
$allow(S, O, read) :- signals \in tag(S), submarine \in tag(O)$
$allow(S, O, read) :- US \in tag(S), enduring\_freedom \in tag(S),$
$high\_res \in tag(O), sat\_732 \in tag(O)$

This policy allows U.S. and French naval officers to access documents about submarines (via rules 1 and 2), all signals officers to access documents about radar systems (rule 3), and all members of the U.S. military serving on Operation Enduring Freedom to access high resolution satellite photographs taken by sat_732 (rule 4). As a result, subject s_1 can access objects o_1 and o_2, while subject s_2 can only access object o_1. ■

Tag-based authorization differs from standard logical access control systems in that tag has a fixed semantics and is defined outside of the policy. The fixed semantics of tag forces policy-writers to define an authorization policy at a higher level of abstraction than the usual $S \times O \times R$. Policies in TBA are really concerned with access control decisions over the space of tags where subjects and objects are replaced by tag sets: $2^T \times 2^T \times R$. This abstraction results in a less flexible system since tag-space may not be the right one for a particular situation; however, the loss of flexibility is the price of a more understandable system for the majority of users. Relatively untrained users can contribute to the system by changing tag, yet trained administrators can utilize the flexibility of logic for expressing an access control policy. Thus, TBA enables a more thorough utilization of the spectrum of skills present in a typical workforce.

2.1 Tag Ontologies

One of TBA's limitations is that the number of relevant tags for a given subject or object can be large and must be managed properly to ensure that (i) everyone uses the same tags to mean the same thing and (ii) people are not routinely forced to tag subjects/objects with hundreds or thousands of tags, *e.g.*, the tag $boat$ may imply the tag $aquatic$, $aquatic$ might imply $vehicle$, and so on.

Both to help people reach consensus on tag meanings and to reduce the burden of document tagging, we propose employing an ontology (*e.g.*, [22]) to encode the relationships among tags. An ontology is helpful in the context of TBA in three ways. First, an ontology states which tags imply other tags, thereby reducing the number of tags that must be explicitly assigned to a subject or object; all tags implied are implicitly included, *e.g.*, tagging an object with $boat$ implicitly includes the tags $aquatic$ and $vehicle$. Second, an ontology simplifies policy-writing because it states that some tag combinations are illegal, *e.g.*, $short$ and $tall$, and the policy need not cover illegal tag combinations. Third, an ontology helps people communicate the meanings of tags because it explicitly states the relationships to other tags, *e.g.*, if bat implies $animal$, it is clear that bat refers to an animal instead of sports equipment.

Formally, a tag ontology Γ is a set of statements in propositional logic where the propositions are tags. A set of tags G is a legal combination whenever $G \cup \Gamma$ is logically consistent. The set of tags implied by some tag set G is the set of all t such that $G \cup \Gamma$ entails t, denoted $Cn^{\Gamma}(G)$. We use $Cn^{\Gamma}(tag)$ to denote the application of Cn to all tag sets in the tag function tag.

Employing ontologies leads to a new version of tag-based authorization.

Definition 2 (Ontology-aided TBA). *Suppose Δ is a TBA premise set, tag is a tag function, and Γ is an ontology. For every $x \in S \cup O$, $tag(x) \cup \Gamma$ must be consistent.*

$$auth_{TBA}(s, o, r) \text{ iff } \Delta, Cn^{\Gamma}(tag) \models auth(s, o, r)$$

Example 2 (Ontology-Aided TBA). Consider a system containing some subject s and some object o that can be described as follows:

- $tag(s) = \{France, Navy\}$
- $tag(o) = \{submarine, radar\}$

Further, assume that the $policy$ is the following DATALOG.

$$auth(S, O, read) :- France \in tag(S), Navy \in tag(S), watercraft \in tag(O)$$

Intuitively, this policy allows French naval officers access to documents about watercrafts. In the basic tag-based authorization model, subject s would be denied access to object o because o is not explicitly tagged as a document about watercrafts. However, given a tag ontology containing the assertion $submarine \Rightarrow watercraft$, subject s would be permitted access because $tag(o)$ would implicitly include $watercraft$. ■

Ontology-aided TBA further enables all classes of users to contribute to the running of the system. Untrained personnel, who contribute mainly through generating and tagging data, can do so with even less effort thanks to the ability to tag with a smaller number of more specific tags. Administrative personnel also benefit because they can ignore incompatible tag combinations, inevitably leading to shorter policies.

3 Delegation

TBA employs a single logical policy to represent all access control decisions, but often that single policy is derived from many conceptually separate policies written by different security experts. The standard approach to providing the illusion of a single policy from multiple disparate policies is to include delegation primitives in the logical policy language that dictate how the disparate policies are to be combined, *e.g.*, [2, 20]. This approach is supremely flexible. For example, policy A might import policy B's decisions as long as policy C imports policy D's decisions on a specific topic. The downside to adding delegation to the language is that it can be difficult to understand how a given set of policies contribute to the overall policy—it might require reasoning about the logical consequences of all of the policies at once; moreover, small changes to any one policy may radically alter how the policies are pieced together.

Instead of adding delegation *inside* the logical language, TBA adds delegation *outside* of the logical language, thereby separating delegation decisions from access control decisions. In particular, we utilize constructs that arrange a set of policies into a partial order, where if $A \prec B$ then B delegates to A. Not only is this form of delegation especially simple to understand, it allows different security experts to choose different logical languages for writing their policies. The only restriction is that all of the logical languages used in the partial order must make access control decisions that are axiomatizable in a common logical language; otherwise, there would be no way to combine the access control decisions made by distinct policies. We call one of these partial orders of policies a *structured policy*.

More precisely, a structured policy is comprised of (i) a set of basic policies, (ii) a partial order of those policies, (iii) a set of guards on the partial order, (iv) a metalanguage in which access control decisions are axiomatizable, and (v) a conflict resolution operator. A partial order over the policies enables delegation and *implicitly* imposes limits on the decisions delegated; the guards on the partial order *explicitly* limit the decisions that are delegated. If policy A is greater in the partial order than B then A delegates to B the decisions A does not make, and if that delegation is guarded by G then B's actual decisions are limited to those described by G. Because the ordering on policies is partial instead of total, some access control decisions are ambiguous, and the conflict resolution mechanism is used to disambiguate such decisions.

For example, in the U.S. military, basic policies might be written by the President, his chiefs of staff, and others. The partial order includes a single maximal policy: the President's. If the President allows or denies a request, the decision has been made; otherwise, the chiefs of staff have the opportunity to make a decision. The Army chief of staff is restricted from making decisions the Air Force chief of staff ought to make because of guards that restrict the Army to the Army-pertinent decisions and the Air Force to the Air Force-pertinent decisions. If the Army and Air Force make opposing decisions about a request that is pertinent to them both, the conflict resolution mechanism dictates whose decision will be enforced.

Definition 3 (Structured Policy). *A structured policy is a five-tuple* $\langle P, \prec, G, N, res \rangle$.

- P*: a finite set of basic policies. If policy* q *is written in logical language* $\langle \mathcal{P}_q, \mathcal{L}_q, \models_q \rangle$ *then both* $allow(s, o, r)$ *and* $deny(s, o, r)$ *belong to* $\mathcal{L}_q$ *for all* s*,* o*,* r*.*

- $\prec$*: a binary relation over* P *whose transitive closure is irreflexive (*i.e.*, no cycles)*
- G*: a set of functions* $guard_{B \prec C} : S \times O \times R \to \{true, false\}$ *for every* $B \prec C$
- N*: the meta language,* i.e.*, a logical language* $\langle \mathcal{P}^*, \mathcal{L}^*, \models^* \rangle$ *such that*
  - *all subsets of* $\bigcup_{q \in P} \mathcal{L}_q$ *are included in* $\mathcal{P}^*$
  - $\mathcal{L}^*$ *includes* $allow(s, o, r)$ *and* $deny(s, o, r)$ *for all s,o,r.*
- $res : 2^{\mathcal{L}^*} \times S \times O \times R \to \{allow, deny\}$ *is a conflict resolution operator: if* $allow(s, o, r)$ *is part of its input but* $deny(s, o, r)$ *is not then it returns* $allow$*, and vice versa.*

In this definition, the guard for an ordering $B \prec C$ is formalized as a function that dictates which subset of access control requests B is permitted to make. In practice that function is expressed in a logical policy language. For example, the guard might itself be a TBA (structured) policy, thereby deciding which requests are pertinent for B based on the tags for the subjects and objects.

Example 3 (Guards). Suppose the President wanted to scope the policy of his Army Chief of Staff so that it could only make authorization decisions about the objects the Army is primarily responsible for. If all such objects are tagged with $army$, the guard on the ordering $Army \prec Pres$ might be expressed as

$$allow(S, O, R) :- army \in tag(O). \qquad \blacksquare$$

Another noteworthy part of our structured policy definition is the meta-language N. N represents a logical language in which the access control decisions of all the basic policies can be combined. Formally, the process of combining access control decisions is achieved with N's entailment relation: given the decisions made by (possibly) different policies, compute all the implications of those decisions. Technically, this requires the premise sets of N to include all possible combinations of access control decisions from the individual policy languages—a constraint included in the definition.

The final component of a structured policy that warrants discussion is the conflict resolution operator res. res is given the implications of all the appropriate policy decisions and must choose whether to allow or deny. For unambiguous cases (where either $allow$ or $deny$ is present but not both), its behavior is fixed, but for ambiguous cases where its input includes both allow and deny, it is free to make either decision. Because the language of access control decisions is unconstrained, those decisions can record a plethora of information important for conflict resolution, *e.g.*, the source of the decision or its proof. Thus, the conflict resolution operator may be given not only a series of $allow$ and $deny$ statements but also statements that justify each $allow$ and $deny$. For example, for conflict resolution that utilizes proofs, the individual policy decisions might always include a sentence of the form $explanation(allow/deny(s, o, r), proof)$. Thus, TBA makes no commitment to a particular conflict resolution operator or even the information upon which conflicts are resolved, as these issues have been studied heavily in the literature, *e.g.*, [1, 3–6, 11, 14, 17, 24, 30].

The formal semantics of a structured policy is defined in terms of the decision a given basic policy p makes about a given access control request $\langle s, o, r \rangle$. If p either allows or denies the request, p's decision stands; otherwise, p's decision is the combination of its partial decisions together with the union of the decisions made by the

policies to which p delegated (*i.e.*, the policies immediately less than p in the policy ordering). A structured policy allows a request $\langle s, o, r\rangle$ if the conflict resolution operator when applied to the union of the decisions made by the maximal policies in the ordering returns $allow$; otherwise, the structured policy denies the request.

Furthermore, because the definition for a structured policy allows basic policies to be written in different logical languages (including *e.g.*, linear logic [9], first-order logic [16], and ASP [4]), the formal semantics correctly addresses heterogenous collections of basic policies, using $\models_p$ to denote the entailment relation for policy p.

Definition 4 (Structured Policy Semantics). *Consider a structured policy* $\langle P, \prec, G, \langle \mathcal{P}^*, \mathcal{L}^*, \models^*\rangle, res\rangle$*, tag function* tag*, ontology* Γ*, and an access control request* $\langle s, o, r\rangle$*. First, for all* $x \in S \cup O$*,* $tag(x) \cup \Gamma$ *is consistent. Second we define the point semantics of policy* $p \in P$ *on* $\langle s, o, r\rangle$*, written* $Point[p, s, o, r]$*, which is an element of* $\mathcal{P}^*$*. Let* $S = \{\phi \mid p, Cn^{\Gamma}(tag) \models_p \phi\}$.

1. *If* S *includes* $allow(s, o, r)$ *and/or* $deny(s, o, r)$ *then* $Point[p, s, o, r] = S$.
2. *Otherwise,* $Point[p, s, o, r] = S \cup \bigcup_{\substack{q \prec p \text{ and} \\ guard_{q \prec p}(s, o, r)}} Point[q, s, o, r]$.

Finally we define the structured policy semantics.

$$auth_{\mathsf{TBA}}(s, o, r) \text{ iff } res\left(Cn^*\left(\bigcup_{p \mid \nexists q. p \prec q} Point[p, s, o, r]\right)\right) = allow$$

Admittedly the formal definitions for a structured policy are not so simple; however, once the logical policy languages are chosen, explaining to policy writers how to use a structured policy is especially simple: write basic policies to make access control decisions and adjust the partial order and its guards to delegate those decisions.

Example 4 (Simple ordering). Suppose the U.S. President wanted to make decisions about information the general public was allowed to have, *e.g.*, the list of visitors who met with the President. He writes a policy A that allows everyone access to the appropriate objects and neither allows nor denies any of the remaining requests. The President then delegates the remaining decision to his chiefs of staff for the Army, Navy, and Air Force, who write policies B, C, and D, respectively. The partial order is then $B \prec A$, $C \prec A$, $D \prec A$. The list of Presidential visitors are always allowed because policy A allows them and does not deny them. Any decision not made by policy A is then delegated to policies B, C, and D. If for some request B allows but C denies, then the conflict resolution operator resolves the ambiguity. ■

Example 5 (Disjunctive decisions). Suppose an upper-level manager wants to ensure that every employee is either given access to object o_1 or object o_2 but not both. Moreover, she wants to delegate the choice to the low-level managers in the company. She can author a (first-order logic) policy, A, that says $\forall s.(allow(s, o_1, read) \vee allow(s, o_2, read))$ and $\forall s.(deny(s, o_1, read) \vee deny(s, o_2, read))$. Then if the low-level manager policies are B_1,...,B_n, the upper-level manager ensures that $B_i \prec A$

with appropriate guards for $i \in \{1, \ldots, n\}$. Each policy B_i can then choose which of the objects to grant for each employee. Furthermore, if one of the low-level managers writes a policy that grants access to both objects or to neither, there will be a conflict, and the conflict resolution operator can choose to enforce A's policy by arbitrarily choosing between o_1 and o_2. ■

4 Algorithms

In this section we introduce an algorithm that use an ontology-aided, structured TBA policy to either allow or deny a given request. The algorithm was designed for a meta-language that is a simple fragment of first-order logic: all premise sets are ground atoms, the queries are simply $allow(s, o, r)$ and $deny(s, o, r)$, and the entailment relation is standard first-order entailment. The algorithm is sound and complete as long as the proof systems for basic policies are sound and complete (along with a few other simple conditions), and except for tag expansion it runs in polynomial time under similar conditions. Moreover, we identify a class of ontologies for which tag expansion runs in polynomial time.

The algorithm begins by expanding the subject and object tags to include all their consequences as prescribed by the ontology. It then recursively walks the partial order of policies to compute the point semantics of each policy, while memoizing the results. The point semantics computation requires computing the consequences of the basic theory, evaluating guards on lesser policies, and possibly recursing. Then the algorithm computes the union of the point semantics of the maximal policies and gives the result to the conflict resolution operator. Algorithm 1 and Algorithm 2 give the pseudo-code.

Algorithm 1 TBA (request $\langle s, o, r\rangle$, tags tag, Horn ontology Γ, policy $\langle P, \prec, G, res\rangle$)

Returns: **true** if $\langle s, o, r\rangle$ is allowed and **false** otherwise

1: $point(p) := \bot$ for all policies $p \in P$
2: $decisions := \emptyset$
3: $expandedtag := Cn^{\Gamma}(tag)$
4: **for all** $p \in P$ where p is maximal in $\prec$ **do**
5: $\quad point := \text{UPDATEPOINT}(point, p, expandedtag)$
6: $\quad decisions := decisions \cup point(p, expandedtag)$
7: **return** $res(decisions) == allow$

Theorem 1. *Algorithm 1 is sound and complete if (i) for all basic policies p, the proof system for p is sound and complete and (ii) res when applied to atoms X and s, o, r ensures $res(X, s, o, r) = res(X \cap \{allow(s, o, r), deny(s, o, r)\}, s, o, r)$.*

Proof. Suppose Algorithm 1 is invoked on $\langle s, o, r\rangle$, tag, $Horn(\Gamma)$, and $\langle P, \prec, G, res\rangle$.

(Soundness and Completeness) Algorithm 1 allows the request $\langle s, o, r\rangle$ if and only if res returns $allow$ when applied to the union of the computed point semantics of the maximal policies in $\prec$ (restricted to $allow(s, o, r)$ and $deny(s, o, r)$) using the tag

Algorithm 2 UPDATEPOINT(reference $\&point$, policy p, tag function tag)

Assumes: Parameters to TBA are accessible

Returns: Point semantics for p on $\langle s, o, r\rangle$, recursively memoizing results in $point$

1: **if** $point(p) = \bot$ **then**
2: $V := \{allow(s,o,r) \mid p, tag \vdash_p allow(s,o,r)\} \cup$
$\{deny(s,o,r) \mid p, tag \vdash_p deny(s,o,r)\}$.
3: **if** $V \neq \emptyset$ **then**
4: $point(p) := V$
5: **else**
6: $decisions = \emptyset$
7: **for all** $q \in P$ where $q \prec p$ and $guard_{q \prec p}(s,o,r) = true$ **do**
8: $decisions := decisions \cup$ UPDATEPOINT$(point, q, tag)$
9: $point(p) := decisions$
10: **return** $point(p)$

function $Cn^{Horn(\Gamma)}(tag)$. In contrast, the semantics makes the same decision but gives res as input the atoms of all the consequences of the union of the maximal point semantics and uses the tag function $Cn^{\Gamma}(tag)$. First note that Lemma (1) guarantees that $Cn^{Horn(\Gamma)}(tag) = Cn^{\Gamma}(tag)$ since every tag set is required to be consistent with Γ. Second, because of the restriction on res, the non-$\langle s, o, r\rangle$ decisions are irrelevant to res; thus, it suffices to show that if the maximal policies in $\prec$ are $\{p_1, \ldots, p_k\}$, then $point(p_1) \cup \cdots \cup point(p_k) = Cn^*(Point[p_1, s, o, r] \cup \cdots \cup Point[p_k, s, o, r]) \cap \{allow(s,o,r), deny(s,o,r)\}$. To prove this equality, we first claim that for every i, $point(p_i) = Point[p_i, s, o, r] \cap \{allow(s,o,r), deny(s,o,r)\}$. Then we proceed as follows.

$point(p_1) \cup \cdots \cup point(p_k)$
by our claim
$= Point[p_1, s, o, r] \cap \{allow(s,o,r), deny(s,o,r)\} \cup \cdots \cup Point[p_1, s, o, r] \cap \{allow(s,o,r), deny(s,o,r)\}$
by distributivity
$= (Point[p_1, s, o, r] \cup \cdots \cup Point[p_1, s, o, r]) \cap \{allow(s,o,r), deny(s,o,r)\}$
since the meta-language is first-order logic where $\mathcal{L}^*$ and therefore Point is atomic
$= Cn^*(Point[p_1, s, o, r] \cup \cdots \cup Point[p_1, s, o, r]) \cap \{allow(s,o,r), deny(s,o,r)\}$

Thus, Algorithm 1 is sound and complete as long as the point semantics are computed as claimed above.

(Point Semantics) The computed point semantics for each policy p, $point(p)$, is computed and memoized with respect to the given request $\langle s, o, r\rangle$ by UPDATEPOINT. Using induction on the ordering $\prec$ we prove that $point(p) = Point[p_i, s, o, r] \cap \{allow(s,o,r), deny(s,o,r)\}$. Base case is the minimal elements of $\prec$. Inductive case assumes the hypothesis for all policies less than a given policy. The two cases only differ in that in the base case the set of policies less than the given policy is known to be empty, whereas in the inductive case there may be 0 or more lesser policies. So consider an invocation of UPDATEPOINT for policy p. If the point semantics for p has already been computed, UPDATEPOINT returns that value: $point(p)$. Otherwise, it computes a set of ground atoms V from p using $\vdash_p$ and the expanded tag function, which by the soundness and completeness of $\vdash_p$ ensures $V = Cn(p) \cap \{allow(s,o,r), deny(s,o,r)\}$.

In contrast, the semantics compute the set $S = Cn(p)$, which differs from V because S may contain allow/deny atoms for other access control requests. But clearly $V = S \cap \{allow(s, o, r), deny(s, o, r)\}$. If V is non-empty, $point(p) = V$, making $point(p) = Point[p, s, o, r] \cap \{allow(s, o, r), deny(s, o, r)\}$. Otherwise, UPDATE-POINT computes $point(q)$ for all $q \prec p$ where $guard_{q \prec p}(s, o, r)$ is true and sets $point(p)$ to the union of the results plus V, *i.e.*, $point(p) = V \cup \bigcup_{q \prec p} point(q)$. We proceed as follows.

$point(p)$
$= V \cup \bigcup_{q \prec p} point(q)$
by V's equality above and the inductive hypothesis
$= S \cap \{allow(s, o, r), deny(s, o, r)\} \cup \bigcup_{q \prec p} Point[q, s, o, r] \cap \{allow(s, o, r), deny(s, o, r)\}$
by distributivity
$= (S \cup \bigcup_{q \prec p} Point[q, s, o, r]) \cap \{allow(s, o, r), deny(s, o, r)\}$
by semantics
$= Point[p, s, o, r] \cap \{allow(s, o, r), deny(s, o, r)\}$

This completes the inductive step and the inductive proof. ■

Expanding the tags for a given subject or object is coNP-complete because it amounts to computing entailment in propositional logic. Moving tag expansion offline, *e.g.*, each time a subject or object is entered into the system, would ensure Algorithm 1 always runs in polynomial time. The drawback of offline tag expansion is that it requires storing the consequences of all the tags for all the subjects and objects and updating all those consequences each time the ontology changes. Online tag expansion, on the other hand, would enable that computation to be performed for only the relevant subjects and objects; moreover, the results would not need to be stored.

For situations where online tag expansion is deemed necessary, we identify a special class of ontologies that can be compiled offline to make online tag expansion run in polynomial time. Such ontologies have the property that their Horn consequences are polynomial in their size. Since the Horn consequences are sufficient for performing tag expansion and that the algorithm for Horn tag expansion runs in polynomial time, computing the Horn consequences of the ontology offline enables polynomial online tag expansion. We say that such ontologies have a *small Horn theory*.

Definition 5 (Small Horn Theory). *The Horn consequences of an ontology Γ, denoted $Horn(\Gamma)$, is the set of all statements of the form $p_1 \Leftarrow p_2 \wedge \cdots \wedge p_n$ that are entailed by Γ, where p_i is a proposition for all i. Γ has a small Horn theory if the size of $Horn(\Gamma)$ is polynomial in the size of Γ.*

To utilize these results, we invoke Algorithm 1 with $Horn(\Gamma)$ instead of Γ and ensure that the Cn^{Γ} operator implements the polynomial-time version of Horn entailment. If Γ has a small Horn theory, tag expansion runs in polynomial time in Γ; otherwise, $Horn(\Gamma)$ is exponential in Γ and tag expansion runs in exponential time in Γ.

Lemma 1. *Suppose Γ is an ontology, and G is a set of tags such that $\Gamma \cup G$ is consistent. If $\Gamma \cup G \models t$ then $Horn(\Gamma) \cup G \models t$. Moreover, the set of all t such that $Horn(\Gamma) \cup G \models t$ can be computed in polynomial time.*

Proof. First we show that the Horn consequences of a theory are sufficient for entailment. Suppose $\Gamma \cup G \models t$; then since $\Gamma \cup G$ is consistent by the generative completeness of resolution there is a resolution proof ending in the singleton clause $\{t\}$. We demonstrate how to construct a proof of the singleton clause $\{t\}$ from $Horn(\Gamma) \cup G$. First se the associativity and commutativity of resolution to reorder the original proof so that all the resolutions with elements from G occur at the end. Second, starting at $\{t\}$, walk backwards up the proof. If there is no ancestor of $\{t\}$ that belongs to the resolution closure, then the proof of $\{t\}$ depends only on G, and hence there is a proof of $\{t\}$ from G and therefore from $Horn(\Gamma) \cup G$. So suppose there is an ancestor clause C of $\{t\}$ that belongs to the resolution closure of Γ. The remainder of the proof resolves away all of the literals in C except for t. Since all of those resolution steps resolve with elements of G, which are all positive, all of the literals of C are negative except for t. Thus C is a Horn clause, and by construction there is a proof of $\{t\}$ from C and G; ergo, there is a proof of $\{t\}$ from $Horn(\Gamma) \cup G$.

Next we recall the proof that computing all the atomic consequences of a propositional Horn theory requires polynomial time. This ensures that the consequences of $Horn(\Gamma) \cup G$ is computable in polynomial-time because $Horn(\Gamma) \cup G$ is a propositional Horn theory. Here we assume the Horn theory is written in rule form. The algorithm is a simple variation of the CFG marking algorithm found in Sipser [26]. Begin by finding the propositions occurring in the head of a rule with an empty body, and place a mark on all occurrences of those propositions throughout the rule set. Next find all those propositions occurring in the head of rule where all the propositions in the body have been marked, and mark all occurrences of those propositions in the rules; iterate until no new marks are made. The set of all propositions entailed is the set of all marked propositions. Each iteration of this algorithm runs in time linear in the size of the Horn rules, and the number of iterations is bounded from above by the number of propositions in the rules since each iteration must mark at least one proposition. Thus, the total runtime is linear in the size of the Horn theory. ■

Theorem 2. *Algorithm 1 runs in polynomial time if (i) the proof systems for all the basic policies run in polynomial time, (ii) the evaluation of $guard_{B \prec C}$ runs in polynomial time, (iii) the ontology has a small Horn theory, and (iv) the conflict resolution mechanism runs in polynomial time.*

Proof. Suppose Algorithm 1 is invoked on $\langle s, o, r \rangle$, tag, $Horn(\Gamma)$, and $\langle P, \prec, G, res \rangle$.

First since (i) Γ has a small Horn theory, $Horn(\Gamma)$ is guaranteed to be polynomial in the size of Γ, and (ii) the lemma above guarantees that computing the Horn consequences of a theory and a tag set is polynomial, the computation of the expansion of any single tag set V is polynomial in Γ and V. The algorithm expands the tag set for each of the subjects and objects, making the total expansion computation run in time polynomial in the size of Γ and tag; moreover, the expanded tag function is polynomial in the size of Γ and tag.

Second, the algorithm computes the point semantics for each policy $p \in P$ at most once; thus, the point semantics computation in the worst case amounts to walking the partial order and for each policy, (i) computing the consequences of that policy, (ii), computing several guards, (iii) unioning the point semantics of child policies. Step (i) is

assumed to run in polynomial time in its inputs. Since the expanded tag is polynomial in the size of the original tag and Γ, and the policy p is unchanged, the inputs to $\vdash_p$ are polynomial in the size of p, tag, and Γ; thus, computing the consequences requires polynomial time and therefore produces a polynomial-sized result. Step (ii) is assumed to run in polynomial time for each guard and because of memoization, each guard is computed at most once, making this step run in time polynomial in the guard computation and the size of $\prec$. Step (iii) is the union of at most $|P|$ polynomial-sized results, which produces a polynomial-sized result in polynomial time. Thus the point semantics computation runs in time polynomial in the size of P, $\prec$, Γ, G, tag.

Finally, the conflict resolution mechanism is applied to the result of the point semantics computation. Since it is assumed to run in polynomial time, and its inputs are polynomial in Algorithm 1's inputs, the entire algorithm runs in polynomial time. ■

5 Evaluation

In this section, we evaluate the utility of TBA by exploring its expressive power. We first demonstrate that various incarnations of TBA can be used to express a range of common policy idioms. We then use the formal reduction framework developed by Tripunitara and Li [27] to demonstrate that TBA is more expressive than several representative access control schemes from the literature. Our reductions show that TBA can provide the same features as these schemes while also preserving the compositional security analysis properties of these systems.

5.1 Representing Common Policy Idioms

Below we enumerate a list of well-known authorization policy idioms and show that each can be represented using some tag-based authorization system. When using tag-based authorization to represent each of these idioms, we use DATALOG as the underlying policy language. In doing so, we assume that every request not explicitly allowed is denied.

Access matrix. The access matrix uses a function $matrix : S \times O \to 2^R$ to store the rights that each subject has over every object. An access is permitted under the following condition: $auth_{mat}(s, o, r)$ iff $r \in matrix(s, o)$. To implement this scheme using tag-based authorization, there must be a unique tag for each document (*e.g.*, its inode number) and each user (*e.g.*, her uid). The policy consists of a series of simple statements such as the one below.

$$allow(S, O, read) :- user123 \in tag(S), doc789 \in tag(O)$$

Attribute-based access control. In attribute-based authorization systems, access decisions are made based on the attributes ascribed to a user by their organization. Basically, ABAC is TBA without object tags (or, more formally, where every object is tagged with the empty set). The following example allows any user to read $doc789$, provided that she is a member of the security group and has not been blacklisted.

$$allow(S, doc789, read) :- security \in tag(S), blacklist \notin tag(S)$$

Role-based access control. In RBAC systems, users are assigned to roles representing their job functions, and permissions are given to roles. Here we focus on RBAC1 as defined in [12], where the roles are arranged in a hierarchy, and a user is granted access when one of her roles is higher in the hierarchy than some role that is permitted access.

$$auth_{RBAC1}(s, o, r) \text{ iff } \exists g, g'.UR(s, g) \wedge g \geq g' \wedge PA(g', o, r))$$

To implement RBAC1 with tag-based authorization, the tag set is defined as $T = G \cup G \times R$, *i.e.*, the set of roles and the set of (role, right) tuples. Users are tagged with their roles, and documents are tagged with (role, right) tuples. The role hierarchy is axiomatized as an ontology Γ so that for every pair of roles such that $g \geq g'$, we have $\Gamma \models g \Rightarrow g'$. The following DATALOG policy implements RBAC1.

$$allow(S, O, R) :- G \in tag(S), \langle G, R \rangle \in tag(O)$$

Discretionary access control (Linux). In the Linux authorization model, each document has different rights for its owner, group, and the world, which we represent with the function $rights$. $rights$ maps each object to three sets of rights $O \rightarrow 2^R \times 2^R \times 2^R$. The functions $user$, $group$, and $world$, when applied to the rights of a document extract the rights that the user, the group, and the world have for that object access, respectively. Additionally each object is associated with a user owner and a group owner, represented by the functions $userown : O \rightarrow S$ and $groupown : O \rightarrow G$ (where G is the set of all groups). Each user is assigned some number of groups, represented by the function $groups : S \rightarrow 2^G$.

$$\begin{aligned} &auth_{linux}(s, o, r) \text{ iff} \\ &\quad (userown(o) = s \wedge r \in user(rights(d))) \vee \\ &\quad (groupown(o) \in groups(s) \wedge r \in group(rights(d)) \vee \\ &\quad r \in world(rights(d)) \end{aligned}$$

The Linux authorization policy gives a user access if (i) the user owns the document and the owner has access, (ii) the user belongs to the group that owns the document and the group has access, or (iii) the world has access. To implement the Linux scheme with TBA, the document tags T consist of the subjects S, groups G, and the rights tags $\{userread, groupread, worldread\}$. Permissions other than *read* can be handled in a similar manner. Each document is tagged with the user owner, the group owner, and a subset of the rights tags, and each user is tagged with the groups she belongs to. The Linux read policy is then given by the DATALOG fragment below.

$$\begin{aligned} &allow(U, D, read) :- U \in tag(D), userread \in tag(D) \\ &allow(U, D, read) :- G \in tag(D), groupread \in tag(D), G \in tag(U) \\ &allow(U, D, read) :- worldread \in tag(D) \end{aligned}$$

Mandatory (Lattice-Based) Access Control In an LBAC system, L is a set of classification/clearance levels (*e.g.*, Secret, TopSecret), and C is the set of compartments (*e.g.*, Nuclear, Submarine). The function $classification$ assigns each object a classification and a set of compartments: $O \rightarrow L \times 2^C$, and the function $clearance$ assigns each subject a clearance and set of compartments: $S \rightarrow L \times 2^C$. There is a total ordering $\leq$

on L, and $\sqsubseteq$ denotes the partial ordering on $L \times 2^C$: $(l_1, c_1) \sqsubseteq (l_2, c_2)$ if and only if $l_1 \leq l_2$ and $c_1 \subseteq c_2$. Users can read documents whose classifications are dominated by their clearance (no read up) and write documents whose classifications dominate their clearance (no write down):

$$auth_{LBAC}(u, d, read) \text{ iff } classification(d) \sqsubseteq clearance(u)$$
$$auth_{LBAC}(u, d, write) \text{ iff } classification(d) \sqsupseteq clearance(u)$$

To implement this policy idiom with tag-based authorization, the set of tags is the set of all compartments and security levels: $T = L \cup C$. Each subject and object is tagged with its level and all its compartments. Then the DATALOG (with negation) policy for the LBAC no-read-up idiom is given below, where compartment tags are identified by $comp$, level tags by $level$, and the total ordering on levels is represented by leq.

$$\begin{aligned}
&allow(S, O, read) :- allowlevel(S, O), allowcomp(S, O)\\
&allowlevel(S, O) :- C \in tag(S), level(C),\\
&\qquad\qquad E \in tag(O), level(E), leq(E, C)\\
&allowcomp(S, O) :- \neg somecompmissing(S, O)\\
&somecompmissing(S, O) :- C \in tag(O), comp(C), C \notin tag(S)
\end{aligned}$$

In the first rule, the first condition, $allowlevel$, ensures that the object's security level is less than the subject's level. The second condition, $allowcomp$, ensures that the object's compartments are a subset of the subject's compartments, which is implemented by ensuring it is not the case that one of the object's compartments fails to be one of the subject's compartments.

The RT Trust Management Language. RT [20] employs a form of role-based delegation that consists of the four types of rules shown in Table 1. Structured policies of TBA can express a certain fragment of RT-style delegation. Given a set of rules of types 1–3, where the delegation graph of those rules is acyclic, we can emulate those rules by constructing a structured TBA policy. (The delegation graph consists of one node per principal and an edge from A to B if A delegates to B.) The partial order $\prec$ includes $B \prec A$ if A delegates to B. Then by using a new right $activate$, we proceed as follows for each of the rule types.

1. • Add $allow(userB, roleA.R, activate)$ to policy A
2. • Add $allow(S, roleA.R, activate) :- allow(S, roleB.R_1, activate)$ to A
   • Add $\langle S, roleB.R_1, activate\rangle$ to $guard_{B \prec A}$ for all S
3. • Add $allow(S, roleA.R, activate) :- allow(S, roleB.R_1, activate)$, $allow(S, roleC.R_2, activate)$ to policy A
   • Add $\langle S, roleB.R_1, activate\rangle$ to $guard_{B \prec A}$ for all S
   • Add $\langle S, roleC.R_2, activate\rangle$ to $guard_{C \prec A}$ for all S

Rules of type (4) are not expressible since they cause the delegation graph to be dependent on the contents of basic policies. Even if it were reasonable to require that dynamic delegation graph to be acyclic, emulating type (4) rules would require changing $\prec$ each time a basic policy changed.

Type	Rule	Description
1	$A.R \leftarrow B$	User B is a member of the role R defined by user A
2	$A.R \leftarrow B.R_1$	A's role R contains all members of B's role R_1
3	$A.R \leftarrow B.R_1 \cap B.R_2$	A's role R contains all users who are members of both B's role R_1 and C's role R_2
4	$A.R \leftarrow A.R_1.R_2$	A's role R contains all users who are members of X's role R_2 for some X in A's role R_1

Table 1. The four types of RT rules.

5.2 Formal Expressive Power Analysis

The preceding section demonstrates that TBA is capable of encoding many common policy idioms, but says nothing about whether these encodings have the same safety analysis properties of common implementations of these idioms. In [27], Tripunitara and Li introduce a framework for comparing the expressiveness of access control systems that views an access control system as a state transition system and performs comparisons using a type of bisimulation and a generalized definition of safety. The crux of their framework relies on demonstrating the existence or non-existence of a *state-matching reduction* between two systems. Intuitively, a state-matching reduction from A to B is a mapping from the states of A to the states of B so that an external observer affecting access control changes and making queries can not distinguish whether she is using A or B, and further implies that B maintains all safety analysis properties of A. As a result, state-matching reductions are a way of analyzing the relative expressive power of two systems.

We now present theorems comparing TBA to a number of well-known systems in terms of state-matching reductions. Proofs of these theorems are available in the appendices of this paper. We first show that TBA is at least as expressive as a common discretionary access control scheme (SDCO), a common role-based access control scheme (ARBAC97), and a common mandatory access control scheme (the Bell-La Padula model).

Theorem 3. *There exists a state-matching reduction from SDCO to TBA.*

Theorem 4. *There exists a state-matching reduction from ARBAC97 to TBA.*

Theorem 5. *There exists a state-matching reduction from BLP to TBA.*

The preceding three theorems are proven by construction; *i.e.*, by first giving a mapping to TBA from each access control scheme, and then proving that the mapping satisfies the requirements of a state-matching reduction. Since a state-matching reduction from A to B proves that B is at least as expressive as A, the above results show that TBA is at least as expressive as SDCO, ARBAC97, and BLP. The next results ensure that none of these schemes are as expressive as TBA.

Theorem 6. *There exists no state-matching reduction from TBA to SDCO.*

Theorem 7. *There exists no state-matching reduction from TBA to ARBAC97.*

Theorem 8. *There exists no state-matching reduction from TBA to BLP.*

Non-existence is proven by contradiction; *i.e.*, by first assuming the existence of a state-matching reduction from TBA to another system, and then providing a state and transition rule for TBA under which it is impossible for any mapping to the other system to satisfy the structural requirements of a state-matching reduction. When a state-matching reduction from A to B is accompanied by the nonexistence of a state-matching reduction in the reverse direction, it proves that B is *strictly* more expressive than A. As a result, we have shown that TBA is strictly more expressive than all of SDCO, ARBAC97, and BLP.

6 Related Work

TBA has been studied informally in [23, 29], though that work allows tags on subjects but not objects. Section 5 compares TBA to several well-known authorization paradigms. We do not survey related work on tag ontologies, which have been studied extensively by the Semantic Web community, but as evidence of viability simply point to two organizations employing ontologies to handle large terminologies: the U.S. National Cancer Institute (NCI Thesaurus `http://www.cancer.gov/cancertopics/terminologyresources`) and the U.S. National Library of Medicine (SNOMED-CT `http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html`). In this section, we discuss work related to structured policies, conflict resolution, and delegation.

Policy Structure. Structured policies can be seen as combining two operations on policies: the "override" operator in [8] (also called "exceptions" in [5]) and the "scoping" operator in [8, 24]. Our choice to employ these two policy combination operators instead of a richer framework [8, 10] was driven by our desire for a conceptually simple and therefore highly usable framework; these two operators seemed to be the minimal necessary to support delegation.

Conflict resolution. Conflict resolution is important in the context of structured policies, where conflicts must be resolved within basic policies as well as across policies. We make no commitment to a particular scheme but provide a framework for implementing other proposals in the literature. Our framework is based on the premise of a fixed global operator such as [4, 11, 24], though user-settable conflict resolution schemes for each policy such as in [3, 14, 30] can be achieved by building them into the entailment relations for the individual policies.

Trust Management and delegation. Trust management [2, 7, 15, 18–20, 25, 31, 32] is concerned with distributed authorization and therefore focuses extensively on delegation [2, 13, 21]. TBA's delegation functionality was designed for simplicity, and as shown in Section 5 is less powerful than RT's delegation primitives. This decision was made to improve usability, with an acknowledged decrease of flexibility and expressiveness.

7 Conclusion

Logical access control systems are attractive for their power and flexibility, while extensional access control systems are known for their simplicity and the ease with which

relatively untrained users can contribute. Tag-based authorization combines these qualities into a single system. Subjects and objects are assigned tags, and access is decided by a policy over those tags. TBA is powerful and flexible through its logical policy, and achieves nearly the expressiveness of existing logical access control systems. At the same time, it is simple to describe and allows relatively untrained personnel to assign tags to objects, more fully utilizing a diverse workforce. Tag ontologies can further simplify both object tagging and policy writing. In addition, our approach to delegation externalizes the mechanism through which policies are combined, enabling different sub-policies to be written in distinct languages.

To evaluate TBA, we explored its ability to express common access control policy idioms and its formal expressive power. We show via simple example instantiations that TBA is capable of expressing the access matrix, attribute-based, role-based, discretionary and mandatory access control paradigms. Then, by utilizing the reduction framework of Tripunitara and Li, we show that TBA is strictly more expressive than specific, common implementations of these paradigms, namely SDCO (a common access matrix system), ARBAC97 (a common role-based system), and BLP (the U.S. military's extended mandatory system). Thus, TBA is not only much more intuitive to describe and use than other current logical authorization systems, but also strictly more expressive than current extensional access control systems, making it a true hybrid of these two types of access control metaphors.

References

1. Adam Barth, John C. Mitchell, and Justin Rosenstein. Conflict and combination in privacy policy languages. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 2004.
2. Moritz Y. Becker, Cedric Y. Fournet, and Andrew D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 2009.
3. Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1):71–127, 2003.
4. Elisa Bertino, Elena Ferrari, Francesco Buccafurri, and Pasquale Rullo. A logical framework for reasoning on data access control policies. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1999.
5. Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems*, 17(2):101–140, 1999.
6. Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.
7. Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–173, 1996.
8. Piero A. Bonatti, Sabrina D. di Vimercati, and Pierangela Samarati. A modular approach to composing access control policies. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 164–173, 2000.
9. Kevin D. Bowers, Lujo Bauer, Deepak Garg, Frank Pfenning, and Michael K. Reiter. Consumable credentials in logic-based access-control systems. In *Proceedings of the Network and Distributed System Security Symposium*, pages 143–157, 2007.

10. Glenn Bruns and Michael Huth. Access-control policies via belnap logic: Effective and efficient composition and analysis. In *Proceedings of the IEEE Computer Security Foundations Symposium*, 2008.
11. Laurence Cholvy and Frederic Cuppens. Analyzing consistency of security policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1997.
12. Jason Crampton. Understanding and developing role-based administrative models. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 158–167, 2005.
13. Jason Crampton, George Loizou, and Greg Oshea. A logic of access control. *The Computer Journal*, 44(1):137–149, 2001.
14. Frederic Cuppens, Laurence Cholvy, Claire Saurel, and Jerome Carrere. Merging security policies: analysis of a practical example. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1998.
15. John DeTreville. Binder, a logic-based security language. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 105–113, May 2002.
16. Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *Proceedings of the IEEE Computer Security Foundations Symposium*, 2003.
17. Sushil Jajodia, Pierangela Samarati, and V S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 31–42, 1997.
18. Ninghui Li, Benjamin Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. In *Proceedings of the ACM Transactions on Information and System Security*, pages 128–171, 2003.
19. Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Symposium on Practical Aspects of Declarative Languages*, 2003.
20. Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.
21. Ninghui Li and Mahesh V. Tripunitara. On safety in discretionary access control. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 96–109, 2005.
22. OWL web ontology language: Semantics and abstract syntax. `http://www.w3.org/TR/owl-semantics/`, 2004.
23. Maryam Najafian Razavi and Lee Iverson. Supporting selective information sharing with people-tagging. In *CHI Extended Abstracts*, pages 3423–3428, 2008.
24. Carlos Ribeiro, Andre Zuquete, Paulo Ferreira, and Paulo Guedes. SPL: An access control language for security policies with complex constraints. In *Proceedings of the Network and Distributed System Security Symposium*, 2001.
25. Ronald L. Rivest and Butler Lampson. Sdsi - a simple distributed security infrastructure. Technical report, Massachusetts Institute of Technology, 1996.
26. Michael Sipser. *Introduction to the Theory of Computation*. Brooks Cole, 1996.
27. Mahesh V. Tripunitara and Ninghui Li. A theory for comparing the expressive power of access control models. *Journal of Computer Security*, 15(2):231–272, 2007.
28. U.S. Air Force Scientific Advisory Board. Networking to enable coalition operations. Technical report, MITRE Corporation, 2004.
29. Qihua Wang, Hongxia Jin, and Ninghui Li. Usable access control in collaborative environments: Authorization based on people-tagging. In *ESORICS*, pages 268–284, 2009.
30. Duminda Wijesekera and Sushil Jajodia. Policy algebras for access control - the predicate case. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 171–180, 2001.

31. Marianne Winslett, Charles C. Zhang, and Piero A. Bonatti. Peeraccess: A logic for distributed authorization. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 168–179, 2005.
32. Charles C. Zhang and Marianne Winslett. Multitrust: An authorization framework with customizable distributed proof construction. In *Proceedings of the Joint Workshop on Foundations of Computer Security, Automated Reasoning for Security Protocol Analysis, and Issues in the Theory of Security*, 2008.

A An alternate formalization of tag-based authorization

In this section, we formalize TBA within the definition of access control scheme proposed by Tripunitara and Li [27].

In Tripunitara and Li's definition, an access control scheme is a state-transition system $\langle \Gamma, \Psi, Q, \vdash \rangle$, where:

- Γ is a set of states. Each state contains all the information needed to make an access control decision at any given moment.
- Ψ is a state-transition rule that describes how the system changes state.
- Q is a set of queries. Each query is answered by $true$ or $false$.
- $\vdash$ is the entailment relation that determines whether a given query is true or false in a given state.

In TBA, we assume the existence of:

- T, the set of possible tags.
- I, the set of access rights.
- L, the language used to define the policy.

These components are not defined as part of the state, as they do not change.

TBA is then defined as the state-transition system $\langle \Gamma^T, \Psi^T, Q^T, \vdash^T \rangle$. Each TBA state $\gamma^T \in \Gamma^T$ is defined by $\langle S, O, tag, P \rangle$, where:

- S is the set of subjects.
- O is the set of objects.
- $tag \subseteq (S \cup O) \times T$ is the tag relation, and contains the pair $\langle s, t \rangle$ for each subject s that has tag t and the pair $\langle o, t \rangle$ for each object o that has tag t.
- P is a set of policy sentences, written in language L.

Ψ^T is defined using the following commands.

```
command  create_object(s, o)
    O = O ∪ {o}
    tag = tag ∪ {⟨o, o.id⟩}

command  destroy_object(s, o)
    if(s has delete for o)
        O = O − {o}
        tag = tag − {⟨o, o.id⟩}

command  create_subject(s1, s2)
    if(s1 can create subjects)
        S = S ∪ {s2}
        tag = tag ∪ {⟨s2, s2.id⟩}

command  destroy_subject(s1, s2)
    if(s1 can delete subject s2)
        S = S − {s2}
        tag = tag − {⟨s2, s2.id⟩}

command  assign_tags(s, x, T1)
```

```
    if(s can edit tags for x)
        for each t ∈ T1
            tag = tag ∪ {⟨x, t⟩}

command  revoke_tags(s, x, T1)
    if (u can edit tags for x)
        for each t ∈ T1
            tag = tag − {⟨x, t⟩}

command  transform_policy(s, P+, P−)
    if(s can change P)
        P = P − P− ∪ P+
```

Q^T includes all queries of the following forms: (1) "Does subject s exist?", (2) "Does subject s have tag t?", (3) "Does object o have tag t?", (4) "Is policy sentence p in the policy?", and (5) "Does subject s have access i to object o?". $\vdash^T$ is defined as follows for queries of each of the forms above: (1) $true$ if and only if $s \in S$, (2) $true$ if and only if $\langle s, t\rangle \in tag$, (3) $true$ if and only if $\langle o, t\rangle \in tag$, (4) $true$ if and only if $p \in P$, and (5) $true$ if and only if $\exists T_1 \subseteq T, T_2 \subseteq T : \forall t_1 \in T_1, \langle o, t_1\rangle \in tag \wedge \forall t_2 \in T_2, \langle s, t_2\rangle \in tag \wedge \exists p \in P$ that grants subjects with tag set T_2 access i to objects with tag set T_1 under language L.

B Comparison to SDCO

In this section, we formalize a common access matrix-based access control scheme, strict discretionary access control with change of ownership (SDCO), and show that TBA is strictly more expressive than SDCO.

B.1 Formalization of SDCO

In SDCO, we assume the existence of I, the set of access rights, including own. SDCO is then defined as the state-transition system $\left\langle \Gamma^S, \Psi^S, Q^S, \vdash^S \right\rangle$. Each SDCO state $\gamma^S \in \Gamma^S$ is defined by $\langle S, O, M\rangle$, where:

- S is the set of subjects.
- O is the set of objects.
- $M : S \times O \to 2^I$ is the access matrix.

Ψ^S is defined using the following commands.

```
command  create_object(s, o)
    O = O ∪ {o}
    M[s, o] = own

command  destroy_object(s, o)
    if own ∈ M[s, o]
        O = O − {o}

command  grant_own(s, s′, o)
    if own ∈ M[s, o]
        M[s′, o] = M[s′, o] ∪ {own}
        M[s, o] = M[s, o] − {own}
```

And, for each $i \in (I - own)$:

```
command  grant_i(s, s', o)
    if own ∈ M[s, o]
        M[s', o] = M[s', o] ∪ {i}

command  revoke_i(s, s', o)
    if own ∈ M[s, o]
        M[s', o] = M[s', o] − {i}
```

Q^S includes all queries of the form "Does subject s have access i to object o?". $\vdash^S$ is defined as $true$ if and only if $i \in M[s, o]$.

B.2 Reduction from SDCO

Theorem 9. *There exists a state-matching reduction from SDCO to TBA.*

Proof. By construction. Presented is a mapping, and proof that the mapping satisfies the two properties for it to be a state-matching reduction by Tripunitara and Li's definition 7.

The mapping, σ, needs to be able to map every $\langle \gamma, \psi \rangle$ in SDCO to $\sigma(\langle \gamma, \psi \rangle) = \langle \gamma^T, \psi^T \rangle$ in TBA, as well as every q in SDCO to $\sigma(q) = q^T$ in TBA.

$$
\begin{aligned}
\text{Let } \sigma(\gamma) = \gamma^T = &\langle S_\gamma, O_\gamma, tag_\gamma, P_\gamma \rangle \text{ where} \\
S_\gamma = &S \cup \{sim\_admin\} \\
O_\gamma = &O \\
tag_\gamma = &\{\forall s \in S : \langle s, s.id \rangle\} \cup \{\forall o \in O : \langle o, o.id \rangle\} \\
P_\gamma = &\{\forall s \in S, o \in O, \forall i \in M[s, o] : \text{“}s.id : i : o.id\text{”}\}
\end{aligned}
$$

Here, L_γ is the set of sentences of the form "$t_1 : i : t_2$" where $t_1, t_2 \in T$ and $i \in I$. A policy P is consistent only if $\forall o \in O \exists s \in S$: "$s.id : own : o.id$" $\in P \wedge t \in T \neq s.id \implies$ "$t : own : o.id$" $\notin P$. The inference procedure for L_γ is as follows. The sentence "$t_1 : i : t_2$" grants any subject with the tag t_1 the right i to objects with the tag t_2.

Since queries in SDCO have the same form as form-(4) queries in TBA, let $\sigma(q) = q^T = q$.

Let γ_0 be a start state in SDCO. Produce γ_0^T in TBA using σ. Given γ_k such that $\gamma_0 \overset{*}{\mapsto}_\psi \gamma_k$, we show that there exists γ_k^T such that $\gamma_0^T \overset{*}{\mapsto}_{\psi^T} \gamma_k^T$ where, for all q, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Consider the case where $\gamma_k = \gamma_0$, then let $\gamma_k^T = \gamma_0^T$. In $\gamma_0^T = \sigma(\gamma_0)$, s will be given right i over o only using the following sentence in P: "$s.id : i : o.id$". Such a line will be entered if and only if $i \in M[s, o]$, so for all q, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Next, consider some arbitrary γ_k reachable from γ_0. We construct γ_k^T that is reachable from γ_0^T and that answers every q^T the same way γ_k answers q, as follows. Consider each state-transition in the sequence $\gamma_0 \mapsto_\psi \gamma_1 \mapsto_\psi \ldots \mapsto_\psi \gamma_k$ in the SDCO system. If the state-transition in SDCO is the execution of `create_object(s, o)`, we execute `transform_policy(`sim_admin`, {`"$s.id : own : o.id$"`}, {})` followed by `create_object(s, o)`. If the state-transition in SDCO is the execution of `destroy_object(s, o)`, we execute `destroy_object(s, o)`, followed by `transform_policy(`sim_admin`, {}, {`$\forall t, i :$ "$t : i : o.id$"`})`. If the state-transition in SDCO is the execution of `grant_own(s, s', o)`, we execute `transform_policy(`sim_admin`, {`"$s'.id : own : o.id$"`}, {`"$s.id : own : o.id$"`})`.

If the state-transition in SDCO is the execution of `grant_i(s, s', o)`, then we execute `transform_policy(`sim_admin`, {`"$s'.id : i : o.id$"`}, {})`. If the state-transition in SDCO is the execution of `revoke_i(s, s', o)`, then we execute `transform_policy(`sim_admin`, {}, {`"$s'.id : i : o.id$"`})`.

Now, consider each possible query q. Since q is of the form "Does subject s have access i to object o?", q^T is also "Does subject s have access i to object o?". In this case, $\gamma_k \vdash q$ if and only if i has been granted to s by the owner of o. This is true if and only if we have added the policy sentence "$s.id : i : o.id$" to P. Thus, $\gamma_k \vdash q$ if and only if $\gamma_k^T \vdash q^T$.

Therefore, we've proven property (1) for state-matching reductions.

We prove that property (2) for a state-matching reduction is satisfied by our mapping also by construction. Let γ_0^T be the start-state in TBA corresponding to γ_0, the start-state in SDCO. Then, if γ_k^T is a state reachable from γ_0^T and q^T is a query in TBA whose corresponding query in SDCO is q, we construct γ_k, a state in SDCO reachable from γ_0 as follows. For each sentence in P of the form "$s.id : own : o.id$", we execute `create_object(s, o)`. Then, for each sentence in P of the form "$s'.id : i : o.id$" where $i \neq own$, we execute `grant_i(s, s', o)`, where s is the owner of o.

Since q is of the form "Does subject s have access i to object o?", q^T is also "Does subject s have access i to object o?", which means that $\gamma_k^T \vdash q^T$ if and only if "$s.id : i : o.id$" $\in P$. The condition that q^T is true is the only one in which we would have added the right i to $M[s, o]$, and therefore $\gamma_k \vdash q$ if and only if $\gamma_k^T \vdash q^T$.

Therefore, we've proven property (2) for state-matching reductions, and proven that our mapping σ is a state-matching reduction.

B.3 Reduction to SDCO

Theorem 10. *There exists no state-matching reduction from TBA to SDCO.*

Proof. By contradiction. Assume there exists a state-matching reduction from TBA to SDCO. In TBA, adopt as γ a state where $s \in S$, $o \in O$, $P = \{\}$, and L is as described in Theorem 9 except the inference procedure is augmented as follows. The sentence "$t_1 : i^* : t_2$", in addition to granting subjects with tag t_1 the right i^* over objects with tag t_2, also grants such subjects the right i' over the same objects, regardless of whether P also contains the sentence "$t_1 : i' : t_2$". Languages like this can be used to express a heirarchy of rights, for example the *execute* right carries with it automatic *read* right. Let $q_1 =$ "Does s have right i^* to o?" and let $q_2 =$ "Does s have right i' to o?".

Observe that $\gamma \vdash \neg q_1 \wedge \neg q_2$. Consider the state γ^S in SDCO that is equivalent to γ (if there does not exist one, the contradiction of the existence of a state-matching reduction is found). We know that $\gamma^S \vdash \neg q_1^S \wedge \neg q_2^S$. Observe that, given $i^* \neq own$, there exists $\tilde{\gamma}^S$ reachable from γ^S such that $\tilde{\gamma}^S \vdash q_1^S \wedge \neg q_2^S$ via the execution of `grant_`i^*. However, the only $\tilde{\gamma}$ such that $\tilde{\gamma} \vdash q_1$ is that in which $\exists t_1, t_2 \in T$: "$t_1 : i^* : t_2$" $\in P \wedge \langle s, t_1 \rangle, \langle o, t_2 \rangle \in tag$. Due to the inference procedure of L, such a $\tilde{\gamma} \vdash q_1 \wedge q_2$, leaving no such $\tilde{\gamma} \vdash q_1 \wedge \neg q_2$, meaning there is no $\tilde{\gamma}$ that is equivalent to $\tilde{\gamma}^S$. This contradicts property (2) for state-matching reductions, giving us the needed contradiction and proof of the non-existence of a state-matching reduction from TBA to SDCO.

C Comparison to ARBAC97

In this section, we formalize a common role-based access control scheme used in corporate environments, ARBAC97, and show that TBA is strictly more expressive than ARBAC97.

C.1 Formalization of ARBAC97

In ARBAC97, we assume the existence of the countably infinite sets $\mathcal{U}$ (users), $\mathcal{P}$ (permissions) and $\mathcal{R}$ (roles), which will not be part of the state. ARBAC97 is then defined as the state-transition system $\langle \Gamma^A, \Psi^A, Q^A, \vdash^A \rangle$. Each ARBAC97 state $\gamma^A \in \Gamma^A$ is defined by $\langle UA, PA, RH, AR \rangle$, where:

- $UA \subseteq \mathcal{U} \times \mathcal{R}$ contains the pair $\langle u, r \rangle$ for each user u that is assigned to role r.
- $PA \subseteq \mathcal{P} \times \mathcal{R}$ contains the pair $\langle p, r \rangle$ for each permission p that is assigned to the role r.
- RH is the role-hierarchy, and for $r_1, r_2, \in \mathcal{R}$, $r_1 \succeq r_2 \in RH$ means that all users that are members of r_1 are also conceptually members of r_2, and all permissions that are assigned to r_2 are authorized to users that are members of r_1.
- $AR \subset \mathcal{R}$ is a set of administrative roles. Our definition does not model changes to AR since it is only changed by a trusted administrator who must do the security analysis to ensure that the resulting state is acceptable.

A role range, ξ, is written as (r_1, r_2) where $r_1, r_2 \in \mathcal{R}$, and every role that satisfies $r_1 \succeq r \wedge r \succeq r_2 \wedge r \neq r_1 \wedge r \neq r_2$ is in the role range ξ. The set of all role ranges is designated as Ξ. Preconditions for commands in Ψ^A are based on sets $URA97$, $PRA97$, and $RRA97$, which define:

$$
\begin{aligned}
can\_assign &\subseteq AR \times CR \times \Xi \\
can\_revoke &\subseteq AR \times \Xi \\
can\_assignp &\subseteq AR \times CR \times \Xi \\
can\_revokep &\subseteq AR \times \Xi \\
can\_modify &\subseteq AR \times \Xi
\end{aligned}
$$

where CR is a set of prerequisite conditions, and a prerequisite condition is a propositional logic formula over non-administrative roles. Ψ^A, then, is defined using the following commands.

```
command  assignUser(a, u, r)
    if ∃ ⟨ar, c, ξ⟩ ∈ can_assign such that
    ⟨a, ar⟩ ∈ UA ∧ u satisfies c ∧ r ∈ ξ
        UA = UA ∪ {⟨u, r⟩}

command  revokeUser(a, u, r)
    if ∃ ⟨ar, ξ⟩ ∈ can_revoke such that
    ⟨a, ar⟩ ∈ UA ∧ r ∈ ξ
        UA = UA − {⟨u, r⟩}

command  assignPermission(a, p, r)
    if ∃ ⟨ar, c, ξ⟩ ∈ can_assignp such that
    ⟨a, ar⟩ ∈ UA ∧ p satisfies c ∧ r ∈ ξ
        PA = PA ∪ {⟨u, r⟩}

command  revokePermission(a, p, r)
    if ∃ ⟨ar, ξ⟩ ∈ can_revokep such that
    ⟨a, ar⟩ ∈ UA ∧ r ∈ ξ
        PA = PA − {⟨p, r⟩}

command  addToRange(a, ξ, r)
```

```
    if ∃ ⟨ar, ξ⟩ ∈ can_modify such that
    ⟨a, ar⟩ ∈ UA
        RH = RH ∪ {r1 ⪰ r, r ⪰ r2}
        where ξ = (r1, r2) ∧ r ≠ r1 ∧ r ≠ r2

command  removeFromRange(a, ξ, r)
    if ∃ ⟨ar, ξ⟩ ∈ can_modify such that
    ⟨a, ar⟩ ∈ UA
        RH = RH − {r1 ⪰ r, r ⪰ r2}
        where ξ = (r1, r2) ∧ r ≠ r1 ∧ r ≠ r2

command  addAsSenior(a, r, s)
    if ∃ ⟨ar, ξ⟩ ∈ can_modify such that
    ⟨a, ar⟩ ∈ UA ∧ r ∈ ξ ∧ s ∈ ξ
        RH = RH ∪ {r ⪰ s}

command  removeAsSenior(a, r, s)
    if ∃ ⟨ar, ξ⟩ ∈ can_modify such that
    ⟨a, ar⟩ ∈ UA ∧ r ∈ ξ ∧ s ∈ ξ
        RH = RH − {r ⪰ s}
```

Q^A includes all queries of the following forms: (1) "Does user u belong to role r?", (2) "Is role r authorized to have permission p?", (3) "Is role r_1 a senior of role r_2?", and (4) "Is user u authorized to have permission p?". $\vdash^A$ is defined as follows for queries of each of the forms above: (1) $true$ if and only if $\langle u, r\rangle \in UA$, (2) $true$ if and only if $\langle p, r\rangle \in PA$, (3) $true$ if and only if $r_1 \succeq r_2 \in RH$, and (4) $true$ if and only if $\exists r_1, r_2 \in \mathcal{R} : \langle u, r_1\rangle \in UA \wedge \langle p, r_2\rangle \in PA \wedge r_1 \succeq r_2 \in RH$.

C.2 Reduction from ARBAC97

Theorem 11. *There exists a state-matching reduction from ARBAC97 to* TBA.

Proof. By construction. Presented is a mapping, and proof that the mapping satisfies the two properties for it to be a state-matching reduction.

The mapping, σ, needs to be able to map every $\langle\gamma, \psi\rangle$ in ARBAC97 to $\sigma(\langle\gamma, \psi\rangle) = \langle\gamma^T, \psi^T\rangle$ in TBA, as well as every q in ARBAC97 to $\sigma(q) = q^T$ in TBA.

$$
\begin{aligned}
\text{Let } \sigma(\gamma) = \gamma^T = &\langle S_\gamma, O_\gamma, tag_\gamma, P_\gamma\rangle \text{ where} \\
S_\gamma = &\mathcal{U} \\
O_\gamma = &\mathcal{P} \\
tag_\gamma = &\{\forall u \in \mathcal{U} : \langle u, u.id\rangle\} \cup UA \cup \{\forall p \in \mathcal{P} : \langle p, p.id\rangle\} \\
P_\gamma = &\{\forall \langle p, r\rangle \in PA : \text{“}r : p.id\text{”}\} \cup \{\forall r_1 \succeq r_2 \in RH : \text{“}r_1 \succeq r_2\text{”}\}
\end{aligned}
$$

Here, L_γ is the set of sentences of the form "$t_1 : t_2$" where $t_1, t_2 \in T$, and sentences of the form "$t_1 \succeq t_2$" where $t_1, t_2 \in T$. The inference procedure for L_γ is as follows. A sentence "$t_1 : t_2$" grants any subject with the tag t_1 access to objects with tag t_2. A sentence "$t_1 \succeq t_2$" implicitly grants any subject with the tag t_1 the additional tag t_2. That is, this sentence grants all permissions assigned to subjects with tag t_2 also to subjects with tag t_1.

Where $q =$ "Does user u belong to role r?", let $\sigma(q) = q^T =$ "Does subject u have tag r?". Where $q =$ "Is role r authorized to have permission p?", let $q^T =$ "Is policy sentence "$r : p.id$" in

the policy?". Where $q =$ "Is role r_1 a senior of role r_2?", let $q^T =$ "Is policy sentence "$r_1 \succeq r_2$" in the policy?". Where $q =$ "Is user u authorized to have permission p?", let $q^T =$ "Does subject u have access i_0 to object p?". Note that in this last form, we use i_0 to denote the only access right, since ARBAC97 uses permissions, not object/right pairs.

Let γ_0 be a start state in ARBAC97. Produce γ_0^T in TBA using σ. Given γ_k such that $\gamma_0 \overset{*}{\mapsto}_\psi \gamma_k$, we show that there exists γ_k^T such that $\gamma_0^T \overset{*}{\mapsto}_{\psi^T} \gamma_k^T$ where, for all q, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Consider the case where $\gamma_k = \gamma_0$, then let $\gamma_k^T = \gamma_0^T$. Then, consider each possible query q. If q is of the form "Does user u belong to role r?", q^T is "Does subject u have tag r?". In this case, $\gamma_k \vdash q$ if and only if $\langle u, r\rangle \in tag$, in which case $\langle u, r\rangle \in tag_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is role r authorized to have permission p?", q^T is "Is policy sentence "$r : p.id$" in the policy?". In this case, $\gamma_k \vdash q$ if and only if $\langle p, r\rangle \in PA$, in which case "$r, p.id$" $\in P_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is role r_1 a senior of role r_2?", q^T is "Is policy sentence "$r_1 \succeq r_2$" in the policy?". In this case, $\gamma_k \vdash q$ if and only if $r_1 \succeq r_2 \in RH$, in which case "$r_1 \succeq r_2$" $\in P$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is user u authorized to have permission p?", q^T is "Does subject u have access i_0 to object p?". In this case, $\gamma_k \vdash q$ if and only if $\exists r_1, r_2 \in \mathcal{R} : \langle u, r_1\rangle \in UA \wedge \langle p, r_2\rangle \in PA \wedge r_1 \succeq r_2 \in RH$, in which case $\langle u, r_1\rangle \in tag_\gamma \wedge$ "$r_2 : p.id$", "$r_1 \succeq r_2$" $\in P_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Thus, when $\gamma_k = \gamma_0$, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Next, consider some arbitrary γ_k reachable from γ_0. We construct γ_k^T that is reachable from γ_0^T and that answers every q^T the same way γ_k answers q, as follows. Consider each state-transition in the sequence $\gamma_0 \mapsto_\psi \gamma_1 \mapsto \ldots \mapsto \gamma_k$ in the ARBAC97 system. If the state-transition in ARBAC97 is the execution of `assignUser(`a`, `u`, `r`)`, we execute `assign_tags(`a`, `u`, {`r`})`. If the state-transition in ARBAC97 is the execution of `revokeUser(`a`, `u`, `r`)`, we execute `revoke_tags(`a`, `u`, {`r`})`. If the state-transition in ARBAC97 is the execution of `assignPermission(`a`, `p`, `r`)`, we execute `transform_policy(`a`, {`"$r : p.id$"`}, {})`. If the state-transition in ARBAC97 is the execution of `revokePermission(`a`, `p`, `r`)`, we execute `transform_policy(`a`, {}, {`"$r : p.id$"`})`. If the state-transition in ARBAC97 is the execution of `addToRange(`a`, `ξ`, `r`)`, where $\xi = (r_1, r_2), r \neq r_1, r \neq r_2$, we execute `transform_policy(`$a$`, {`"$r_1 \succeq r$", "$r \succeq r_2$"`}, {})`. If the state-transition in ARBAC97 is the execution of `removeFromRange(`a`, `ξ`, `r`)`, where $\xi = (r_1, r_2), r \neq r_1, r \neq r_2$, we execute `transform_policy(`$a$`, {}, {`"$r_1 \succeq r$", "$r \succeq r_2$"`}`. If the state-transition in ARBAC97 is the execution of `addAsSenior(`a`, `r`, `s`)`, we execute `transform_policy(`a`, {`"$r \succeq s$"`}, {})`. If the state-transition in ARBAC97 is the execution of `removeAsSenior(`a`, `r`, `s`)`, we execute `transform_policy(`a`, {}, {`"$r \succeq s$"`})`.

Now, consider each possible query q. If q is of the form "Does user u belong to role r?", q^T is "Does subject u have tag r?". In this case, $\gamma_k \vdash q$ if and only if $\langle u, r\rangle \in UA$. Such a state would be caused by an execution of `assignUser`, in which case we would have executed `assign_tags` to cause a state γ_k^T in which $\langle u, r\rangle \in tag_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is role r authorized to have permission p?", q^T is "Is policy sentence "$r : p.id$" in the policy?". In this case, $\gamma_k \vdash q$ if and only if $\langle p, r\rangle \in PA$. Such a state would be caused by an execution of `assignPermission`, in which case we would have executed `transform_policy` to cause a state γ_k^T in which "$r, p.id$" $\in P_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is role r_1 a senior of role r_2?", q^T is "Is policy sentence "$r_1 \succeq r_2$" in the policy?". In this case, $\gamma_k \vdash q$ if and only if $r_1 \succeq r_2 \in RH$. Such a state would be caused by an execution of `addToRange` or `addAsSenior`, in which case we would have executed `transform_policy` to cause a state γ_k^T in which "$r_1 \succeq r_2$" $\in P$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is

user u authorized to have permission p?", q^T is "Does subject u have access i_0 to object p?". In this case, $\gamma_k \vdash q$ if and only if $\exists r_1, r_2 \in \mathcal{R} : \langle u, r_1 \rangle \in UA \wedge \langle p, r_2 \rangle \in PA \wedge r_1 \succeq r_2 \in RH$. Such a state must be caused by an execution of `assignUser`, in which case we would have executed `assign_tags`; `assignPermission`, in which case we would have executed `transform_policy`; and either `addToRange` or `addAsSenior`, in which case we would have executed `transform_policy`. This combinition of commands would have resulted in a state in TBA in which $\langle u, r_1 \rangle \in tag_\gamma \wedge$ "$r_2 : p.id$", "$r_1 \succeq r_2$" $\in P_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Thus, in all cases, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Therefore, we've proven property (1) for state-matching reductions.

We prove that property (2) for a state-matching reduction is satisfied by our mapping also by construction. Let γ_0^T be a start-state in TBA corresponding to γ_0, the start-state in ARBAC97. Then, if γ_k^T is a state reachable from γ_0^T and q^T is a query in TBA whose corresponding query in ARBAC97 is q, we construct γ_k, a state in ARBAC97 reachable from γ_0 as follows. For each sentence in P of the form "$r : p.id$", we execute `assignPermission(`u_a`, p, r)`, where $u_a \in \mathcal{U}$ is an administrator. For each sentence in P of the form "$r_1 \succeq r_2$", we execute `addAsSenior(`u_a`, r1, r2)`, where $u_a \in \mathcal{U}$ is an administrator. For each $\langle u, r \rangle \in tag$ where $u \in S \wedge r \neq u.id$, we execute `assignUser(`$u_a$`, u, r)`, where $u_a \in \mathcal{U}$ is an administrator.

Now, consider each possible query q. If q is of the form "Does user u belong to role r?", q^T is "Does subject u have tag r?". In this case, $\gamma_k^T \vdash q^T$ if and only if $\langle u, r \rangle \in tag_\gamma$, in which case we would have executed `assignUser(`u_a`, u, r)`, creating a state in which $\langle u, r \rangle \in UR$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is role r authorized to have permission p?", q^T is "Is policy sentence "$r : p.id$" in the policy?". In this case, $\gamma_k^T \vdash q^T$ if and only if "$r, p.id$" $\in P_\gamma$, in which case we would have executed `assignPermission(`u_a`, p, r)`, creating a state in which $\langle p, r \rangle \in PA$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is role r_1 a senior of role r_2?", q^T is "Is policy sentence "$r_1 \succeq r_2$" in the policy?". In this case, $\gamma_k^T \vdash q^T$ if and only if "$r_1 \succeq r_2$" $\in P$, in which case we would have executed `addAsSenior(`u_a`, r1, r2)`, creating a state in which $r_1 \succeq r_2 \in RH$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Is user u authorized to have permission p?", q^T is "Does subject u have access i_0 to object p?". In this case, $\gamma_k^T \vdash q^T$ if and only if $\langle u, r_1 \rangle \in tag_\gamma \wedge$ "$r_2 : p.id$", "$r_1 \succeq r_2$" $\in P_\gamma$, in which case we would have executed `assignUser(`u_a`, u, r1)`, `assignPermission(`u_a`, p, r2)`, and `addAsSenior(`u_a`, r1, r2)`, creating a state in which $\exists r_1, r_2 \in \mathcal{R} : \langle u, r_1 \rangle \in UA \wedge \langle p, r_2 \rangle \in PA \wedge r_1 \succeq r_2 \in RH$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Thus, in all cases, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Therefore, we've proven property (2) for state-matching reductions, and proven that our mapping σ is a state-matching reduction.

C.3 Reduction to ARBAC97

Theorem 12. *There exists no state-matching reduction from TBA to ARBAC97.*

Proof. By contradiction. Assume there exists a state-matching reduction from TBA to ARBAC97. In TBA, adopt as γ a state where $s \in S$, $p^*, p' \in O$, $P = \{\}$, and L is as described in theorem 11 except the inference procedure is augmented as follows. The sentence "$t : p^*.id$", in addition to granting subjects with tag t access to objects with tag $p^*.id$, also grants such subjects access to objects with tag $p'.id$, regardless of whether P also contains the sentence "$t : p'.id$". This type of language may be useful when we want to have related permissions that imply one another. It is similar to L from the proof of theorem 10, but uses related objects rather than related

rights to the same object since ARBAC97 uses permissions instead of object/right pairs. Let $q_1 =$ "Does s have right i_0 to p^*?" and let $q_2 =$ "Does s have right i_0 to p'?".

Observe that $\gamma \vdash \neg q_1 \wedge \neg q_2$. Consider the state γ^A in ARBAC97 that is equivalent to γ (if there does not exist one, the contradiction of the existence of a state-matching reduction is found). We know that $\gamma^A \vdash \neg q_1^A \wedge \neg q_2^A$. Observe that there exists $\tilde{\gamma}^A$ reachable from γ^A such that $\tilde{\gamma}^A \vdash q_1^A \wedge \neg q_2^A$ via the execution of `assignPermission` and/or `assignUser`. However, the only $\tilde{\gamma}$ such that $\tilde{\gamma} \vdash q_1$ is that in which $\exists r_1, r_2 \in \mathcal{R} : \langle s, r_1 \rangle \in UA \wedge \langle p^*, r_2 \rangle \in PA \wedge r_1 \succeq r_2 \in RH$. In this case, due to the inference procedure of L, $\tilde{\gamma} \vdash q_1 \wedge q_2$, leaving no such $\tilde{\gamma}$ such that $\tilde{\gamma} \vdash q_1 \wedge \neg q_2$, so there is no $\tilde{\gamma}$ that is equivalent to $\tilde{\gamma}^A$. This contradicts property (2) for state-matching reductions, giving us the needed contradiction and proof of the non-existence of a state-matching reduction from TBA to ARBAC97.

D Comparison to BLP

In this section, we formalize the access control scheme used by the U.S. military that has both mandatory and discretionary components, the Bell-La Padula scheme (BLP), and show that TBA is strictly more expressive than BLP.

D.1 Formalization of BLP

In BLP, we assume the existence of the set of access attributes ($A = \{e, r, a, w\}$), the finite set used for both clearances and classifications (C), and the countably infinite set of possible compartments (P). These do not change and are thus not part of the state. A security level is an element of $C \times 2^P$. BLP is then defined as the state-transition system $\left\langle \Gamma^B, \Psi^B, Q^B, \vdash^B \right\rangle$. Each BLP state $\gamma^B \in \Gamma^B$ is defined by $\langle S, O, b, M, f_S, f_C, f_O \rangle$, where:

- S is the set of subjects.
- O is the hierarchical set of objects.
- $b \subseteq S \times O \times A$ is the current access set, and contains the tuple $\langle s, o, i \rangle$ for each subject s that has current access i to object o.
- $M : S \times O \to 2^A$ is the access matrix.
- $f_S : S \to C \times 2^P$ is the maximum security level function of subjects.
- $f_C : S \to C \times 2^P$ is the current security level function of subjects.
- $f_O : O \to C \times 2^P$ is the security level function of objects.

Ψ^B is defined using the following commands. Here, $\mathcal{P}(o)$ denotes the parent object of object o, and $\mathcal{C}(o)$ denotes the set of children objects of object o. Commands with names ending in x_i exist for each $x_i \in A$.

```
command  get-read(s, o)
    if r ∈ M[s, o] ∧ f_S(s) ⊀ f_O(o) ∧
    (s is trusted ∨ f_C(s) ⊀ f_O(o))
        b = b ∪ {⟨s, o, r⟩}

command  get-append(s, o)
    if a ∈ M[s, o] ∧
    (s is trusted ∨ f_O(o) ⊀ f_C(s))
        b = b ∪ {⟨s, o, a⟩}
```

command get-execute(s, o)
    if $e \in M[s, o]$
        $b = b \cup \{\langle s, o, e\rangle\}$

command get-write(s, o)
    if $w \in M[s, o] \wedge f_S(s) \succ f_O(o) \wedge$
    (s is trusted $\vee$ $f_C(s) = f_O(o)$)
        $b = b \cup \{\langle s, o, w\rangle\}$

command release-x_i(s, o)
    $b = b - \{\langle s, o, x_i\rangle\}$

command give-x_i(s_1, s_2, o)
    if (o is not the root of $O \wedge w \in M[s_1, \mathcal{P}(o)]$) $\vee$
    (o is the root of $O \wedge s_1$ is allowed to give access to
    the root object)
        $M[s_2, o] = M[s_2, o] \cup \{x_i\}$

command rescind-x_i(s_1, s_2, o)
    if (o is not the root of $O \wedge w \in M[s_1, \mathcal{P}(o)]$) $\vee$
    (o is the root of $O \wedge s_1$ is allowed to rescind access to
    the root object)
        $b = b - \{\langle s_2, o, x_i\rangle\}$
        $M[s_2, o] = M[s_2, o] - \{x_i\}$

command create-object(s, o, l)
    if w or $a \in M[s, o] \wedge l \succ f_O(o)$
        $f_O = f_O \cup \{\langle o_{new}, l\rangle\}$
        $O = O \cup \{o_{new}\}$ with o_{new} as a child of o

command delete-object-group(s, o)
    if o is not the root of $O \wedge w \in M[s, \mathcal{P}(o)]$
        for all s, x, for any o_i such that
        $o_i = o \vee o_i$ is below o in O
            $b = b - \{\langle s, o_i, x\rangle\}$
            $M[s, o_i] = \{\}$
            $O = O - \{o_i\}$

command change-subject-current-security-level(s, l)
    if $f_S \succ l \wedge$
    (s is trusted $\vee$ *-property would be satisfied)
        $f_C(s) = l$

command change-object-security-level(s, o, l)
    if the following conditions are true:
    • s is trusted $\wedge f_C(s) \succ f_O(o)$
      $\vee f_C(s) \succ l \wedge l \succ f_O(o)$
    • for all s_i such that $\langle s_i, o, r\rangle \in b$
      $\vee \langle s_i, o, w\rangle \in b$, $f_C(s) \succ l$
    • the change won't violate *-property

- $l \asymp f_O(\mathcal{P}(o)) \wedge \forall c \in \mathcal{C}(o) : f_O(c) \asymp l$
- s `is allowed to change security level of` o
  $f_O(o) = l$

Q^B includes all queries of the following forms: (1) "Does subject s have discretionary access i to object o?", (2) "Does subject s have current access i to object o?". $\vdash^T$ is defined as follows for queries of each of the forms above: (1) $true$ if and only if $i \in M[s, o]$, (2) $true$ if and only if $\langle s, o, i\rangle \in b$.

D.2 Reduction from BLP

Theorem 13. *There exists a state-matching reduction from BLP to* TBA.

Proof. By construction. Presented is a mapping, and proof that the mapping satisfies the two properties for it to be a state-matching reduction.

The mapping, σ, needs to be able to map every $\langle \gamma, \psi\rangle$ in BLP to $\sigma(\langle \gamma, \psi\rangle) = \langle \gamma^T, \psi^T\rangle$ in TBA, as well as every q in BLP to $\sigma(q) = q^T$ in TBA.

$$
\begin{aligned}
\text{Let } \sigma(\gamma) = \gamma^T = &\langle S_\gamma, O_\gamma, tag_\gamma, P_\gamma\rangle \text{ where}\\
S_\gamma = &S\\
O_\gamma = &O\\
tag_\gamma = &\{\forall s \in S : \langle s, f_S(s)[c]\rangle\} \cup \{\forall s \in S \forall p \in f_S(s)[p] : \langle s, p\rangle\} \cup\\
&\{\forall o \in O : \langle o, f_O(o)[c]\rangle\} \cup \{\forall o \in O \forall p \in f_O(o)[p] : \langle o, p\rangle\}\\
P_\gamma = &\{\forall \langle s, o, i\rangle \in b : \text{“}b \sim s : o : i\text{”}\} \cup\\
&\{\forall s \in S : \text{“}f_C \sim s : f_C(s)[c] : f_C(s)[p]\text{”}\} \cup\\
&\{\forall \langle s, o, i\rangle \in M : \text{“}M \sim s : o : i\text{”}\} \cup\\
&\{\forall o \in O : \text{“}O \sim o : \mathcal{P}(o)\text{”}\}
\end{aligned}
$$

Here, $I_\gamma = A$, and L_γ is the set of sentences of the form "$\phi \sim s : o : i$" where $\phi \in \{b, M\}$, $s \in S$, $o \in O$, $i \in I$, the set of sentences of the form "$f_C \sim s : c : \{p_1, \ldots, p_k\}$" where $s \in S$, $c \in C$, $p_1, \ldots, p_k \in P$, and the set of sentences of the form "$O \sim o_1 : o_2$", where $o_1, o_2 \in O$. The inference procedure for L_γ is as follows. A sentence "$b \sim s : o : i$" gives subject s the access i to object o, *unless* this would violate the *-property, in which case it has no effect. A sentence "$M \sim s : o : i$" gives subject s *discretionary* access i to object o, which is only used to answer certain types of queries. The sentence "$f_C \sim s : c : \{p_1, \ldots, p_k\}$" makes the effective security clearance for subject s be the lower of c and $f_S(s)[c]$, and its effective compartment set the intersection of $\{p_1, \ldots, p_k\}$ and $f_S(s)[p]$. Multiple lines of this form with the same s have the same effect as one line with the highest of the clearances and the union of all specified compartments. The sentence "$O \sim o_1 : o_2$" declares that o_2 is the parent object of o_1, and thus the $delete$ right to o_2 is denied while o_1 exists.

Where $q =$ "Does subject s have discretionary access i to object o?", let $\sigma(q) = q^T =$ "Is policy sentence "$M \sim s : o : i$" in the policy?". Where $q =$ "Does subject s have current access i to object o?", let $q^T =$ "Does subject u have access i to object o?".

Let γ_0 be a start state in BLP. Produce γ_0^T in TBA using σ. Given γ_k such that $\gamma_0 \overset{*}{\mapsto}_\psi \gamma_k$, we show that there exists γ_k^T such that $\gamma_0^T \overset{*}{\mapsto}_{\psi^T} \gamma_k^T$ where, for all q, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Consider the case where $\gamma_k = \gamma_0$, then let $\gamma_k^T = \gamma_0^T$. Then, consider each possible query q. If q is of the form "Does subject s have discretionary access i to object o?", q^T is "Is policy sentence "$M \sim s : o : i$" in the policy?". In this case, $\gamma_k \vdash q$ if and only if $i \in M[s, o]$, in which

case "$M \sim s : o : i$" $\in P_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Does subject s have current access i to object o?", q^T is "Does subject u have access i to object o?". In this case, $\gamma_k \vdash q$ if and only if $\langle s, o, i\rangle \in b$, in which case "$b \sim s : o : i$" $\in P_\gamma$. This will make the q^T true as long as *-property isn't violated by the access. However, this access can not violate *-property since the preconditions to the BLP commands would not have allowed it to be added to b if it did. Therefore, for all q, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Thus, when $\gamma_k = \gamma_0$, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Next, consider some arbitrary γ_k reachable from γ_0. We construct γ_k^T that is reachable from γ_0^T and that answers every q^T the same way γ_k answers q, as follows. Consider each state-transition in the sequence $\gamma_0 \mapsto_\psi \gamma_1 \mapsto \ldots \mapsto \gamma_k$ in the BLP system. If the state-transition in BLP is the execution of `get-`x_i`(`s`, `o`)`, we execute `transform_policy(`s`, {`"$b \sim s : o : x_i$"`}, {})`. If the state-transition in BLP is the execution of `release-`x_i`(`s`, `o`)`, we execute `transform_policy(`s`, {}, {`"$b \sim s : o : x_i$"`})`. If the state-transition in BLP is the execution of `give-`x_i`(`s_1`, `s_2`, `o`)`, we execute `transform_policy(`s`, {`"$M \sim s_2 : o : x_i$"`}, {})`. If the state-transition in BLP is the execution of `rescind-`x_i`(`s_1`, `s_2`, `o`)`, we execute `transform_policy(`s`, {}, {`"$M \sim s_2 : o : x_i$"`})`. If the state-transition in BLP is the execution of `create-object(`s`, `o`, `l`)`, we execute `create_object(`s`, `o`)`, followed by `assign_tags(`s`, `o`, `$\{l[c]\} \cup l[p]$`)`. If the state-transition in BLP is the execution of `delete-object-group(`s`, `o`)`, we execute `destroy_object(`s`, `o_i`)` for each $o_i \in \mathcal{C}(o)$, *then* execute `destroy_object(`s`, `o`)`. This must be done recursively so that no object is deleted before its children. If the state-transition in BLP is the execution of `change-subject-current-security-level(`s`, `l`)`, we execute `transform_policy(`s`, {`"$f_C \sim s : l[c] : l[p]$"`}, {`"$f_C \sim s : f_C(s)[c] : f_C(s)[p]$"`}`. If the state-transition in BLP is the execution of `change-object-security-level(`s`, `o`, `l`)`, we execute `assign_tags(`s`, `o`, `$\{l[c]\} \cup l[p]$`)`, followed by `revoke_tags(`s`, `o`, `T_i`)`, where T_i is the set of tags that o previously had (i.e., o's previous classification and compartments) that have been invalidated.

Now, consider each possible query q. If q is of the form "Does subject s have discretionary access i to object o?", q^T is "Is policy sentence "$M \sim s : o : i$" in the policy?". In this case, $\gamma_k \vdash q$ if and only if $i \in M[s, o]$. Such a state would be caused by an execution of `give-`i, in which case we would have executed `transform_policy` to cause a state γ_k^T in which "$M \sim s : o : i$" $\in P_\gamma$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Does subject s have current access i to object o?", q^T is "Does subject u have access i to object o?". In this case, $\gamma_k \vdash q$ if and only if $\langle s, o, i\rangle \in b$. Such a state would be caused by an execution of `get-`i, in which case we would have executed `transform_policy` to cause a state γ_k^T in which "$b \sim s : o : i$" $\in P_\gamma$. If this action violates *-property, it will not have the effect of granting the access, but in this case the command will fail at the preconditions in BLP. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Thus, in all cases, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Therefore, we've proven property (1) for state-matching reductions.

We prove that property (2) for a state-matching reduction is satisfied by our mapping also by construction. Let γ_0^T be a start-state in TBA corresponding to γ_0, the start-state in BLP. Then, if γ_k^T is a state reachable from γ_0^T and q^T is a query in TBA whose corresponding query in BLP is q, we construct γ_k, a state in BLP reachable from γ_0 as follows. For each $o \in O$, we find every $\langle o, t\rangle \in tag$ and build the set T_o including each t from this set of tuples. Then, we execute `change-object-security-level(`s_a`, `o`, `$\langle T_o \cap C, T_o \cap P\rangle$`)`, where s_a is an administrator. For each $o \in O$, we execute `create-object`, being careful to observe the sentences in P of the form "$O \sim o_1 : o_2$" describing the heirarchy of objects. For each sentence in P of the form "$M \sim s : o : i$", we execute `give-`i`(`s_a`, `s`, `o`)`, where

s_a is an administrator. For each sentence in P of the form "$f_C \sim s : c : \{p_1, \ldots, p_k\}$", we execute `change-subject-current-security-level(`$s$`,` $\langle c, \{p_1, \ldots, p_k\}\rangle$. For each sentence in P of the form "$b \sim s : o : i$", we execute `get-`i`(`s`,` o`)`.

Now, consider each possible query q. If q is of the form "Does subject s have discretionary access i to object o?", q^T is of the form "Is policy sentence "$M \sim s : o : i$" in the policy?". In this case, $\gamma_k^T \vdash q^T$ if and only if "$M \sim s : o : i$" $\in P$, in which case we would have executed `give-`i`(`s_a`,` s`,` o`)`, creating a state in which $i \in M[s, o]$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$. If q is of the form "Does subject s have current access i to object o?", q^T is of the form "Does subject u have access i to object o?". In this case, $\gamma_k^T \vdash q^T$ if and only if "$b \sim s : o : i$" $\in P$ (and the access wouldn't violate *-property), in which case we would have executed `get-`i`(`s`,` o`)`, creating a state in which $\langle s, o, i\rangle \in b$. Therefore, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Thus, in all cases, $\gamma_k^T \vdash q^T$ if and only if $\gamma_k \vdash q$.

Therefore, we've proven property (2) for state-matching reductions, and proven that our mapping σ is a state-matching reduction.

D.3 Reduction to BLP

Theorem 14. *There exists no state-matching reduction from* TBA *to BLP.*

Proof. By contradiction. Assume there exists a state-matching reduction from TBA to BLP. In TBA, adopt as γ a state where $s_1, s_2 \in S_a \subseteq S$, "$ON$" $\notin P$, and L is as described in theorem 13 except the inference procedure is augmented as follows. The sentence "ON" grants all subjects in S_a all rights over all objects, and the absence of this sentence implicitly denies all subjects in S_a any right over any object. This type of language may be useful, for example, when we have a set of power users whose full access we wish to enable and disable. Let $q_1 =$ "Does subject s_1 have right e to object o?" and let $q_2 =$ "Does subject s_2 have right e to object o?".

Observe that $\gamma \vdash \neg q_1 \wedge \neg q_2$. Consider the state γ^B in BLP that is equivalent to γ (if there does not exist one, the contradiction of the existence of a state-matching reduction is found). We know that $\gamma^B \vdash \neg q_1^B \wedge \neg q_2^B$. Observe that there exists $\tilde{\gamma}^B$ reachable from γ^B such that $\tilde{\gamma}^B \vdash q_1^B \wedge \neg q_2^B$ via the execution of `get-execute(`s_1`,` o`)`. However, the only $\tilde{\gamma}$ such that $\tilde{\gamma} \vdash q_1$ is that in which "ON" $\in P$. In this case, due to the inference procedure of L, $\tilde{\gamma} \vdash q_1 \wedge q_2$, leaving no such $\tilde{\gamma}$ such that $\tilde{\gamma} \vdash q_1 \wedge \neg q_2$, so there is no $\tilde{\gamma}$ that is equivalent to $\tilde{\gamma}^B$. This contradicts property (2) for state-matching reductions, giving us the needed contradiction and proof of the non-existence of a state-matching reduction from TBA to BLP.