Safety of $\text{ABAC}_\alpha$ is Decidable

Tahmina Ahmed and Ravi Sandhu

Institute for Cyber Security and Department of Computer Science,
University of Texas at San Antonio,
One UTSA Circle, San Antonio, Texas 78249, USA
tahmina.csebuet@gmail.com, ravi.sandhu@utsa.edu

Abstract. The $\text{ABAC}_\alpha$ model was recently defined with the motivation to demonstrate a minimal set of capabilities for attribute-based access control (ABAC) which can configure typical forms of the three dominant traditional access control models: discretionary access control (DAC), mandatory access control (MAC) and role-based access control (RBAC). $\text{ABAC}_\alpha$ showed that attributes can express identities (for DAC), security labels (for MAC) and roles (for RBAC). Safety analysis is a fundamental problem for any access control model. Recently, it has been shown that the pre-authorization usage control model with finite attribute domains ($\text{UCON}_{\text{preA}}^{\text{finite}}$) has decidable safety. $\text{ABAC}_\alpha$ is a pre-authorization model and requires finite attribute domains, but is otherwise quite different from $\text{UCON}_{\text{preA}}^{\text{finite}}$. This paper gives a state-matching reduction from $\text{ABAC}_\alpha$ to $\text{UCON}_{\text{preA}}^{\text{finite}}$. The notion of state-matching reductions was defined by Tripunitara and Li, as reductions that preserve security properties including safety. It follows that safety of $\text{ABAC}_\alpha$ is decidable.

Keywords: $\text{ABAC}_\alpha$, Safety

1 Introduction

Attribute-Based Access Control (ABAC) is gaining attention in recent years for its generalized structure and flexibility in policy specification [2]. Considerable research has been done and a number of formal models have been proposed for ABAC [3–6, 8, 10]. Among them $\text{UCON}_{\text{ABC}}$ [6] and $\text{ABAC}_\alpha$ [4] are two popular ABAC models. $\text{UCON}_{\text{ABC}}$ has been defined to continuously control usage of digital resources which covers authorizations, obligations, conditions, continuity and mutability, while $\text{ABAC}_\alpha$ is defined to configure DAC, MAC and RBAC which shows that attributes can express identities, security labels and roles. $\text{UCON}_{\text{preA}}^{\text{finite}}$ is a member of $\text{UCON}_{\text{ABC}}$ family of models which covers attribute based pre-authorization usage control with finite attribute domains.

Safety is a fundamental problem for any access control model. Harrison et al. [1] introduced the safety question in protection systems, which asks whether or not a subject s can obtain right r for an object o. They showed this problem is undecidable in general. A safety analyzer can answer decidable safety questions. A recent result shows that safety of $\text{UCON}_{\text{preA}}^{\text{finite}}$ is decidable [7]. Since $\text{UCON}_{\text{preA}}^{\text{finite}}$
ABAC\textsubscript{\alpha} allows unbounded creation of subjects and objects, in general a UCON\textsubscript{finite} system can grow without bound.

ABAC\textsubscript{\alpha} shares some characteristics with UCON\textsubscript{finite}. Both models restrict attributes to finite constant domains, and both allow unbounded creation of subjects and objects. Nonetheless there are significant differences between the two models, as discussed in Sections 2 and 3. The central result of this paper is that the safety problem for ABAC\textsubscript{\alpha} can be reduced to that for UCON\textsubscript{finite}, and hence is decidable. Our reduction follows the notion of state-matching \cite{9} and preserves security properties, including safety.

The rest of the paper is organized as follows. Section 2 reviews the ABAC\textsubscript{\alpha} model, and provides a slightly re-casted, but essentially identical, formal definition relative to its original definition \cite{4}. Section 3 reviews the formal description of UCON\textsubscript{finite} model. Section 4 presents a reduction from ABAC\textsubscript{\alpha} to UCON\textsubscript{finite}. Section 5 proves that the reduction of Section 4 is state-matching, from which decidability of ABAC\textsubscript{\alpha} follows. Section 6 concludes the paper.

2 The ABAC\textsubscript{\alpha} Formal Model (Review)

ABAC\textsubscript{\alpha} is an ABAC model that has “just sufficient” features to be “easily and naturally” configured to do DAC, MAC and RBAC \cite{4}. The core components of this model are: users (U), subjects (S), objects (O), user attributes (UA), subject attributes (SA), object attributes (OA), permissions (P), authorization policy, creation and modification policy, and policy languages. The structure of ABAC\textsubscript{\alpha} model is shown in Fig. 1. Table 1 gives the formal definition of ABAC\textsubscript{\alpha}.

2.1 Users, Subjects, Objects and their Attributes

Users (U) represent human beings in an ABAC\textsubscript{\alpha} system who create and modify subjects, and access resources through subjects. Subjects (S) are processes created by users to perform some actions in the system. ABAC\textsubscript{\alpha} resources are represented as Objects (O). Users, subjects and objects are mutually disjoint in ABAC\textsubscript{\alpha}, and are collectively called entities. NAME is the set of all names for various entities in the system. Attributes are set-valued or atomic-valued
### Table 1. ABAC α Formal Model

<table>
<thead>
<tr>
<th>Basic Sets and Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>U, S, O are finite sets of existing users, subjects and objects</td>
</tr>
<tr>
<td>UA = {ua_1, ua_2, \ldots, ua_l}, finite set of user attributes</td>
</tr>
<tr>
<td>SA = {sa_1, sa_2, \ldots, sa_m}, finite set of subject attributes</td>
</tr>
<tr>
<td>OA = {oa_1, oa_2, \ldots, oa_n}, finite set of object attributes</td>
</tr>
<tr>
<td>SubCreator: S → U. A system function, specifies the creator of a subject.</td>
</tr>
<tr>
<td>attType: UA ∪ SA ∪ OA → {set, atomic}</td>
</tr>
<tr>
<td>For each attribute att ∈ UA ∪ SA ∪ OA:</td>
</tr>
<tr>
<td>SCOPE(att) denotes the finite set of atomic values for attribute att.</td>
</tr>
<tr>
<td>Range(att) represents a finite set of atomic or set values as the range of att.</td>
</tr>
</tbody>
</table>
| Range(att) = \{SCOPE(att) \text{ attType(att) = atomic} \}
| \{^2\text{SCOPE(att)} \text{ attType(att) = set} \} |

#### Tuple Notation

- \text{uavtf}: U → \text{UAVT}, current attribute value tuple for a user
- \text{savtf}: S → \text{SAVT}, current attribute value tuple for a subject
- \text{oavtf}: O → \text{OAVT}, current attribute value tuple for an object

#### Authorization Policy

- P = \{p_1, p_2, \ldots, p_n\}, a finite set of permissions.
- For each p ∈ P, Authorization_p(s:S, o:O) returns true or false.
- Specified in language LAuthorization.

#### Creation and Modification Policy

- Subject Creation Policy:
  - ConstrSub(u:U, s:NAME, savt:SAVT) returns true or false.
  - Specified in language LConstrSub.
- Subject Modification Policy:
  - ConstrSubMod(u:U, s:S, savt:SAVT) returns true or false.
  - Specified in language LConstrSubMod.
- Object Creation Policy:
  - ConstrObj(s:S, o:NAME, oavt:OAVT) returns true or false.
  - Specified in language LConstrObj.
- Object Modification Policy:
  - ConstrObjMod(s:S, o:O, oavt:OAVT) returns true or false.
  - Specified in language LConstrObjMod.

#### Policy Languages

- Each policy language is an instantiation of the Common Policy Language CPL that varies only in the values it can compare. Table 2 defines CPL for ABAC α.

#### Functional Specification

ABAC α operations are formally specified in Table 3.
Table 2. Definition of CPL

<table>
<thead>
<tr>
<th>CPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi ::= \varphi \land \varphi \mid \varphi \lor \varphi \mid ( \varphi ) \mid \neg \varphi \mid \exists x \in \text{set}.\varphi \mid \forall x \in \text{set}.\varphi \mid \text{set setcompare set} \mid \text{atomic} \in \text{set} \mid \text{atomic atomic compare atomic} \mid \text{set compare} ::= \subset \mid \subseteq \mid \subseteq \mid \text{atomic compare} ::= \prec \mid = \mid \leq</td>
</tr>
</tbody>
</table>

functions which take an entity (user, subject or object) and return a value from a finite set of atomic values. Each user, subject, object is associated with a finite set of user attributes (UA), subject attributes (SA) and object attributes (OA) respectively. Each attribute is a set-valued or atomic-valued function. attType is a function that returns type of the attribute, i.e., whether it is set or atomic valued. SCOPE represents the domain of an attribute which is a finite set of atomic values. Potentially infinite domain attribute such as location, age are represented as large finite domains. For each attribute $\text{att}$, SCOPE($\text{att}$) can be an unordered, a totally ordered or a partially ordered set. Range($\text{att}$) is a finite set of all possible atomic or set values for attribute $\text{att}$. Each attribute takes a user or a subject or an object, and returns a value from its range. SubCreator is a system function which specifies the creator of a subject. SubCreator is assigned by the system at subject creation time, and cannot change. UAVT, SAVT, OAVT are sets of all possible Attribute Value Tuples for users, subjects and objects respectively. The functions uavtf, savtf and oavtf, return current attribute value tuples for a particular user, subject or object respectively.

2.2 Authorization Policy

ABAC$_{\alpha}$ authorization policy consists of a single authorization policy for each permission. Permissions are privileges that a user can hold on objects and exercise through subjects. It enables access of a subject on an object in a particular mode, such as read or write. $P = \{ p_1, p_2, \ldots, p_n \}$ is a finite set of permissions. Each Authorization Policy is a boolean function which is associated with a permission, and takes a subject and an object as input and returns true or false based on the boolean expression built from attributes of that subject and object.

2.3 Creation and Modification Policy

User creation, attribute value assignment of user at creation time, user deletion and modification of a user’s attribute values is done by security administrator, and is outside the scope of ABAC$_{\alpha}$. Subject creation and assigning attribute value to subject during creation time is constrained by the values of user attributes. Only creator is allowed to terminate and modify attributes of a subject. Modification of subject attributes is constrained by the creating user’s attribute values, and existing and new attribute values of the concerned subject.$^1$ Objects are created by subjects. Object creation and attribute value assignment

$^1$ In the original definition of ABAC$_{\alpha}$ [4] subject creation and modification have identical policies. However, a correct configuration of MAC in ABAC$_{\alpha}$ requires different policies for these two operations. Hence, we define ABAC$_{\alpha}$ here to have separate policies for these two operations.
Table 3. Functional Specification of ABAC\textsubscript{\alpha} operations

<table>
<thead>
<tr>
<th>Operations</th>
<th>Conditions</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access\textsubscript{p}(s,o)</td>
<td>s ∈ S ∧ o ∈ O ∧ Authorization\textsubscript{p}(s,o)</td>
<td></td>
</tr>
<tr>
<td>CreateSubject (u,s:NAME,savt:SAVT)</td>
<td>u ∈ U ∧ s ∈ S ∧ ConstrSub(u, s, savt)</td>
<td>S’ = S ∪ {s}</td>
</tr>
<tr>
<td>DeleteSubject (u,s:NAME)</td>
<td>s ∈ S ∧ u ∈ U ∧ SubCreator(s) = u</td>
<td>S’ = S {s}</td>
</tr>
<tr>
<td>ModifySubjectAtt (u,s:NAME,savt:SAVT)</td>
<td>u ∈ U ∧ s ∈ S ∧ SubCreator(s) = u ∧ ConstrSubMod(u, s, savt)</td>
<td>savt(s) = savt</td>
</tr>
<tr>
<td>CreateObject (s,o:NAME,oavt:OAVT)</td>
<td>s ∈ S ∧ o ∈ O ∧ ConstrObj(s, o, oavt)</td>
<td>O’ = O ∪ {o}</td>
</tr>
<tr>
<td>ModifyObjectAtt (s,o:NAME,oavt:OAVT)</td>
<td>s ∈ S ∧ o ∈ O ∧ ConstrObjMod(s, o, oavt)</td>
<td>oavt(o) = oavt</td>
</tr>
</tbody>
</table>

at creation time is constrained by creating subject’s attribute values and proposed attribute value for the object. Modification of object attribute value is constrained by subject and object’s existing attribute values and proposed attribute values for object. ABAC\textsubscript{\alpha} has subject deletion however there is no object deletion. An existing subject can be deleted only by its creator.

2.4 Policy Languages

Each policy is expressed using a specific language. CPL is the common policy language part for each language. Each language is a CPL instantiation with different values for set and atomic. CPL is defined in Table 2.

Authorization Policy: The boolean expression of authorization policy is defined using the language LAuthorization which is a CPL instantiation where set and atomic refers to the set and atomic valued attribute of concerned subject and object.

Creation and Modification Policy: Subject creation, subject attribute modification, object creation and object attribute modification policies are all boolean expressions and defined using LConstrSub, LConstrSubMod, LConstrObj and LConstrObjMod respectively. LConstrSub is a CPL instantiation where set and atomic refers to the set and atomic valued attribute of creating user and proposed attribute value for subject being created. LConstrSubMod is a CPL instantiation where set and atomic refers to the set and atomic valued attribute of concerned user and subject and proposed attribute values for the object. LConstrObj is a CPL instantiation where set and atomic refers to the set and atomic valued attribute value of concerned subject and object and proposed attribute values for the object.
2.5 Functional Specification

ABAC$_\alpha$ functional specification has six operations. The operations are: access an object by a subject, creation of subject and object, deletion of subject, modification of subject and object attributes. Each ABAC$_\alpha$ operation has two parts: condition part and update part. Table 3 shows the specification of condition and update parts for ABAC$_\alpha$ operations.

3 The UCON$^{\text{finite preA}}$ Model (Review)

In usage control authorization model entities are subjects and objects, and subjects are a subset of objects. Each object has a unique identifier and a finite set of attributes. Attributes can be mutable or immutable. Usage control Pre-Authorization model (UCON$^{\text{preA}}$) evaluates authorization decisions of permission prior to the execution of commands. Figure 2 shows the components of UCON$^{\text{preA}}$ model.

The UCON$^{\text{finite preA}}$ model, i.e., pre-authorization UCON with finite attributes, is defined through a usage control scheme [7], as follows.

1. Object schema OS$_\Delta$, is of the form $\{a_1: \sigma_1, \ldots, a_n: \sigma_n\}$ where each $a_i$ is the name of an attribute and $\sigma_i$ is a finite set specifying $a_i$'s domain. UCON$^{\text{finite preA}}$ considers single object schema for different objects and considers only atomic values for each domain $\sigma_i$.
2. UR = $\{r_1, r_2, \ldots, r_k\}$, a set of usage rights, where $r_i$ defines a permission enabled by a usage control command.
3. UC = $\{UC_1, UC_2, \ldots, UC_l\}$, a set of usage control commands.
4. ATT = $\{a_1, a_2, \ldots, a_n\}$, a finite set of object attributes.
5. AVT = $\sigma_1 \times \ldots \times \sigma_n$, set of all possible attribute value tuples.
6. avtf: O $\rightarrow$ AVT, returns existing attribute value tuple of an object.
Table 4. UCON_{preA}^{finite} Command Structure

<table>
<thead>
<tr>
<th>Non-Creating Command</th>
<th>Creating Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command Name</strong>, (s,o)</td>
<td><strong>Command Name</strong>, (s,o)</td>
</tr>
<tr>
<td><strong>PreCondition</strong>: ( f_1(s,o) \rightarrow {\text{true, false}} )</td>
<td><strong>PreCondition</strong>: ( f_2(s) \rightarrow {\text{true, false}} )</td>
</tr>
<tr>
<td><strong>PreUpdate</strong>: ( s.a_i := f_{1,a_i}(s,o) );</td>
<td><strong>PreUpdate</strong>: create o;</td>
</tr>
<tr>
<td>( s.a_i := f_{2,a_i}(s,o) );</td>
<td>( o.a_j := f_{2,a_j}(s) );</td>
</tr>
<tr>
<td>( o.a_j := f_{2,a_j}(s,o) );</td>
<td>( o.a_j := f_{2,a_j}(s) );</td>
</tr>
</tbody>
</table>

7. Each command in UC is associated with a right and has two formal parameters \( s \) and \( o \), where \( s \) is a subject trying to access object \( o \) with right \( r \). A single right can be associated with more than one command. So \( l \geq k \), that means number of commands \( (l) \) can be \( \geq \) number of rights \( (k) \). There are two types of usage control commands, Non-Creating Command and Creating Command. Each command has a precondition part and an update part. Table 4 shows the structure of non-creating and creating command of UCON_{preA}^{finite}.

(a) In UCON_{preA}^{finite} non-creating command, \( f_1(s,o) \) is a boolean function which takes the attribute values of \( s \) and \( o \) and returns true or false. If the result is true then the PreUpdate is performed with zero or more attributes of \( s \) and \( o \) independently updated to new values computed from their attribute values prior to the command execution. Also the usage right \( r \) is granted. Otherwise the command terminates without granting \( r \). \( f_1 \) and \( f_2 \) are the computing functions for new values which use existing attribute values of \( s \) and \( o \).

(b) In UCON_{preA}^{finite} creating command, \( f_2(s) \) is a boolean function which takes the attribute values of \( s \) and returns true or false. If the result is true then the PreUpdate is performed with zero or more attributes of \( s \) updated to new values computed from the attribute values of \( s \). All attributes of the newly created object \( o \) are assigned computed attribute values. Also the usage right \( r \) is granted. Otherwise the command terminates without granting \( r \). \( f_1 \) and \( f_2 \) are computing functions for new values which use existing attribute values of \( s \).

4 Reduction from ABAC_{\alpha} to UCON_{preA}^{finite}

In this section we define a reduction from ABAC_{\alpha} to UCON_{preA}^{finite}. For convenience we introduce policy evaluation functions and sets of eligible attribute
value tuples for creation and modification of subjects and objects of ABAC\(\alpha\). We also introduce the PreCondition evaluation functions of UCON\(\text{pre}\) which we will use in the next section. These additional notations enable us to relate the machinery of these two models.

4.1 Policy Evaluation Functions for ABAC\(\alpha\)

Each Policy evaluation function evaluates corresponding policy and returns true or false.

**Authorization Policy Evaluation Function:** \(\text{ChkAuth}(p, \text{savtf}(s), \text{oavtf}(o))\) returns true or false. This function evaluates the authorization policy Authoriz\(\text{ation}(p,s,o)\) to determine whether a subject \(s\) is allowed to have permission \(p\) on object \(o\).

**Creation and Modification Policy Evaluation Functions:**
- \(\text{ChkConstrSub}(\text{uavtf}(u), \text{savt})\) returns true or false. It evaluates the subject creation policy ConstrSub\((u,s,\text{savt})\) as to whether a user \(u\) with attribute value tuple \(\text{uavtf}(u)\) is allowed to create a subject \(s\) with attribute value tuple \(\text{savt}\).
- \(\text{ChkConstrSubMod}(\text{uavtf}(u), \text{savtf}(s), \text{savt})\) returns true or false. It evaluates the subject modification policy ConstrSubMod\((u,s,\text{savt})\) as to whether a user \(u\) with attribute value tuple \(\text{uavtf}(u)\) is allowed to modify a subject \(s\) with attribute value tuple \(\text{savtf}(s)\) to \(\text{savt}\).
- \(\text{ChkConstrobj}(\text{savtf}(s), \text{oavt})\) returns true or false. It evaluates the object creation policy ConstrObj\((s,o,\text{oavt})\) as to whether a subject \(s\) with attribute value tuple \(\text{savtf}(s)\) is allowed to create an object \(o\) with attribute value tuple \(\text{oavt}\).
- \(\text{ChkConstrobjMod}(\text{savtf}(s), \text{oavtf}(o), \text{oavt})\) returns true or false. It evaluates the object modification policy ConstrObjMod\((s,o,\text{oavt})\) as to whether a subject \(s\) with attribute value tuple \(\text{savtf}(s)\) is allowed to modify an object \(o\) with attribute value tuple \(\text{oavtf}(o)\) to \(\text{oavt}\).

4.2 Sets of Eligible Attribute Value Tuples

Using the policy evaluation functions for ABAC\(\alpha\) we define 4 eligible sets for attribute value tuples as follows.

**Definition 1.** set of user-subject-creatable-tuples
\[ \text{UAVTCrSAVT} \subseteq \text{UAVT} \times \text{SAVT} \]
\[ \text{UAVTCrSAVT} = \{ (i,j) \mid i \in \text{UAVT} \land j \in \text{SAVT} \land \text{ChkConstrSub}(i,j) \} \]

**Definition 2.** set of user-subject-modifiable-tuples
\[ \text{UAVTModSAVT} \subseteq \text{UAVT} \times \text{SAVT} \times \text{SAVT} \]
\[ \text{UAVTModSAVT} = \{ (i,j,k) \mid i \in \text{UAVT} \land j \in \text{SAVT} \land k \in \text{SAVT} \land \text{ChkConstrSubMod}(i,j,k) \} \]
Table 5. Reduction from ABAC\textsubscript{\textvisiblespace}$\alpha$ to UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$

<table>
<thead>
<tr>
<th>Object Schema( OS\textsubscript{\textvisiblespace}$\Delta$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>[{\text{entity_type:} {\text{user, subject, object}}, \text{user_name:} U^{ABAC\textsubscript{\textvisiblespace}$\alpha$}, \text{SubCreator:} U^{ABAC\textsubscript{\textvisiblespace}$\alpha$}, \text{isDeleted:} {\text{true, false}}, \text{ua_1:Range}(ua_1), \ldots, \text{ua_m:Range}(ua_m), \text{sa_1:Range}(sa_1), \ldots, \text{sa_n:Range}(sa_n), \text{oa_1:Range}(oa_1), \ldots, \text{oa_p: Range}(oa_p)}]</td>
</tr>
<tr>
<td><strong>Attributes:</strong></td>
</tr>
<tr>
<td>ATT = {entity_type, user_name, SubCreator, isDeleted}</td>
</tr>
<tr>
<td>(\cup) U\textsuperscript{ABAC\textsubscript{\textvisiblespace}$\alpha$} \cup S\textsuperscript{ABAC\textsubscript{\textvisiblespace}$\alpha$} \cup O\textsuperscript{ABAC\textsubscript{\textvisiblespace}$\alpha$}</td>
</tr>
<tr>
<td><strong>Usage Rights:</strong></td>
</tr>
<tr>
<td>UR = P\textsuperscript{ABAC\textsubscript{\textvisiblespace}$\alpha$} \cup {d}</td>
</tr>
<tr>
<td><strong>Commands:</strong></td>
</tr>
<tr>
<td>UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$ commands are defined in Table 6 and 7</td>
</tr>
</tbody>
</table>

**Definition 3.** set of subject-object-creatable-tuples
\[
\text{SAVTCrOAVT} \subseteq \text{SAVT} \times \text{OAVT}
\]
\[
\text{SAVTCrOAVT} = \{(i, j) \mid i \in \text{SAVT} \land j \in \text{OAVT} \land \text{ChkConstrObj}(i, j)\}
\]

**Definition 4.** set of subject-object-modifiable-tuples
\[
\text{SAVTModOAVT} \subseteq \text{SAVT} \times \text{OAVT} \times \text{OAVT}
\]
\[
\text{SAVTModOAVT} = \{(i, j, k) \mid i \in \text{SAVT} \land j \in \text{OAVT} \land k \in \text{OAVT} \land \text{ChkConstrObjMod}(i, j, k)\}
\]

### 4.3 PreCondition Evaluation Functions for UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$

PreCondition evaluation functions of UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$ check the PreConditions of UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$ commands and return true or false.

- **CheckPCNCR**(uc\textsubscript{rcr}, avtf\textsubscript{f}(s), avtf\textsubscript{f}(o), avt\textsubscript{1}, avt\textsubscript{2}) returns true or false. It evaluates the PreCondition \(f_b(s, o)\) and PreUpdate of non-creating command uc\textsubscript{rcr}(s, o) as to whether a subject s is allowed to execute command uc\textsubscript{rcr} on object o and if allowed whether it modifies s’s attribute value tuple from avtf\textsubscript{f}(s) to avt\textsubscript{1} and o’s attribute value tuple from avtf\textsubscript{f}(o) to avt\textsubscript{2}.

- **CheckPCCR**(uc\textsubscript{rcr}, avtf\textsubscript{f}(s), o, avt\textsubscript{1}, avt\textsubscript{2}) returns true or false. It evaluates the PreCondition \(f_b(s)\) and PreUpdate of creating command uc\textsubscript{rcr}(s, o) as to whether a subject s is allowed to execute the command uc with right r and if allowed whether it creates object o with attribute value tuple to avt\textsubscript{2} and modifies s’s own attribute value tuple from avtf\textsubscript{f}(s) to avt\textsubscript{1}.

### 4.4 Reduction from ABAC\textsubscript{\textvisiblespace}$\alpha$ to UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$

The reduction is presented showing the configuration of UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$ object schema, rights and commands to do ABAC\textsubscript{\textvisiblespace}$\alpha$. Table 5 shows the reduction.

**Object Schema of UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$:** Every ABAC\textsubscript{\textvisiblespace}$\alpha$ entity (user, subject, object) is represented as a UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$ object and the attribute entity\_type specifies whether a particular UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$ object is ABAC\textsubscript{\textvisiblespace}$\alpha$ user, subject or object. User, subject and object attributes of ABAC\textsubscript{\textvisiblespace}$\alpha$ are represented as UCON\textsubscript{\textvisiblespace}$_{preA}^{\text{finite}}$ object attributes. There is no user creation in ABAC\textsubscript{\textvisiblespace}$\alpha$ so U\textsuperscript{ABAC\textsubscript{\textvisiblespace}$\alpha$} is a finite set. ABAC\textsubscript{\textvisiblespace}$\alpha$
Table 6. $\text{UCON}_{\text{preA}}^\text{finite}$ Non-Creating Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>PreCondition</th>
<th>PreUpdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Access}_d(s,o)$</td>
<td>$s$.entity.type = user \land o.entity.type = subject \land o.SubCreator = s.user.name \land o.isDeleted = false</td>
<td>$\text{N/A}$</td>
</tr>
<tr>
<td>$\text{DeleteSubject}_d(s,o)$</td>
<td>$s$.entity.type = user \land o.entity.type = subject</td>
<td>$o$.isDeleted = true</td>
</tr>
<tr>
<td>$\text{ModifySubjectAtt}_{ijk}(s,o)$</td>
<td>$s$.entity.type = user \land o.entity.type = subject \land o.SubCreator = s.user.name \land o.isDeleted = false \land s.isDeleted = false \land (s.ua_1, ..., s.ua_n) = (i_1, ..., i_m) \land (o.sa_1, ..., o.sa_n) = (j_1, ..., j_n)</td>
<td>$o$.sa_1 = k_1 \land \vdots \land o$.sa_n = k_n \land o$.oa_1 = k_1 \land \vdots \land o$.oa_p = k_p</td>
</tr>
<tr>
<td>$\text{ModifyObjectAtt}_{ijk}(s,o)$</td>
<td>$s$.entity.type = subject \land o.entity.type = object \land s.isDeleted = false \land (s.sa_1, ..., s.sa_n) = (i_1, ..., i_n) \land (o.oa_1, ..., o.oa_p) = (j_1, ..., j_p)</td>
<td>$\text{N/A}$</td>
</tr>
</tbody>
</table>

Function $\text{SubCreator}$ is configured here with a mandatory $\text{UCON}_{\text{preA}}^\text{finite}$ object attribute whose domain would be finite set of users ($U_{\text{ABAC}}^\text{ABAC}$). To determine which user is the creator of an $\text{ABAC}_\alpha$ subject, $\text{UCON}_{\text{preA}}^\text{finite}$ object needs to have another mandatory attribute $\text{user.name}$ whose range is also finite set of users ($U_{\text{ABAC}}^\text{ABAC}$). $\text{ABAC}_\alpha$ has a subject deletion operation. In [7] it is shown that deletion of a subject can be simulated by using a special boolean attribute $\text{isDeleted}$ which has a boolean domain. We consider “NULL” as a special attribute value for any atomic or set valued attribute. It is assigned to an attribute which is not appropriate for a particular entity. We need to add “NULL” in the range of $U_{\text{ABAC}}^\text{ABAC}$ for this reduction. As there is no user deletion and object deletion in $\text{ABAC}_\alpha$, $\text{isDeleted}$ would be “NULL” for both users and objects. $\text{UCON}_{\text{preA}}^\text{finite}$ attribute set $\text{ATT} = \{\text{entity.type}, \text{user.name}, \text{SubCreator}, \text{isDeleted}\} \cup U_{\text{ABAC}}^\text{ABAC} \cup S_{\text{ABAC}} \cup O_{\text{ABAC}}$.

$\text{UCON}_{\text{preA}}^\text{finite}$ usage rights $\text{UR}$: In this reduction each $\text{ABAC}_\alpha$ permission is considered as a usage right in $\text{UCON}_{\text{preA}}^\text{finite}$ and additionally a dummy right $d$ is introduced. Each $\text{UCON}_{\text{preA}}^\text{finite}$ command associates with a right. We use dummy right $d$ for association with the commands which are defined to configure $\text{ABAC}_\alpha$ operations. Usage Right $\text{UR}_{\text{preA}}^\text{finite} = P_{\text{ABAC}}^\text{ABAC} \cup \{d\}$.

$\text{UCON}_{\text{preA}}^\text{finite}$ commands: $\text{ABAC}_\alpha$ operations are reduced to specific $\text{UCON}_{\text{preA}}^\text{finite}$ commands. We use the sets of eligible attribute value tuples to define $\text{UCON}_{\text{preA}}^\text{finite}$ commands. It defines a creating command for each element of $U_{\text{AVT}}^{\text{CrSAVT}}$ and $\text{SAVTCrOAVT}$ and a non-creating command for each element of $U_{\text{AVT}}^{\text{ModSAVT}}$ and $\text{SAVTModOAVT}$. For example consider an $\text{ABAC}_\alpha$ subject creation policy where a user $u$ with attribute value tuple $uavt$ is allowed to create a subject $s$ with attribute value tuple $savt$, so by definition $\langle uavt, savt \rangle$
Table 7. UCON_finite \_preA Creating Commands

<table>
<thead>
<tr>
<th>For each ( (i, j) \in \text{UAVTCrSAVT} )</th>
<th>For each ( (i, j) \in \text{SAVTCrOAVT} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{CreateSubject}_{ij}(s,o) (</td>
<td>)</td>
</tr>
<tr>
<td>PreCondition: ( s.entity_{type} = \text{user} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( \land \langle s.ua_1, \ldots, s.ua_m \rangle = \langle i_1, \ldots, i_m \rangle ) (</td>
<td>)</td>
</tr>
<tr>
<td>PreUpdate: create ( o ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.entity_{type} = \text{subject} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.user_{name} = \text{NULL} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.SubCreator = s.user_{name} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.isDeleted = \text{false} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.ua_1 = \text{NULL} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( \vdots ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.ua_m = \text{NULL} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.sa_1 = j_1 ) (</td>
<td>)</td>
</tr>
<tr>
<td>( \vdots ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.sa_n = j_n ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.oa_1 = \text{NULL} ) (</td>
<td>)</td>
</tr>
<tr>
<td>( \vdots ) (</td>
<td>)</td>
</tr>
<tr>
<td>( o.oa_p = \text{NULL} ) (</td>
<td>)</td>
</tr>
</tbody>
</table>

\( \in \text{UAVTCrSAVT} \). For each element \( (i, j) \in \text{UAVTCrSAVT} \) this reduction has a command named \textbf{CreateSubject}_{ij}(s,o) \(|\) which creates an object \( o \) with entity_{type} = subject. Each Access_{r}^{\text{UCON}_\text{finite}_\text{preA}}(s,o) \(|\) configures Access_{p}^{\text{ABAC}_\alpha}(s,o) \(|\) where \( r = p \). Here Access_{r}^{\text{UCON}_\text{finite}_\text{preA}}(s,o) \(|\) is a non-creating command with PreCondition part only and PreCondition checks the authorization evaluation function of ABAC_\alpha. Each DeleteSubject_{j}^{\text{UCON}_\text{finite}_\text{preA}}(s,o) \(|\) configures DeleteSubject_{j}^{\text{ABAC}_\alpha}(u,s) \(|\) which is also a non-creating command and sets o.isDeleted = true. Tables 6 and 7 show the configuration of non-creating and creating commands for this construction.

5 Safety of ABAC_\alpha

In this section we show that safety of ABAC_\alpha is decidable. We prove that the reduction provided in the previous section is state matching, so it preserves security properties including safety. Decidable safety for ABAC_\alpha then follows from decidable safety for UCON_finite_\text{preA}.

Tripunitara and Li [9] define an access control model as a set of access control schemes. An access control scheme is a state transition system \( \langle \Gamma, \Psi, Q, \vdash \rangle \), where \( \Gamma \) is a set of states, \( \Psi \) is a set of state transition rules, \( Q \) is a set of queries and \( \vdash \): \( \Gamma \times Q \rightarrow \{ \text{true}, \text{false} \} \) is the entailment relation. The notion of state-matching reduction is defined as follows.
Definition 5. State Matching Reduction:
Given two schemes A and B and a mapping A to B, \( \sigma : (\Gamma^A \times \Psi^A) \cup Q^A \to (\Gamma^B \times \Psi^B) \cup Q^B \), we say that the two states \( \gamma^A \) and \( \gamma^B \) are equivalent under the mapping \( \sigma \) when for every \( q^A \in Q^A \), \( \gamma^A \vdash^A q^A \) if and only if \( \gamma^B \vdash^B \sigma(q^A) \).
A mapping \( \sigma \) from A to B is said to be a state-matching reduction if for every \( \gamma^A \in \Gamma^A \) and every \( \psi^A \in \Psi^A \), \( \langle \gamma^A, \psi^A \rangle = \sigma(\langle \gamma^A, \psi^A \rangle) \) has the following two properties:

1. For every \( \gamma^A \) in scheme A such that \( \gamma^A \vdash^A \gamma^A \), there exists a state \( \gamma^B \) such that \( \gamma^B \vdash^B \gamma^B \) and \( \gamma^A \) and \( \gamma^B \) are equivalent under \( \sigma \).
2. For every \( \gamma^B \) in scheme B such that \( \gamma^B \vdash^B \gamma^B \), there exists a state \( \gamma^A \) such that \( \gamma^A \vdash^A \gamma^A \) and \( \gamma^B \) and \( \gamma^A \) are equivalent under \( \sigma \).

In order to show that a reduction from \( \text{ABAC}_\alpha \) and \( \text{UCON}_\alpha^{\text{finite pre}_\alpha} \) is state matching, we have to show the following:

1. Represent \( \text{ABAC}_\alpha \) and \( \text{UCON}_\alpha^{\text{finite pre}_\alpha} \) models as \( \text{ABAC}_\alpha \) and \( \text{UCON}_\alpha^{\text{finite pre}_\alpha} \) schemes
2. Construct a mapping \( \sigma^{\text{ABAC}_\alpha} \) that maps \( \text{ABAC}_\alpha \) to \( \text{UCON}_\alpha^{\text{finite pre}_\alpha} \)
3. Prove that \( \sigma^{\text{ABAC}_\alpha} \) mapping from \( \text{ABAC}_\alpha \) to \( \text{UCON}_\alpha^{\text{finite pre}_\alpha} \) satisfies the following two requirements for state matching reduction:
   (a) for every state \( \gamma^{\text{ABAC}_\alpha} \) reachable from \( \gamma^{\text{ABAC}_\alpha} \) under the mapping \( \sigma^{\text{ABAC}_\alpha} \) there exists a reachable state in \( \text{UCON}_\alpha^{\text{finite pre}_\alpha} \) scheme that is equivalent (answers all the queries in the same way)
   (b) for every state \( \gamma^{\text{UCON}_\alpha^{\text{finite pre}_\alpha}} \) reachable from \( \gamma^{\text{UCON}_\alpha^{\text{finite pre}_\alpha}} \) under the mapping \( \sigma^{\text{ABAC}_\alpha} \) there exists a reachable state in \( \text{ABAC}_\alpha \) scheme that is equivalent (answers all the queries in the same way)

5.1 ABAC_\alpha Scheme
An \( \text{ABAC}_\alpha \) scheme consists of \( \langle \Gamma^{\text{ABAC}_\alpha}, \Psi^{\text{ABAC}_\alpha}, Q^{\text{ABAC}_\alpha}, \vdash^{\text{ABAC}_\alpha} \rangle \). Where

- \( \Gamma^{\text{ABAC}_\alpha} \) is the set of all states. Where each state \( \gamma^{\text{ABAC}_\alpha} \in \Gamma^{\text{ABAC}_\alpha} \) is characterized by (\( U, S, O, \), UA, SA, OA, uavtf, savtf, oavtf, P, SubCreator)
where \( U, S, O, \) are set of users, subjects objects respectively in state \( \gamma \).

- \( \Psi^{\text{ABAC}_\alpha} \) is the set of state transition rules which are all \( \text{ABAC}_\alpha \) operations defined in Table 3.

- \( Q^{\text{ABAC}_\alpha} \) is the set of queries of type:
  1. Authorization\((s, o)\) for \( p \in P^{\text{ABAC}_\alpha}, s \in S^{\text{ABAC}_\alpha}, o \in O^{\text{ABAC}_\alpha} \).
  2. ConstrSub\((u, s, savt)\) for \( u \in U^{\text{ABAC}_\alpha}, s \notin S^{\text{ABAC}_\alpha}, savt \in SAVT^{\text{ABAC}_\alpha} \).
  3. ConstrSubMod\((u, s, savt)\) for \( u \in U^{\text{ABAC}_\alpha}, s \in S^{\text{ABAC}_\alpha}, savt \in SAVT^{\text{ABAC}_\alpha} \).
  4. ConstrObj\((s, o, oavt)\) for \( s \in S^{\text{ABAC}_\alpha}, o \notin O^{\text{ABAC}_\alpha}, oavt \in OAVT^{\text{ABAC}_\alpha} \).
  5. ConstrObjMod\((s, o, oavt)\) for \( s \in S^{\text{ABAC}_\alpha}, o \in O^{\text{ABAC}_\alpha}, oavt \in OAVT^{\text{ABAC}_\alpha} \).

- Entailment \( \vdash \) specifies that given a state \( \gamma \in \Gamma^{\text{ABAC}_\alpha} \) and a query \( q \in Q^{\text{ABAC}_\alpha} \), \( \gamma \vdash q \) if and only if \( q \) returns true in state \( \gamma \).
5.2 UCON
finite
preA Scheme

An UCON
finite
preA scheme consists of \( \{ \Gamma^\text{UCON}_\text{preA}^\text{finite}, \Psi^\text{UCON}_\text{preA}^\text{finite}, Q^\text{UCON}_\text{preA}^\text{finite}, \vdash^\text{UCON}_\text{preA}^\text{finite} \} \), as follows.

- \( \Gamma^\text{UCON}_\text{preA}^\text{finite} \) is the set of all states. Where each state \( \gamma^\text{UCON}_\text{preA}^\text{finite} \in \Gamma^\text{UCON}_\text{preA}^\text{finite} \) is characterized by \( \langle \text{OS}, \text{UR}, \text{ATT}, \text{AVT}, \text{avtf} \rangle \). Here \( \text{OS} \) is the object schema in state \( \gamma \).
- \( \Psi^\text{UCON}_\text{preA}^\text{finite} \) is a set of state transition rules which are the set of creating and non-creating commands of UCON
finite
preA defined in Table 6 and 7.
- \( Q^\text{ABAC}_\alpha \) is the set of queries and of following types:
  1. CheckPCNCR(\( ucr, avtf(s), avtf(o), avtf(s), avtf(o) \)) for \( ucr \in \text{UC}, r \in \text{UR}, s \) and \( o \) are UCON
finite
preA objects.
  2. Whether CheckPCCR(\( ucr, avtf(s), avtf(s), avtf(s), avtf(s) \)) for \( ucr \in \text{UC}, r \in \text{UR}, s \) is an UCON
finite
preA object.
- Entailment \( \vdash \) specifies that given a state \( \gamma \in \Gamma^\text{UCON}_\text{preA}^\text{finite} \) and a query \( q \in Q^\text{UCON}_\text{preA}^\text{finite}, \gamma \vdash q \) if and only if \( q \) returns true in state \( \gamma \).

5.3 Mapping from ABAC\( \alpha \) to UCON
finite
preA (\( \sigma^\text{ABAC}_\alpha \))

- Mapping of \( \Gamma^\text{ABAC}_\alpha \) to \( \Gamma^\text{UCON}_\text{preA}^\text{finite} \)
  - Mapping Object Schema(\( \text{OS} \)), ATT and UR is provided in Table 5
  - Mapping of \( \Psi^\text{ABAC}_\alpha \) to \( \Psi^\text{UCON}_\text{preA}^\text{finite} \)
    - \( \sigma(\text{Access}_p) = \text{Access}_r \) where \( r = p \).
    - \( \sigma(\text{CreateSubject}(u, s, savt)) = \text{CreateSubject}_{ij}(s, o), i = \text{uavtf}(u) \) and \( j = \text{savt} \).
    - \( \sigma(\text{DeleteSubject}(u, s)) = \text{DeleteSubject}_{ij}(s, o). \)
    - \( \sigma(\text{ModifySubjectAtt}(u, s, savt)) = \text{ModifySubjectAtt}_{ij}(s, o), i = \text{uavtf}(u) \) and \( j = \text{savtf}(s) \) and \( k = \text{savt} \).
    - \( \sigma(\text{CreateObject}(s, o, oavt)) = \text{CreateObject}_{ij}(s, o), i = \text{savtf}(s) \) and \( j = \text{oavtf}(o) \) and \( k = \text{oavt} \).
- Mapping of \( Q^\text{ABAC}_\alpha \) to \( Q^\text{UCON}_\text{preA}^\text{finite} \) is provided below
  - \( \sigma(\text{Authorization}_p(s, o)) = \text{CheckPCNCR}(\text{Access}_p, avtf(s), avtf(o), avtf(s), avtf(o)). \)
  - \( \sigma(\text{ConstrSub}(u, s, savt)) = \text{CheckPCCR}(\text{CreateSubject}_{ij}, avtf(s), o, avtf(s), \langle \text{subject, NULL, u, false, NULL, NULL, NULL, NULL, NULL, NULL, NULL} \rangle) \) where \( i = \text{uavtf}(u) \) and \( j = \text{savt} \).
  - \( \sigma(\text{ConstrSubMod}(u, s, savt)) = \text{CheckPCCR}(\text{ModifySubjectAtt}_{ij}, avtf(s), o, avtf(s), \langle \text{savtf, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL} \rangle) \) where \( i = \text{uavtf}(u) \) and \( j = \text{savt} \) and \( k = \text{savt} \).
  - \( \sigma(\text{ConstrObj}(s, o, oavt)) = \text{CheckPCCR}(\text{CreateObject}_{ij}, avtf(s), o, avtf(s), \langle \text{object, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL} \rangle) \) where \( i = \text{savtf}(s) \) and \( j = \text{oavtf}(o) \) and \( k = \text{oavt} \).
  - \( \sigma(\text{ConstrObjMod}(s, o, oavt)) = \text{CheckPCCR}(\text{ModifyObjectAtt}_{ij}, avtf(s), o, avtf(s), \langle \text{oavtf, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL} \rangle) \) where \( i = \text{savtf}(s) \) and \( j = \text{oavtf}(o) \) and \( k = \text{oavt} \).
5.4 Proof that $\sigma^{\text{ABAC}_\alpha}$ is State-Matching

The proof that the mapping provided above is a state matching reduction is lengthy and tedious. Here we present an outline of the main argument.

**Lemma 1.** $\sigma^{\text{ABAC}_\alpha}$ satisfies assertion 1 of the state matching reduction of definition 5.

**Proof.** (Sketch): Assertion 1 requires that, for every $\gamma^{\text{ABAC}_\alpha} \in F^{\text{ABAC}_\alpha}$ and every $\psi^{\text{ABAC}_\alpha} \in \Psi^{\text{ABAC}_\alpha}$, $\langle \gamma^{\text{ABAC}_\alpha}, \psi^{\text{ABAC}_\alpha} \rangle = \sigma' (\langle \gamma^{\text{ABAC}_\alpha}, \psi^{\text{ABAC}_\alpha} \rangle)$ has the following property:

For every $\gamma_1^{\text{ABAC}_\alpha}$ in scheme $\text{ABAC}_\alpha$ such that $\gamma^{\text{ABAC}_\alpha} \xrightarrow{\psi^{\text{ABAC}_\alpha}} \gamma_1^{\text{ABAC}_\alpha}$,

1. For every query $q^{\text{ABAC}_\alpha} \in Q^{\text{ABAC}_\alpha}$, $\gamma_1^{\text{ABAC}_\alpha} \vdash \text{ABAC}_\alpha q^{\text{ABAC}_\alpha}$ if and only if $\gamma^{\text{ABAC}_\alpha} \vdash \text{ABAC}_\alpha q^{\text{ABAC}_\alpha}$.

2. For every query $q^{\text{ABAC}_\alpha} \in Q^{\text{ABAC}_\alpha}$, $\gamma_1^{\text{ABAC}_\alpha} \vdash \text{ABAC}_\alpha q^{\text{ABAC}_\alpha}$ if and only if $\gamma^{\text{ABAC}_\alpha} \vdash \text{ABAC}_\alpha q^{\text{ABAC}_\alpha}$.

The proof is by induction on number of steps $n$ in $\gamma^{\text{ABAC}_\alpha} \xrightarrow{\psi^{\text{ABAC}_\alpha}} \gamma_1^{\text{ABAC}_\alpha}$.

**Lemma 2.** $\sigma^{\text{ABAC}_\alpha}$ satisfies assertion 2 of the state matching reduction of definition 5.

**Proof.** (Sketch): Assertion 2 requires that, for every $\gamma^{\text{ABAC}_\alpha} \in F^{\text{ABAC}_\alpha}$ and every $\psi^{\text{ABAC}_\alpha} \in \Psi^{\text{ABAC}_\alpha}$, $\langle \gamma^{\text{ABAC}_\alpha}, \psi^{\text{ABAC}_\alpha} \rangle = \sigma' (\langle \gamma^{\text{ABAC}_\alpha}, \psi^{\text{ABAC}_\alpha} \rangle)$ has the following property:

For every $\gamma_1^{\text{ABAC}_\alpha}$ in scheme $\text{ABAC}_\alpha$ such that $\gamma^{\text{ABAC}_\alpha} \xrightarrow{\psi^{\text{ABAC}_\alpha}} \gamma_1^{\text{ABAC}_\alpha}$,

1. $\gamma_1^{\text{ABAC}_\alpha} \vdash \text{ABAC}_\alpha q^{\text{ABAC}_\alpha}$.

2. For every query $q^{\text{ABAC}_\alpha} \in Q^{\text{ABAC}_\alpha}$, $\gamma_1^{\text{ABAC}_\alpha} \vdash \text{ABAC}_\alpha q^{\text{ABAC}_\alpha}$ if and only if $\gamma^{\text{ABAC}_\alpha} \vdash \text{ABAC}_\alpha q^{\text{ABAC}_\alpha}$.

The proof is by induction on number of steps $n$ in $\gamma^{\text{ABAC}_\alpha} \xrightarrow{\psi^{\text{ABAC}_\alpha}} \gamma_1^{\text{ABAC}_\alpha}$.
Theorem 1. $\sigma^{ABAC_\alpha}$ is a state matching reduction.

Proof. Lemma 1 shows that $\sigma^{ABAC_\alpha}$ satisfies assertion 1 of definition 5 and Lemma 2 shows that $\sigma^{ABAC_\alpha}$ satisfies assertion 2 of definition 5. According to the definition 5, $\sigma^{ABAC_\alpha}$ is a state matching reduction.

Theorem 2. Safety of $ABAC_\alpha$ is decidable.

Proof. Safety of $UCON_{\text{finite preA}}$ is decidable [7]. Theorem 1 proved there exists a state matching reduction from $ABAC_\alpha$ to $UCON_{\text{finite preA}}$. A state matching reduction preserves security properties [9] including safety.

6 Conclusion

This paper gives a state matching reduction from $ABAC_\alpha$ to $UCON_{\text{finite preA}}$. Safety of $UCON_{\text{finite preA}}$ is decidable [7] and state matching reduction preserves security properties including safety [9]. It follows that safety of $ABAC_\alpha$ is decidable.

Acknowledgments. This research is partially supported by NSF Grants CNS-1111925, CNS-1423481, CNS-1538418, and DoD ARL Grant W911NF-15-1-0518.

References