Safety Decidability for Pre-Authorization Usage Control with Identifier Attribute Domains

P. V. Rajkumar, *Member, IEEE* and Ravi Sandhu, *Fellow, IEEE*

Abstract—Safety analysis is a fundamental problem in authorization models. Safety decidable models provide theoretical foundations for decentralized security administration. Attributes of objects are central to usage control authorization models. It has previously been shown that inclusion of a single infinite attribute leads to undecidable safety, even without any creation of objects. Therefore unrestricted inclusion of infinite attributes is not possible in a safety decidable model. On the other hand, it has recently been shown that the safety problem for the pre-authorization usage control sub-model with finite attribute domains, called $PreUCON_A^{finite}$, is decidable even with unbounded object creation. A major limitation of finite attributes is the inability to link objects through attribute values in presence of unbounded object creation (since attributes that reference other objects must be infinite in this case). It would be desirable to have safety-decidable attribute-based models which include both finite and infinite attributes (necessarily with some restrictions). This paper develops a pre-authorization usage control sub-model, called $PreUCON_A^{id}$, with attribute domains solely comprised of infinite object identifiers with considerable restrictions on how these attributes can be updated. Safety decidability for $PreUCON_A^{id}$ is proved by defining the notion of ω-equivalent usage configurations, and showing that the reachable set of ω-equivalent usage configurations is computable and can be used to answer safety questions. The utility of such models in practice is illustrated by means of an example. The paper further shows that addition of even a single finite domain attribute to $PreUCON_A^{id}$ results in undecidable safety. These results indicate that combining finite and infinite attributes in a safety decidable model is a challenging task, which will likely require carefully crafted restrictions on updates to these attributes. The formulation of such a model remains an important open question.

Index Terms—Security, access control, usage control, authorization, and safety analysis

1 Introduction

Usage control is an unified authorization system which supports wide variety of security policies besides traditional access control [1]. Safety decidability is one of the fundamental requirements for decentralizing and automating the administration of authorization systems. In particular, it is a basic requirement for development of policy analysis tools for the system administrators, to check if the given set of policies and the initial configuration can give out an unintended access right in any of the future states. Such checking is called safety analysis and is known to be undecidable in general for the pre-authorization usage control model [2], hereafter called $PreUCON_A$. Therefore, safety checking of $PreUCON_A$ cannot be automated in its full generality and its safety decidable sub-models necessarily have restrictions on the attributes and update functions.

Attributes of objects along with authorization predicates constitute the core of $PreUCON_A$. $PreUCON_A$ subjects are considered as a subset of objects. Attributes represent the security relevant features of objects in the system. Authorization predicates defined over the attributes represent the authorization security requirements. Usage rights are the security sensitive functions of the system whose executions need to be guarded with authorization predicates. The domain of an object's attribute specifies the set from which that attribute can take a value. A domain can be finite or infinite, so accordingly we call each attribute as a finite domain or infinite domain. Both finite and infinite domain attributes are useful in expressing authorization policies.

In the $PreUCON_A$ model, a subject can execute a usage right over an object if the attribute values of the subject, object pair satisfy the corresponding authorization predicates. Execution of usage rights can dynamically create new objects in the system, as well as delete existing objects and modify attribute values. These features of the model can express security policies that are meant to prevent unauthorized creation, deletion, and usage of files, documents, and processes. Unbounded execution of object-creating commands can bring potentially infinite number of objects in the system. Infinite domain attributes are necessary to express certain authorization requirements, in particular, systems with infinite number of objects. However, safety analysis of $PreUCON_A$ with arbitrary infinite domain attributes, even without object creation, is undecidable [3]. Therefore, to achieve safety decidability the choice of the infinite domain and the update operations need to be restricted.

• P. V. Rajkumar is with the Computer Information Systems and Analytics, University of Central Missouri, Warrensburg, MO 64093. E-mail: rajkumarpv@gmail.com.
• R. Sandhu is with the Computer Science, Institute for Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249. E-mail: ravi.sandhu@utsa.edu.

Manuscript received 16 Aug. 2017; revised 6 Feb. 2018; accepted 8 May 2018. Date of publication 23 May 2018; date of current version 13 May 2020. (Corresponding author: P. V. Rajkumar.) Digital Object Identifier no. 10.1109/TDSC.2018.2839745

1545-5971 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

In prior work, it has been shown that safety is decidable for $PreUCON_A$ with finite attribute domains [4] without any restrictions on object creation or on update operations. We call this the $PreUCON_A^{finite}$ model. It can express authorization requirements with finite domain attributes such as types, security labels and consumable rights.

A subtle but commonly used security attribute in information systems is object identifiers. Uniqueness of such identifiers play a vital role in expressing certain useful security policies. A common use of object identifiers is to establish relationships amongst objects such as parent and child. Another common example arises in dynamic separation of duties, such as a security policy which states that an employee who created a check must be different from the employee who approves the check. These cases can be easily enforced by utilizing attributes whose values come from the domain of object identifiers. In realistic systems this domain is infinite since, in principle, an unbounded number of objects can be created. In general, identities of files and their creators, identities of database records and tables, and unique document numbers in enterprise resource planning system are examples of important security attributes that require infinite domains.

In this paper, we present a safety decidability proof for a $PreUCON_A$ sub-model with infinite identifier domain attributes with constrained update operations, which we call $PreUCON_A^{id}$. In our earlier work on safety decidability for $PreUCON_A^{finite}$ [4], we used the notion of *protection tuples* and their *equivalence*. Protection tuples are the current values in the attributes of the objects in the system. In a given state, the objects with same protection tuples can be grouped as one equivalence class. Due to finite domain attributes, there are only a finite number of equivalence classes. This property was exploited in the decidability proof of $PreUCON_A^{finite}$. With infinite domain attributes this property no longer holds as the unbounded object creation would lead to potentially infinite number of non-equivalent protection tuples. Therefore, the safety decision procedure developed for $PreUCON_A^{finite}$ [4] may not work for certain instances of $PreUCON_A^{id}$. In this work, we develop a safety decision procedure for $PreUCON_A^{id}$ with infinite domain attributes that tracks changes in predicates instead of changes in attribute values. Specifically, we define the notion of ω-equivalence with respect to authorization predicates and show that the set of ω-equivalent authorization predicates that can become true is computable, and can be used to answer the safety question.

$PreUCON_A^{id}$ has its own limitations compared to general $PreUCON_A$ as it has restrictions on attribute updates, does not support finite domain attributes, and lacks arithmetic operations. A $PreUCON_A$ sub-model with both identifier attributes and finite attributes will be more expressive and desirable. Unfortunately, directly combining $PreUCON_A^{id}$ with $PreUCON_A^{finite}$ results in undecidable safety as shown in this paper. A challanging open problem arising from this undecidability is to formulate safety decidable $PreUCON_A$ sub-models that combine both finite and infinite attribute domains with reasonable restrictions.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 reviews the undecidability of safety in $PreUCON_A$ with arbitrary infinite domain attributes [3]. Section 4 presents the usage control authorization model with infinite domain identifier attributes newly developed in this paper, viz., $PreUCON_A^{id}$. Section 5 defines the notion of ω-equivalent protection tuples and shows that the set of reachable ω-equivalent protection tuples is computable and can be used to decide the safety problem. Section 6 presents an illustration of expressiveness of $PreUCON_A^{id}$. Section 7 shows that adding a single finite attribute to $PreUCON_A^{id}$ results in undecidable safety. It also discusses some open questions regarding safety analysis. Section 8 concludes the paper.

2 Related Work

Safety analysis has been actively studied in the context of both access control and usage control authorization models. The Access Matrix Model (ACM) [5] formulates protection systems as a two-dimensional matrix with the rows representing subjects, the columns representing objects and the cells holding a set of rights the subjects have over the objects. The model provides set of commands to change the matrix. Safety of the ACM model had been shown undecidable in general, and decidable if the commands are *non-creating* or *mono-operational*. The non-creating model cannot create any new objects in the system. The mono-operational model can create objects but (i) it cannot distinguish between two newly created objects, and further, (ii) it cannot express any relationship between the creator and the created objects. Recently, safety definitions and proofs presented in the foundation work [5] have been revisited and analyzed in [6]. Set of subjects in ACM is considered as a subset of objects. We also use this convention unless specified otherwise.

The Take-Grant authorization model [7] represents the protection system as a graph with objects as nodes and rights as labeled directed arcs. The model provides graph rewriting rules for execution of *take*, *grant*, *call*, *create*, and *delete* rights. The Take-Grant model's safety is decidable, however, its expressiveness is very limited as the rewrite rules are specified as a part of the model itself. Further, if a subject has a right then any subject which is connected to it in the graph can obtain the right without any constraint. The model has no features that can limit rights that can be acquired from one connected subject to another.

The Schematic Protection Model (SPM) [8], [9] introduced *security types* where every object in the model has a fixed type. The SPM model has two types of rights: *control* rights and *inert* rights. Execution of control rights can change the protection state whereas inert rights cannot. For example, create and delete are control rights while read and write are inert rights. Safety is decidable for the SPM model with acyclic creates. Safety remains decidable for the acyclic SPM with additional features such as (i) inclusion of conditional authorization [10], (ii) revocation of rights [11], and (iii) multi-parent object creation [12].

Typed Access Matrix (TAM) [13] model formulates an access matrix with typed subjects and typed objects. An object's type is decided at the time of its creation, thereafter type cannot be changed. The TAM authorization model with acyclic object creation has decidable safety. Safety decidable TAM combines features in safety decidable fragments of SPM with ACM. Dynamic Typed Access Matrix (DTAM) [14]

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

model provides features for changing object's type after its creation but within a fixed finite domain. Safety of DTAM is decidable if its object creation is acyclic. Safety decidable TAM and DTAM models do not support consumable, regenerative rights, and cyclic creation policies.

Role Based Access Control (RBAC) [15], [16] model associates rights to roles and when a user is assigned to a role he gets the rights that are associated with the role. The RBAC model closely reflects role structured authorization policies within organizations. The RBAC model has been extended to support temporal [17], [18] and location [19] based security features. Administrative models [20], [21] of RBAC are specifically designed to systematically authorize the security administrators to change the role-permission, user-role assignments and role hierarchies. Execution of such administrative changes may make the system unsafe.

Large information systems in organizations like banks have thousands of roles [22] and manual administration of systems with such large number of roles is a difficult task. Automated safety analysis of RBAC has been widely studied [23], [24], [25] and results from other domains like program analysis [26], logic [27], and trust management [28], [29] have been applied for the RBAC safety problem. A detailed study on security analysis of RBAC is given in [29]. Further, model checking based security analysis tool, called Mohawk, has been developed for automatically finding errors in RBAC policies [30], [31]. Recently, the safety problem in temporal administrative RBAC has been mapped to the safety problem in Administrative RBAC [32]. Thereby, Mohawk has been shown to be applicable for security analysis of temporal administrative RBAC policies. Safety problem in Administrative RBAC models concern analysis of user-role assignment, role-permission assignment, and role-role assignment of RBAC models. Whereas the usage control authorization model has been designed to support broader applications than RBAC [1].

The safety problem in the general $PreUCON_A$ model has been shown to be undecidable [2]. Safety is shown to be decidable for two sub-models of $PreUCON_A$ in [2]. First sub-model has finite attribute value domains without creating commands. Second sub-model supports create commands and its safety is decidable if the usage control commands meet three criteria: (i) the *attribute create graph* is acyclic, (ii) the *attribute update graph* has no cycle containing create-parent attribute tuples, and (iii) in creating commands both child and parent objects' attributes are updated. The safety decidable model in [2] cannot express cyclic creation policies and the model supports creation of only a finite number of new objects.

The safety problem in a sub-model of the $OnUCON_A$ ongoing authorization model without object creation commands, has been shown decidable in [33]. Safety decidable model in [33] cannot support creation of any new subjects in the system. An alternative approach to safety analysis is to map the safety problem into a satisfiability problem in logical theories, whereby the decidability results in a specific logical theory can be used to answer the safety question. Safety problem in a sub-model of the $OnUCON_A$ has been mapped to a decidable fragment of many-sorted logic [34]. Safety decidable model in [34] can create new passive objects like files but cannot create new active subjects like processes in the system.

Recently, we have shown that the $PreUCON_A$ sub-model with finite attribute value domains, called $PreUCON_A^{finite}$, has decidable safety property [4] without any restrictions on create commands or update commands. $PreUCON_A^{finite}$ can express acyclic policies and support unbounded creation of new objects. The model can express *uniform security policies* such as (i) "no student is permitted to *change his advisor* more than twice", but it cannot express *individualistic security policies* such as (ii) "a faculty advisor of a student can *approve* project team if he brings another student as his team member". The first policy can be enforced using $PreUCON_A^{finite}$. However, the second policy cannot be correctly enforced with finite attribute domains alone due to potentially infinite number of students.

First policy can be expressed with two finite attribute domains, say *type:{student, faculty}* and *count:{0, 1, 2}*. A usage control command can permit the *changeadvisor* right for s_1 over s_2 if $s_1.type = faculty$, $s_2.type = student$ and $s_2.count > 0$. The *count* attribute is initialized with 2 and each execution of *changeadvisor* decrements its value. The second policy requires four attributes, say *faculty_id*, *advisor_id*, *student_id*, and *team_member_id*. A usage control command can permit the $approve$ right for s_1 over s_2 if $s_1.faculty\_id = s_2.advisor\_id$ and $s_2.student\_id \neq s_2.team\_member\_id$. The attributes $advisor\_id$ and $team\_member\_id$ are the identifiers of the student's advisor and his project teammate, respectively. Correct enforcement of this policy requires infinite value domains for at least the *student_id* and the *team_member_id* attributes. Suppose we use the finite domain $\{x, y\}$ for these two id attributes. Then there can be two sets of students in the system: one set with $student\_id = x$ and another with $student\_id = y$. A student who chooses a team member with the same id will be denied approval which is incorrect. This problem will persist for any choice of finite domain values for these two id attributes.

3 Safety of $PreUCON_A$ with Arbitrary Infinite Domain Attributes is Undecidable

Undecidability of safety in the general $PreUCON_A$ model has been previously shown [2]. In this section, we briefly describe a proof of undecidable safety for a sub-model of $PreUCON_A$ with an arbitrary infinite domain attribute and without create commands. This result was first proved in [3]. It is included here for completeness.

The proof demonstrates that an arbitrary instance of Post Correspondence Problem (PCP) can be mapped to the safety problem in $PreUCON_A$ with an infinite domain attribute. Let the objects in $PreUCON_A$ have an arbitrary infinite domain attribute, say str, which is initialized to the empty string. The initial configuration of the model has two objects s and o. For each pair (str_1, str_2) of strings in PCP, we can construct a usage control command that appends the first string str_1 to the subject's str attribute and the second string str_2 to the object's str attribute. An additional usage control command is included to grant an unsafe right r if the values of the subject's str attribute and the object's str attribute are equal and non-empty. The subject s can get the unsafe usage right r if and only if the arbitrary instance of the PCP problem has a solution. Thereby, the undecidability of safety in the $PreUCON_A$ sub-model follows from the undecidability of the PCP problem. Additional proof details are given in

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2018.2839745.

An important observation from this undecidability result is that the choice of attribute domain and the type of update functions are important aspects that need to be considered while constructing safety decidable $PreUCON_A$ sub-models with infinite domains. Whereas the $PreUCON_A^{finite}$ with arbitrary finite domain attributes and unconstrained update functions has decidable safety, even with unbounded object creation [4].

$PreUCON_A^{finite}$ cannot express certain important relationships like parent and child between the creater subject and the created object. Such linking relationships play a very significant role in enforcing many commonly used security policies. For example policies like *'a parent process can give a subset of its privileges to its child process then eventually revoke the privileges'* cannot be enforced without having the linking relationship between the parent process and the child process. The key element in establishing such relationships is the identifiers of the objects. Further, to establish the relationship between the objects, the update functions in the usage control commands must have the ability to store each others identifiers. The set of object identifiers in the systems with unbounded number of objects must be an infinite set, therefore, the domain of attributes which stores the object indentifiers must be an infinite set as well. In this work, we focus on constructing a safety decidable $PreUCON_A$ model which can express such important relationships between unbounded number of objects.

4 The $PreUCON_A^{id}$ Model

One of the challenges in constructing safety decidable models is to identify the right synergy between the predicates and update functions, such that the model is expressive enough for practical applications while maintaining decidability. Usage control authorization policies specify the authorization security requirements to execute usage rights as well as the consequence of executing the rights. In $PreUCON_A$ security requirements are specified as predicates and the consequences are specified as update operations. The updates may (i) create a new object, (ii) delete an existing object, and (iii) modify attribute values of the objects. We construct a sub-model of $PreUCON_A$ with identifier domain attributes, namely $PreUCON_A^{id}$, that can express important security policies that are not expressible in $PreUCON_A^{finite}$. We define the usage control authorization scheme in $PreUCON_A^{id}$ below, followed by an example.

4.1 Usage Control Authorization Scheme

Definition 4.1. *A usage control authorization scheme U_ω has three components as follows.*

(i) *an object schema OS_ω,*
(ii) *a set of usage rights $UR = \{r_1, r_2, \ldots, r_m\}$, and*
(iii) *a set of usage control commands $UC = \{UC_1, UC_2, \ldots, UC_{cn}\}$ (defined in Section 4.2).*

4.1.1 Object Schema

Security relevant attributes of objects in an information system and the domain of the attributes are specified in an object schema. A crucial feature of the $PreUCON_A^{id}$ model is to express various linking relationships between an unbounded population of objects, for which purpose we use the set of object identifiers in the system as the value domain of the attributes.

Definition 4.2. *The object schema OS_ω is of the form $[id : ID, a_2 : ID, a_3 : ID \ldots a_n : ID]$, where id is the object's own identifier and the remaining a_i's are the attributes that can store an* id *of any object in the system.*

The *id* attribute of each object in the system must have a unique value. Since there could be infinite number of objects, the domains of the *id* attribute must be an infinite set of unique identifiers. For the sake of convenience, we also use the name a_1 to refer the *id* attribute whenever we do not have to differentiate it from the rest of the attributes. We use ω in OS_ω to denote that all the attributes in the object schema are infinite domains attributes.

Definition 4.3. *The domain ID is defined as a countably infinite set of unique identifiers in the system.*

The choice of the set of values for identifiers is left specific to the application. In practical applications, the domain of one set of object identifiers may differ from that of another set. For example, in online course-ware applications, the course identifier attributes value domain $ID_c = \{\text{CS100-}S2016\text{-01}, CS104\text{-}F2016\text{-02}, \ldots\}$ is usually different from both the faculty identifier attribute's value domain $ID_f = \{\text{FCS000158, FMAT0045067, ...}\}$ and the student identifier attribute's value domain $ID_s = \{\text{BS201500018}, MS20160021, \ldots\}$. In order to simplify the presentation, the object schema in $PreUCON_A^{id}$ uses one generic infinite set *ID* as the value domain of all identifier attributes. Such a generic set may be taken as union of value domains of all identifier attributes in the application. For the above example, the domain of *id* may be the union of ID_c, ID_f and ID_s. Further, we also assume that there is a single object schema which uniformly applies to all the objects in the system [2], [4].

Attributes of objects are atomic valued and can hold an identifier of an object in the system. Values of an object's attributes are accessed using *dot*(.) operator, e.g., $o.x$ gives the value of object o's attribute x. An important and limiting restriction in $PreUCON_A^{id}$ is that once an object's attribute is assigned a value then it remains unchanged forever.

4.1.2 Usage Rights

Usage rights are the finite set of permissions defined over the objects. Usage control commands grant the rights. $PreUCON_A^{id}$ supports *creation* and *deletion* of objects beside uninterpreted application-specific usage rights. Execution of create brings a new object into the system. Create also assigns a unique value to the id attribute of newly created object while some of the object's attributes may remain unassigned. Such unassigned attributes are marked with the special symbol $\phi \notin ID$ and may get assigned with a value during execution of subsequent usage control commands.

4.2 Usage Control Commands

Usage control commands in U_ω enforce the authorization requirements and execute update operations before

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

permitting the execution of usage rights. A subject can execute a usage right over an object if the subject, object pair's attribute values satisfy the authorization predicates in a usage control command. There are three types of usage control commands, viz. non-creating, creating and deleting commands. Deleting commands can be ignored for the purpose of safety analysis [4] (see Appendix B, available in the online supplemental material).

4.2.1 Authorization Predicates

Authorization predicates are composition of specific set of atomic predicates. The most useful operations in the identifier domain are to check whether a value is assigned to an attribute and if it is assigned then to check whether it is equal to the value of another attribute. We define two types of atomic predicates to do such checking on attributes of objects, namely, α-predicates and β-predicates. In the definitions of α-predicates and β-predicates, the attribute name a_i is a symbolic placeholder used to denote an attribute of the object o_x including the id attribute.

Definition 4.4. *An α-predicate is of the form $o_x.a_i = \phi$. It evaluates to True if and only if the object o_x's attribute a_i is not yet assigned with a value from ID.*

The α-predicates are used to ensure that an attributes is assigned a value at most once. They also help in expressing certain consumable rights such as "a student can change his project advisor at most twice".

The β-predicates consist of three varieties, viz., β^1, β^2, and β^3.

Definition 4.5. *A β^1-predicate is of the form $o_x.a_i \neq \phi$. It evaluates to True if and only if the object o_x's attribute a_i is assigned with a value from ID.*

The β^1 predicates are used to make sure that the attributes values are non-empty before comparing them with other attribute values. The following two predicates are used to compare equality between attribute values.

Definition 4.6. *A β^2-predicate is of the form $o_x.a_i \doteq o_y.a_j$. It evaluates to True if and only if the following three conditions are satisfied: (i) $o_x.a_i \neq \phi$, (ii) $o_y.a_j \neq \phi$, and (iii) $o_x.a_i = o_y.a_j$.*

Definition 4.7. *A β^3-predicate is of the form $o_x.a_i \dot{\neq} o_y.a_j$. It evaluates to True if and only if the following three conditions are satisfied: (i) $o_x.a_i \neq \phi$, (ii) $o_y.a_j \neq \phi$, and (iii) $o_x.a_i \neq o_y.a_j$.*

The β-predicates are useful in expressing authorization requirements that compare equality between attribute values.

For the moment let us ignore the possibility of object deletion. If the truth value of an α-predicate for an object becomes *false* then it will remain false thereafter. Likewise, if the truth value of a β-predicate for a given subject, object pair becomes *true* then it remains *true* thereafter. Note that the β^2 and β^3 predicates are not opposites of each other. However, if one is *true* then the other must be *false*. It is possible for both to be false for a given subject, object pair. Once one of them becomes *true* for a given subject, object pair it must remain *true* thereafter.

4.2.2 Non-Creating Commands

The non-creating commands involve an existing subject, object pair. The structure of a non-creating command UC_i is as follows.

Command_Name$_r(s, o)$

$$\begin{aligned}
&\textbf{PreCondition:} && P_\alpha \wedge P_\beta\\
&\textbf{PreUpdate:} && s.a_{i_1} := f_{a_{i_1}}(s, o);\\
& && \dots\\
& && s.a_{i_p} := f_{a_{i_p}}(s, o);\\
& && o.a_{j_1} := f_{a_{j_1}}(s, o);\\
& && \dots\\
& && o.a_{j_q} := f_{a_{j_q}}(s, o);
\end{aligned}$$

In this command both s and o are input parameters. P_α and P_β are respectively, the α-predicate conjunct and the β-predicate conjunct defined over the attributes of the subject s and the object o. P_β should have at least two β-predicates $s.id \neq \phi$ and $o.id \neq \phi$ to ensure the existence of both s and o. P_α must include the sub-conjuncts $(s.a_{i_1} = \phi) \wedge \cdots \wedge (s.a_{i_p} = \phi)$ and $(o.a_{j_1} = \phi) \wedge \cdots \wedge (o.a_{j_q} = \phi)$ to check that these attributes are not yet assigned with any values. Values for these attributes are assigned from attributes of s, o which have already been assigned values. Each function f in the $preUpdate$ returns the value stored in a specific attribute of its inputs s and o. If the attribute values of s and o satisfies the predicate conjuncts then the command grants the right r and executes the $preUpdate$ operations. Otherwise, it denies the right and terminates without executing $preUpdate$.

4.2.3 Creating Commands

The structure of a creating command UC_i is as follows.

Command_Name$_{create}(s, o)$

$$\begin{aligned}
&\textbf{PreCondition:} && P_\alpha \wedge P_\beta\\
&\textbf{PreUpdate:} && create\ o;\\
& && s.a_{i_1} := f_{a_{i_1}}(s, o.id);\\
& && \dots\\
& && s.a_{i_p} := f_{a_{i_p}}(s, o.id);\\
& && o.a_{j_1} := f_{a_{j_1}}(s, o.id);\\
& && \dots\\
& && o.a_{j_q} := f_{a_{j_q}}(s, o.id);
\end{aligned}$$

In this command s is an input parameter and o is an output parameter. P_α and P_β are the α-predicate conjunct and the β-predicate conjunct respectively, defined over the attributes of the subject s. P_β should have at least one β-predicate $s.id \neq \phi$ to ensure the existence of subject s. P_α must include the sub-conjunct $(s.a_{i_1} = \phi) \wedge \cdots \wedge (s.a_{i_p} = \phi)$ to check that the attributes $s.a_{i_1} \dots s.a_{i_p}$ are not yet assigned any values. Values for these attributes as well as for some attributes of the newly created object o are assigned from attributes which are already having assigned values. If the attribute values of subject s satisfies the predicate conjuncts, then the command grants the $create$ right and executes the $preUpdate$ operations. Otherwise, the command denies the right and terminates without executing the $preUpdate$ part.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

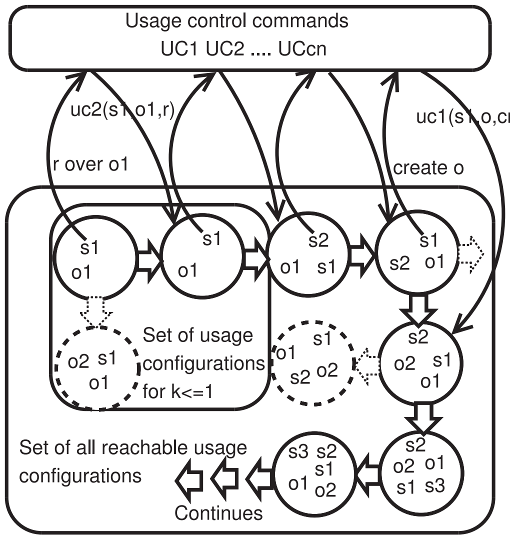

Fig. 1. Pre-authorization system.

The *create* operation in the *preUpdate* part brings a new object o into the usage control system. Further, the *create* operation assigns a unique object identifier value to $o.id$ and assigns ϕ to all other attributes of o. The object's identifier value is from the infinite domain ID and the value is never reused. Each function f_a in the *preUpdate* part returns a specific attribute's value from the attributes of its input s or the value of $o.id$. The Fig. 1 diagrammatically shows the relationship between the objects, usage rights, and usage control commands. Other concepts in the figure such as usage configurations and reachable usage configurations are explained in Sections 4.4, and 5.

4.3 Example Usage Control Authorization Scheme

The following toy game example illustrates components of a $PreUCON_A^{id}$ authorization scheme. The example scheme permits (i) players to red *mark* white balls created by others, (ii) players to *hit* balls red marked by themselves, and (iii) players to add new players and create white balls.

Example 4.8. The components of the usage control authorization scheme $U_{\omega\ game}$ are as follows.

(1) Object schema $OS_{\omega game}$ = [id:ID, player$_{id}$:ID, ball$_{id}$:ID, creator:ID, red:ID]
(2) Usage rights UR = {*mark, hit, addplayer, addball*}
(3) Usage control commands UC = {UC_1, UC_2, UC_3, UC_4} are as follows.

UC_1 : **Permit**$_{mark}(s, o)$

$$\textbf{PreCondition:} \quad s.player_{id} \neq \phi \wedge o.ball_{id} \neq \phi \wedge s.player_{id} \dot{\neq} o.creator \wedge o.red = \phi$$

$$\textbf{PreUpdate:} \quad o.red := s.player_{id};$$

UC_2: **Permit**$_{hit}(s, o)$

$$\textbf{PreCondition:} \quad s.player_{id} \neq \phi \wedge o.ball_{id} \neq \phi \wedge o.red \doteq s.player_{id};$$

UC_3: **Permit**$_{addplayer}(s, o)$

$$\textbf{PreCondition:} \quad s.player_{id} \neq \phi;$$
$$\textbf{PreUpdate:} \quad create\, o; \quad o.player_{id} := o.id;$$

UC_4: **Permit**$_{addball}(s, o)$

$$\textbf{PreCondition:} \quad s.player_{id} \neq \phi;$$
$$\textbf{PreUpdate:} \quad create\, o; \quad o.ball_{id} := o.id; \quad o.creator := s.player_{id};$$

The object schema $OS_{\omega\ game}$ has five attributes: (i) id is the default object identifier attribute, (ii) $player_{id}$ for storing a player's identity, (iii) $ball_{id}$ for storing a ball's identity, (iv) $creator$ for storing the creator's identity of a ball, and (v) red for remembering if a ball is marked red and which player marked it. In this scheme, the set of players are the subjects and the set of balls are the objects. The $player_{id}$ attribute is used in writing the security properties of the subjects and the remaining attributes are unused for subjects. Likewise, the $ball_{id}$, $creator$ and red attributes are used in writing the security properties of the objects and the $player_{id}$ attribute is unused for objects.

The usage rights *mark* and *hit* are protected using non-creating commands UC_1 and UC_2, respectively. The *addplayer* and *addball* rights are protected using creating commands UC_3 and UC_4, respectively. This scheme has the set of α-predicates $\{o.red = \phi\}$ and the set of β-predicates $\{s.player_{id} \neq \phi, o.ball_{id} \neq \phi, o.red \doteq s.player_{id}\}$ as preconditions. The create command UC_3 assigns a unique player identifier to the attribute $o.player_{id}$ and assigns ϕ to the remaining attributes of the newly created object o, which is a new player. Similarly, the create command UC_4 assigns a unique ball identifier to the attribute $o.ball_{id}$, records the ball's creaor in the $o.creator$ attribute and assigns ϕ to the remaining attributes of the newly created object o.

4.4 Usage Control Configuration

A state of the usage control authorization scheme U_ω is called as a usage configuration U_Ξ.

Definition 4.9. *A usage configuration U_Ξ is defined as the set of objects $\{o_1, o_2, \ldots, o_n\}$ along with the values stored in the attributes of each object o_i.*

Example 4.10. An example usage configuration for the usage control authorization scheme $U_{\omega\ game}$ of Example 4.8 with two players, one white ball, and one red marked ball is given as follows

$$\begin{aligned} U_{\Xi t} = \{ & s_1.player_{id} = PX021, s_1.ball_{id} = \phi, \\ & s_1.creator_{id} = \phi, s_1.red = \phi; \\ & s_2.player_{id} = PX756, s_2.ball_{id} = \phi, \\ & s_2.creator_{id} = \phi, s_2.red = \phi; \\ & o_1.player_{id} = \phi, o_1.ball_{id} = BI213, \\ & o_1.creator_{id} = PX021, o_1.red = \phi; \\ & o_2.player_{id} = \phi, o_2.ball_{id} = BI855, \\ & o_2.creator_{id} = PX756, o_2.red = PX021; \}. \end{aligned}$$

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

The initial configuration of the usage control authorization scheme U_ω is symbolically denoted as $U_{\Xi init}$. Likewise $U_{\Xi t}$ denotes the usage configuration t. A usage control system begins with a specified initial configuration $U_{\Xi init}$ and thereafter it evolves as the subjects execute usage control commands. The set of all possible configurations of the scheme is denoted as the set $\widehat{U_{\Xi\omega}}$. This set could be potentially infinite as the scheme supports unbounded executions of creating commands. Appendix C, available in the online supplemental material, gives an example of unbounded creation in context of Example 4.8. An execution of usage control command UC_i with an actual subject s and an actual object o is called a command instance, denoted as $uc_i(s, o)$. Subject s must exist prior to execution of $uc_i(s, o)$. For a non-creating command object o must also exist prior to execution of $uc_i(s, o)$, whereas for a creating command o is newly created.

4.5 Safety Question

Definition 4.11. *Safety question is defined as given a usage control authorization scheme U_ω and its initial configuration $U_{\Xi init}$, can the subject $s_1 \in U_{\Xi init}$ obtain the usage right r over the object $o_1 \in U_{\Xi init}$? A safety question is written as $\Upsilon_{(s_1,o_1,r)}$.*

A safety decision procedure (SDP) takes as input U_ω, $U_{\Xi init}$ and $\Upsilon_{(s_1,o_1,r)}$. It returns yes if subject s_1 can acquire usage right r for o_1 and no otherwise. In order to make the technical presentation easier, we use a more general version of the safety question called $\Upsilon_{(r)}$.

Definition 4.12. *The safety question $\Upsilon_{(r)}$ is defined as given a U_ω and a $U_{\Xi init}$, can any subject obtain the usage right r over any object?*

If there is an SDP to answer the safety question $\Upsilon_{(r)}$ we can use the same SDP to answer the $\Upsilon_{(s_1,o_1,r)}$ as shown in Appendix D, available in the online supplemental material.

5 SAFETY IN $PreUCON^{id}_{A}$ IS DECIDABLE

In this section, we present a proof of safety decidability for $PreUCON^{id}_{A}$. Let $P_{\alpha 1} \wedge P_{\beta 1}, P_{\alpha 2} \wedge P_{\beta 2} \cdots P_{\alpha cn} \wedge P_{\beta cn}$ denote the authorization predicates in precondition part of the usage control commands $UC_1, UC_2 \ldots UC_{cn}$, respectively. We also understand P_i to mean $P_{\alpha i} \wedge P_{\beta i}$ for UC_i.

In a usage configuration multiple subject object pairs may satisfy the preconditions of a usage control command and a pair may satisfy preconditions of multiple commands as well. Subjects can execute the commands at their own discretion without any predefined order. Execution of a command can bring a new object into the system as well as update the object's attribute values. The new attribute values may enable additional commands and the execution of additional commands can bring new objects as well as update the attribute values. This cycle may continue without bound and can create potentially infinite number of new configurations. Therefore, tracking the changes in usage configuration may not terminate. Instead, our safety decision procedure tracks the changes in the set of satisfiable predicate sub-conjuncts of $P_{\alpha 1} \wedge P_{\beta 1}, P_{\alpha 2} \wedge P_{\beta 2} \ldots P_{\alpha cn} \wedge P_{\beta cn}$.

The crux of the safety decidability proof lies in showing that there exists a subset of reachable usage configurations which can satisfy the maximal set of sub-conjuncts that are satisfiable in the set of all reachable configurations. We first define the sub-conjuncts and their maximal set for the given usage control authorization scheme U_ω and the initial configuration $U_{\Xi init}$. The symbols used in developing our proof are summarized in Table 1.

TABLE 1
Summary of Symbols

Symbols	Description
U_ω	The usage control authorization scheme
U_Ξ	A usage configuration for given U_ω
$U_{\Xi init}$	An initial configuration for given U_ω
$U_{\Xi x}$	Usage configuration in state x for given $U_{\Xi init}$ and U_ω
$\widehat{U_{\Xi p}}$	A set of usage configurations
$\widehat{U_{\Xi\omega}}$	The set of all usage configurations for given $U_{\Xi init}$ and U_ω
UC_i	Usage control command
uc_i	Instance of the specific usage control command UC_i
$\dot{uc}$	Instance of any usage control command
$\Upsilon_{(s_1,o_1,r)}$	Safety question: can $s_1 \in U_{\Xi init}$ gain r for $o_1 \in U_{\Xi init}$?
$\Upsilon_{(r)}$	Safety question: can any subject gain r for any object?
α-predicate	An atomic predicate of the form $o_x.a_i = \phi$
β-predicate	An atomic predicate of the form $o_x.a_i \neq \phi$, $o_x.a_i \doteq o_y.a_j$, or $o_x.a_i \not\doteq o_y.a_j$
p_i	An atomic α-predicate or an atomic β-predicate
$P_{\alpha i}$	α-predicate conjunct in the precondition of usage control command UC_i
$P_{\beta i}$	β-predicate conjunct in the precondition of usage control command UC_i
P_i	$P_{\alpha i} \wedge P_{\beta i}$, the precondition of usage control command UC_i
$\|P_{\alpha i}\|$	Number of atomic predicates in $P_{\alpha i}$
$\|P_{\beta i}\|$	Number of atomic predicates in $P_{\beta i}$
$\widehat{P_i}$	Set of all subconjuncts of P_i
$\widehat{P_\omega}$	Set of all subconjuncts of usage control authorization scheme U_ω
$\|\widehat{P_i}\|$	Number of predicates in the set $\widehat{P_i}$
ω_{max}	$\omega_{max} \leq \widehat{P_\omega}\|$, maximum number of predicate conjuncts in U_ω
$\widehat{f_{\Xi\omega}}$	$\widehat{f_{\Xi\omega}} : \widehat{U_{\Xi\omega}} \to \widehat{P_\omega}$, mapping from a set of usage configurations to a set of predicates in $\widehat{P_\omega}$
$\widehat{U_{\Xi i}} \approx_\omega \widehat{U_{\Xi j}}$	$\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi i}}) = \widehat{f_{\Xi\omega}}(\widehat{U_{\Xi j}})$
$\widehat{U_{\Xi init\uparrow 0}}$	$\{U_{\Xi init}\}$
$\widehat{U_{\Xi init\uparrow 1}}$	Set of usage configurations reachable from $U_{\Xi init}$ in 0 or 1 step
$\widehat{U_{\Xi init\uparrow k}}$	Set of usage configurations reachable from $U_{\Xi init}$ in 0 or up to k steps
$\widehat{U_{\Xi i\uparrow *}}$	Set of usage configurations reachable from $U_{\Xi i}$ in 0 or any number of steps

Definition 5.1. *The size of an α-predicate conjunct $P_{\alpha i}$ is defined as the number of atomic α-predicates in $P_{\alpha i}$ and is denoted as $|P_{\alpha i}|$. Likewise, $|P_{\beta i}|$ denotes the size of β-predicate conjunct $P_{\beta i}$ and $|P_i| = |P_{\alpha i}| + |P_{\beta i}|$.*

Definition 5.2. *The set of sub-conjuncts of P_i is defined as the union of all sub-conjuncts of $P_{\alpha i} \wedge P_{\beta i}$ of size 1 to $|P_{\alpha i}| + |P_{\beta i}|$ and is denoted as $\widehat{P_i}$.*

Example 5.3. The set of sub-conjuncts of the predicate in the precondition part of the usage control command UC_2 in the usage control authorization scheme $U_{\omega\ game}$ in the Example 4.8 is given as follows

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Formal Definition of Satisfiability of p^is

$(s,o) \models p_i$	$(s,o_\phi) \models p_i$
$p^1 : s.a_i = \phi$ $(s,o) \models p^1$, iff $s.v_i = \phi$.	$p^{11} : s.a_i = \phi$ $(s,o_\phi) \models p^{11}$, iff $s.v_i = \phi$.
$p^2 : o.a_j = \phi$ $(s,o) \models p^2$, iff $o.v_j = \phi$.	
$p^3 : s.a_i \neq \phi$ $(s,o) \models p^3$, iff $s.v_i \in ID_i$.	$p^{12} : s.a_i \neq \phi$ $(s,o_\phi) \models p^{12}$, iff $s.v_i \in ID_i$.
$p^4 : o.a_j \neq \phi$ $(s,o) \models p^4$, iff $o.v_j \in ID_j$.	
$p^5 : s.a_i \doteq s.a_j$ $(s,o) \models p^5$, iff $s.v_i \in ID_i$, $s.v_j \in ID_j$, and $s.v_i = s.v_j$.	$p^{13} : s.a_i \doteq s.a_j$ $(s,o_\phi) \models p^{13}$, iff $s.v_i \in ID_i$, $s.v_j \in ID_j$, and $s.v_i = s.v_j$.
$p^6 : o.a_i \doteq o.a_j$ $(s,o) \models p^6$, iff $o.v_i \in ID_i$, $o.v_j \in ID_j$, and $o.v_i = o.v_j$.	
$p^7 : s.a_i \doteq o.a_j$ $(s,o) \models p^7$, iff $s.v_i \in ID_i$, $o.v_j \in ID_j$, and $s.v_i = o.v_j$.	
$p^8 : s.a_i \not\doteq s.a_j$ $(s,o) \models p^8$, iff $s.v_i \in ID_i$, $s.v_j \in ID_j$, and $s.v_i \neq s.v_j$.	$p^{14} : s.a_i \not\doteq s.a_j$ $(s,o_\phi) \models p^{14}$, iff $s.v_i \notin ID_i$, $s.v_j \in ID_j$, and $s.v_i \neq s.v_j$.
$p^9 : o.a_i \not\doteq o.a_j$ $(s,o) \models p^9$, iff $o.v_i \in ID_i$, $o.v_j \in ID_j$, and $o.v_i \neq o.v_j$.	
$p^{10} : s.a_i \doteq o.a_j$ $(s,o) \models p^{10}$, iff $s.v_i \in ID_i$, $o.v_j \in ID_j$, and $s.v_i \neq o.v_j$.	

$$\begin{aligned}\widehat{P_2} = \{&s.player_{id} \neq \phi, o.ball_{id} \neq \phi, o.red \doteq s.player_{id},\\ &s.player_{id} \neq \phi \wedge o.ball_{id} \neq \phi,\\ &s.player_{id} \neq \phi \wedge o.red \doteq s.player_{id},\\ &o.ball_{id} \neq \phi \wedge o.red \doteq s.player_{id},\\ &s.player_{id} \neq \phi \wedge o.ball_{id} \neq \phi \wedge o.red \doteq s.player_{id}\}\end{aligned}$$

Definition 5.4. *The set of sub-conjuncts in the given usage control authorization scheme U_ω is defined as the union of sets of sub-conjuncts of all $\widehat{P_i}$'s. That is, $\widehat{P_\omega} = \cup_{i=1}^{cn} \widehat{P_i}$, where cn is the number of usage control commands and each $\widehat{P_i}$ is the set of subconjuncts of the precondition predicate P_i in the usage control command UC_i.*

Definition 5.5. *Let ω_{max} denote the maximum number of predicate conjuncts for a given usage control scheme U_ω.*

Lemma 5.6. $\omega_{max} \leq |\widehat{P_\omega}|$.

Proof. The maximum number of predicate conjuncts in each $\widehat{P_i}$ is $2^{|P_i|} - 1$. Hence the upper bound on maximum number of predicate conjuncts in the set $\widehat{P_\omega}$ is $\sum_{i=1}^{cn} 2^{|P_i|} - cn$. □

For example the usage control authorization scheme $U_{\omega\ game}$ in Example 4.8 has $\omega_{max} = 2^4 + 2^3 + 2^1 + 2^1 - 4 = 24$. The upper bound may not be realized since some atomic predicates may be duplicated in more than one P_i.

Values stored in the attributes of the objects in a usage configuration determine the satisfiability of the predicates in $\widehat{P_\omega}$ in that usage configuration. As the usage configuration changes the satisfiable predicates in $\widehat{P_\omega}$ may also change. We use the notions in Definitions 5.8 through 5.21 below to systematically track those changes.

We use the symbols $s.v_1, s.v_2 \ldots s.v_n$ to denote the values stored in the attributes $a_1(id), a_2 \ldots a_n$ of the subject s and the symbols $o.v_1, o.v_2 \ldots o.v_n$ to denote the values stored in the attributes $a_1(id), a_2 \ldots a_n$ of the object o. If an attribute a_i of an object o is yet to store a value, $o.v_i$ refers to the empty value ϕ. Satisfiability of the atomic predicates in the usage control commands are checked over the values stored in the subject object pairs.

The atomic predicates in the usage control commands are represented using the letter p with subscripts like $p_1, p_2 \ldots etc$. Depending on the relationship the predicates check and the attributes over which they check the relationship, there are fourteen forms of atomic predicates. Different forms of atomic predicates are represented using the letter p with superscripts like $p^1, p^2 \ldots p^{14}$. The formal definitions of satisfiability of each form of p^i is given in the Table 2. Each atomic predicate p_i in the usage control commands will be in one of the fourteen forms defined in the Table 2.

The left column in the table provides the definitions of satisfiability of atomic predicates p_i that are defined over the attributes of a subject, object pair (s,o) in non-creating usage control commands. The atomic predicate forms p^1, p^3, p^5 and p^8 are meant to check the relationships between the attribute values of the subject s and the atomic predicate forms p^2, p^4, p^6 and p^9 are meant to check the relationships between the attribute values of the object o. The remaining predicates in the left column, p^7 and p^{10}, are meant to check the relationships between the attribute values of s and o.

Example 5.7. Let the atomic predicate p_1 be $o.red \doteq s.player_{id}$. Then, the subject object pair (s_1, o_2) in Example 4.10 satisfies p_1, i.e., $(s_1, o_2) \models p_1$ while (s_1, o_1) does not, i.e., $(s_1, o_1) \not\models p_1$.

The predicates in non-creating commands are defined on the attributes of both the subject s and the object o whereas the predicates in creating usage control commands are defined only on the attributes of the subject s. In order to simplify the tracking of the changes in satisfiability of predicates of both create and non-create commands, we use o_ϕ to denote an empty object with no stored values in its attributes including the identifier attribute. The right column in the Table 2 provides the definitions of satisfiability of atomic predicates p_i that are defined over the attribute values of subject (s) in the creating usage control commands. We use the notation (s, o_ϕ) to refer an ordered pair of subject and object in the creating usage control commands, here, o_ϕ is the symbolic place holder for the newly created object.

For sake of brevity, we use the following definition to check if an atomic predicate p_i is satisfiable by the values stored in the attributes of (s,o) for both non-creating and creating commands.

Definition 5.8. *A subject object pair (s,o) is said to satisfy an atomic predicate p_i if and only if the values stored in the attributes of the pair satisfies the relation $(=, \neq, \doteq, or \not\doteq)$ defined in p_i over the attributes of (s, o) and it is written as $(s,o) \models p_i$.*

A usage configuration can have multiple subject object pairs. The safety decision procedure searches for a usage configuration in which a subject can get the unsafe right over an object. Therefore, we extend the Definition 5.8 to usage configurations as follows.

Definition 5.9. *An atomic predicate p_i is said to be satisfied in some usage configuration $U_{\Xi x}$ if and only if there exist an*

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

ordered subject object pair (s,o): $s, o \in U_{\Xi x}$ and $(s, o) \models p_i$. This is written as $U_{\Xi x} \models p_i$.

Example 5.10. The usage configuration $U_{\Xi t}$ in the Example 4.10 satisfies the atomic predicate p_1 in the example, that is, $U_{\Xi t} \models p_1$.

Usage control commands are guarded with the conjunction of multiple atomic predicates and a usage configuration can also satisfy multiple atomic predicates. All possible sub-conjuncts of $P_{\alpha 1} \wedge P_{\beta 1}, P_{\alpha 2} \wedge P_{\beta 2} \ldots P_{\alpha cn} \wedge P_{\beta cn}$ are in the set $\widehat{P_\omega}$. Therefore, we extend the Definition 5.9 to conjuncts of atomic predicates in $\widehat{P_\omega}$.

Definition 5.11. *A predicate conjunct $P_i \in \widehat{P_\omega}$, where $P_i = p_{i_1} \wedge p_{i_2} \wedge \cdots \wedge p_{i_n}$ is said to be satisfied in some usage configuration $U_{\Xi x}$ if and only if there exists a subject object pair (s, o): $s, o \in U_{\Xi x}$ and $(s, o) \models p_{i_1}$, $(s, o) \models p_{i_2} \ldots (s, o) \models p_{i_n}$. It is written as $U_{\Xi x} \models P_i$.*

Example 5.12. Let the predicate conjunct P_1 be $(s.player_{id} \neq \phi \wedge o.ball_{id} \neq \phi \wedge \quad o.red \doteq s.player_{id})$. The usage configuration $U_{\Xi t}$ in the Example 4.10 satisfies P_1, that is, $U_{\Xi t} \models P_1$. Here, $s_1, o_1 \in U_{\Xi t}$ and $(s_1, o_1) \models (s.player_{id} \neq \phi)$, $(s_1, o_1) \models (o.ball_{id} \neq \phi)$, and $(s_1, o_1) \models (o.red \doteq s.player_{id})$.

If a usage configuration $U_{\Xi x}$ satisfies the predicate conjunct $P_{\alpha i} \wedge P_{\beta i}$ in a usage control command UC_i then the command is enabled in $U_{\Xi x}$ and a subject in $U_{\Xi x}$ can execute the command. The execution of a command may create a new object and store values into the attributes of the existing object, as a result, the current usage configuration will change. We formalize the notion of successful execution of usage control command and the corresponding change in usage configuration using the following definition.

Definition 5.13. *An execution of a usage control command UC_i in a usage configuration $U_{\Xi x}$ is defined as an instance of the command and it is denoted as $uc_i(s, o)$, where s is the subject which executes the command UC_i on the object o. We use the notation $U_{\Xi x} \xrightarrow{uc_i(s,o)} U_{\Xi x'}$ to denote that the command instance $uc_i(s, o)$ occurred in $U_{\Xi x}$ and the $U_{\Xi x'}$ is the new usage configuration after the occurrence of $uc_i(s, o)$.*

If the execution of the usage control command is not successful then there will be not any change in the current usage configuration and it will not be called as an instance of the command.

Example 5.14. In the Example 4.10, the occurrence of usage control command instance $uc_4(s_1, o_\phi)$ in $U_{\Xi t}$ could change the configuration $U_{\Xi t}$ to $U_{\Xi t'} = \{s_1, s_2, o_1, o_2, o_3.player_{id} = \phi, o_3.ball_{id} = BI865, o_3.creator_{id} = PX021, o_3.red = \phi\}$. It is denoted as $U_{\Xi t} \xrightarrow{uc_4(s_1, o_\phi)} U_{\Xi t'}$.

If multiple usage control commands are enabled then any one of the enabled command can be executed in the current usage configuration. Likewise, if multiple subject object pairs satisfy the predicate conjunct $P_{\alpha i} \wedge P_{\beta i}$ in the usage control command UC_i, then any one of the pair can execute the usage control command UC_i.

We use the symbol uc_i to refer to a usage control command instance when the names of the particular subject and the object are not important in the context of discussion. Likewise, we use $\dot{u}c$ to denote an instance of a usage control command without naming the particular objects and without naming the particular usage control command.

The usage control command instances change the usage configurations and new configurations may satisfy additional atomic predicates and eventually reach a configuration where new usage control commands are enabled. For example, the faculty member can *submit* the final grade for a student after *evaluating* all his assignments and exams. In this example, each time the faculty member *evaluates* an assignment, the usage configuration will change and eventually when he completes the evaluation of all assignments and exams, the resulting usage control configuration will enable the $submit$ command. To analyze the cumulative effect of such incremental changes we need to define the notion of reachable configurations.

Definition 5.15. *A usage configuration $U_{\Xi x'}$ is said to be reachable from another usage configuration $U_{\Xi x}$ if and only if there is a sequence of command instances such that $U_{\Xi x} \xrightarrow{\dot{u}c} U_{\Xi x_1} \xrightarrow{\dot{u}c} U_{\Xi x_2} \ldots \xrightarrow{\dot{u}c} U_{\Xi x'}$.*

Example 5.16. In the Example 4.10, the usage configuration $U_{\Xi t''} = \{s_1,\ s_2,\ o_2,\ o_3,\ o_1.player_{id} = \phi, o_1.ball_{id} = BI213, o_1.creator_{id} = PX021, o_1.red = PX756\}$ is reachable from $U_{\Xi t}$ since there is a sequence of command instances from $U_{\Xi t}$ to $U_{\Xi t''}$, that is, $U_{\Xi t} \xrightarrow{uc_4(s_1, o_\phi)} U_{\Xi t'} \xrightarrow{uc_1(s_2, o_1)} U_{\Xi t''}$.

In a usage configuration, a subject may have options to execute more than one usage control command. For example, a student can take either a music course or a management course as an elective and depending on the elective he takes, he may have different sets of additional rights over different objects. Though both the choices together are not realizable, nevertheless both of them are individually reachable from the initial configuration. Therefore, the safety decision procedure needs to explore all such non-deterministic choices left with each individual subjects. This non-determinism adds another level of complexity to the safety analysis as it provides more than one possible future usage configurations for one usage configuration. To simplify the analysis, we extend the Definition 5.11 to a set of usage configurations as follows.

Definition 5.17. *A predicate conjunct $P_i \in \widehat{P_\omega}$, $P_i = (p_{i1} \wedge p_{i2} \wedge \cdots \wedge p_{in})$ is said to be satisfied in a set of usage configurations $\widehat{U_{\Xi X}}$ if and only if $\exists U_{\Xi x} \in \widehat{U_{\Xi X}}$ such that $U_{\Xi x} \models P_i$. It is written as $\widehat{U_{\Xi X}} \models P_i$.*

Example 5.18. Let the predicate conjuncts P_2 be $(s.player_{id} \neq \phi \wedge o.ball_{id} \neq \phi)$ and $P_3 : (s.player_{id} \neq \phi \wedge o.ball_{id} \neq \phi \wedge o.red \doteq s.player_{id})$. Let the two sets of usage configurations from Example 5.16 be $\widehat{U_{\Xi T_1}} = \{U_{\Xi t}, U_{\Xi t'}\}$ and $\widehat{U_{\Xi T_2}} = \{U_{\Xi t'}, U_{\Xi t''}\}$. In this example, $\widehat{U_{\Xi T_1}} \models P_2$, $\widehat{U_{\Xi T_1}} \not\models P_3$, $\widehat{U_{\Xi T_2}} \models P_2$ and $\widehat{U_{\Xi T_2}} \models P_3$.

With help of Definition 5.17, we can track the changes in the satisfiability of set of predicate conjuncts in $\widehat{P_\omega}$ in a set of reachable configurations. We formally define the set of reachable configurations in k or less steps as follows.

Definition 5.19. *A set of usage configurations $\{U_{\Xi x_1}, U_{\Xi x_2}, \ldots, U_{\Xi x_n}\}$ is said to be reachable from the initial configuration $U_{\Xi init}$ in k or less steps if and only if for each $U_{\Xi x_i}$, either $U_{\Xi x_i} = U_{\Xi init}$ or there exists a sequence of k or less number of command instances such that $U_{\Xi init} \xrightarrow{\dot{u}c} U_{\Xi x_{i_1}} \xrightarrow{\dot{u}c} U_{\Xi x_{i_2}} \ldots \xrightarrow{\dot{u}c} U_{\Xi x_i}$. This set is denoted as $\widehat{U_{\Xi init \uparrow k}}$. We use k as the step-variable.*

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

The set of configurations reachable in zero steps $\widehat{U_{\Xi init\uparrow 0}}$ is the singleton $\{U_{\Xi init}\}$.

Example 5.20. In Example 4.10, the set of usage configurations reachable in one step is $\widehat{U_{\Xi init\uparrow 1}}=\{U_{\Xi t}, U_{\Xi t'}, U_{\Xi t''}, U_{\Xi t'''}\}$, where $U_{\Xi t}, U_{\Xi t'}$ are same as in Example 5.14, $U_{\Xi t''} = \{s_1, s_2, o_1, o_2\}$ as in Example 5.16, and $U_{\Xi t'''} = \{s_1, s_2, o_1, o_2, s_3.player_{id} = PX356, s_3.ball_{id} = \phi, s_3.creator_{id} = \phi, s_3.red = \phi\}$.

We define the function $\widehat{f_{\Xi\omega}}$ to track the changes in the satisfiable predicate conjuncts in $\widehat{P_\omega}$.

Definition 5.21. *The predicate function* $\widehat{f_{\Xi\omega}} : \widehat{U_{\Xi X}} \rightarrow \mathcal{P}(\widehat{P_\omega})$ *returns a set of predicate conjuncts* $\{P_{i_1}, P_{i_2} \ldots P_{i_n} : \forall k, 1 \leq k \leq n, (P_{i_k} \in \widehat{P_\omega})\ \&\ (\widehat{U_{\Xi X}} \models P_{i_k})\}$. *The symbol* $\mathcal{P}(\widehat{P_\omega})$ *denotes the power set of* $\widehat{P_\omega}$.

Example 5.22. Let $\widehat{P_\omega} = \{P_1, P_2, P_3\}$ be a set of predicate conjuncts, where P_1 is from Example 5.12 and P_2, P_3 are from Example 5.18. Then, $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi T_1}}) = \{P_1, P_2\}$ and $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi T_2}}) = \{P_1, P_2, P_3\}$, where $\widehat{U_{\Xi T_1}}$ and $\widehat{U_{\Xi T_2}}$ are from Example 5.18.

We pictorially summarize the notions defined so far in the Fig. 1. The upward arrows denote the requests from the subjects to execute the usage control commands. The downward arrows denote the successful execution of usage control commands, also called as the command instances.

The sequence of solid circles denotes one possible sequence of usage configurations created as a result of a sequence of usage command instances. The dotted circles denote the starting usage configurations of possible alternative sequences of usage configurations. The set of reachable usage configurations in a $PreUCON_A^{id}$ system is the collection of all such sequences of configurations. It is denoted by the larger square which includes all the circles in the figure. The smaller square denotes the set of usage configurations reachable in 1 or less number of steps.

The set of configurations reachable from $U_{\Xi init}$ in unbounded number of steps is symbolically denoted by $\widehat{U_{\Xi init\uparrow *}}$. In general $\widehat{U_{\Xi init\uparrow *}}$ is countably infinite, since the number of objects that can be created is unbounded. Each newly created object by definition results in a new configuration. However, for a fixed k the set of reachable configurations is finite as shown in Lemma 5.2 in [4].

Definition 5.23. *Two sets of usage configurations* $\widehat{U_{\Xi X}}$ *and* $\widehat{U_{\Xi Y}}$ *are said to be ω-equivalent, written* $\widehat{U_{\Xi X}} \approx_\omega \widehat{U_{\Xi Y}}$, *if and only if* $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi X}}) = \widehat{f_{\Xi\omega}}(\widehat{U_{\Xi Y}})$.

Example 5.24. In the Example 5.18, for $\widehat{P_\omega}$ in the Example 5.22, $\widehat{U_{\Xi T_1}} \not\approx_\omega \widehat{U_{\Xi T_2}}$. Let $\widehat{U_{\Xi T_3}} = \{U_{\Xi t}, U_{\Xi t'}, U_{\Xi t''}\}$, then $\widehat{U_{\Xi T_3}} \approx_\omega \widehat{U_{\Xi T_2}}$.

This brings us to the central result of this paper.

Theorem 5.25. $\widehat{U_{\Xi init\uparrow \omega_{max}}} \approx_\omega \widehat{U_{\Xi init\uparrow *}}$.

Proof. Follows from Lemmas 5.26 and 5.27 below. □

Lemma 5.26. *Suppose there exists* k *such that* $\widehat{U_{\Xi init\uparrow k+1}} \approx_\omega \widehat{U_{\Xi init\uparrow k}}$. *Then for all* $n \geq k+2$, $\widehat{U_{\Xi init\uparrow n}} \approx_\omega \widehat{U_{\Xi init\uparrow k}}$.

Proof. By assumption, $\widehat{U_{\Xi init\uparrow k+1}} \approx_\omega \widehat{U_{\Xi init\uparrow k}}$.

Assume for contradiction that $\widehat{U_{\Xi init\uparrow k+2}} \not\approx_\omega \widehat{U_{\Xi init\uparrow k+1}}$.

Since $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow k}}) = \widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow k+1}})$, the above contradiction can occur only if there is some $U_{\Xi p} \in \widehat{U_{\Xi init\uparrow k+2}}$ and $\forall q$, $U_{\Xi q} \in \widehat{U_{\Xi init\uparrow k+1}}$, $\widehat{f_{\Xi\omega}}(U_{\Xi q}) \neq \widehat{f_{\Xi\omega}}(U_{\Xi p})$. Such a $U_{\Xi p}$ must be the result of executing some command $uc_t(s_x, o_y)$, for some s_x, o_y, enabled in a $U_{\Xi r} \in \widehat{U_{\Xi init\uparrow k+1}}$ and not enabled by any configuration in $\widehat{U_{\Xi init\uparrow k}}$. We prove that our usage control authorization scheme does not have any such commands.

Case a: Let us suppose that $uc_t(s_x, o_y)$ is a non-creating command guarded with the α-predicate conjunct $P_{\alpha t}$ and the β-predicate conjunct $P_{\beta t}$. If the predicate conjunct $P_{\alpha t} \wedge P_{\beta t}$ is satisfied for an s_x, o_y in $U_{\Xi r} \in \widehat{U_{\Xi init\uparrow k+1}}$ then there must exist some s'_x, o'_y in some $U_{\Xi r'} \in \widehat{U_{\Xi init\uparrow k}}$ that satisfies the same predicate conjunct $P_{\alpha t} \wedge P_{\beta t}$. Therefore, if the non-creating command uc_t is enabled in $\widehat{U_{\Xi init\uparrow k+1}}$ then it must be enabled in $\widehat{U_{\Xi init\uparrow k}}$ as well. Further, the execution of uc_t in $U_{\Xi r'}$ must produce a configuration $U_{\Xi s}$, $U_{\Xi s} \in \widehat{U_{\Xi init\uparrow k+1}}$, $\widehat{f_{\Xi\omega}}(U_{\Xi s}) = \widehat{f_{\Xi\omega}}(U_{\Xi p})$ which is a contradiction to $\forall q, U_{\Xi q} \in \widehat{U_{\Xi init\uparrow k+1}}, \widehat{f_{\Xi\omega}}(U_{\Xi q}) \neq \widehat{f_{\Xi\omega}}(U_{\Xi p})$. Otherwise, $\widehat{U_{\Xi init\uparrow k+1}} \approx_\omega \widehat{U_{\Xi init\uparrow k}}$ would not have been true.

Case b: Let us suppose that $uc_t(s_x, o_y)$ is a creating command guarded with the α-predicate conjunct $P_{\alpha t}$ and the β-predicate conjunct $P_{\beta t}$. If the predicate conjunct $P_{\alpha t} \wedge P_{\beta t}$ is satisfied for some s_x in $U_{\Xi r} \in \widehat{U_{\Xi init\uparrow k+1}}$ then there must exist some s'_x in some $U_{\Xi r'} \in \widehat{U_{\Xi init\uparrow k}}$ that satisfies the same predicate conjunct $P_{\alpha t} \wedge P_{\beta t}$. Therefore, if the creating command uc_t is enabled in $\widehat{U_{\Xi init\uparrow k+1}}$ then it must be enabled in $\widehat{U_{\Xi init\uparrow k}}$ as well. Further, the execution of uc_t in $U_{\Xi r'}$ must produce a configuration $U_{\Xi s}$, $U_{\Xi s} \in \widehat{U_{\Xi init\uparrow k+1}}$, $\widehat{f_{\Xi\omega}}(U_{\Xi s}) = \widehat{f_{\Xi\omega}}(U_{\Xi p})$ which is a contradiction to $\forall q, U_{\Xi q} \in \widehat{U_{\Xi init\uparrow k+1}}$, $\widehat{f_{\Xi\omega}}(U_{\Xi q}) \neq \widehat{f_{\Xi\omega}}(U_{\Xi p})$. Otherwise, $\widehat{U_{\Xi init\uparrow k+1}} \approx_\omega \widehat{U_{\Xi init\uparrow k}}$ would not have been true.

Therefore, the assumption $\widehat{U_{\Xi init\uparrow k+2}} \not\approx_\omega \widehat{U_{\Xi init\uparrow k+1}}$ cannot be true, by contradiction. This argument can be applied inductively for $n > k+2$. Note that, if the attribute update operations are allowed to use arithmetic operations like add and subtract then this argument would not be valid. □

Lemma 5.27. $\widehat{U_{\Xi init\uparrow \omega_{max}+1}} \approx_\omega \widehat{U_{\Xi init\uparrow \omega_{max}}}$.

Proof. There are at most ω_{max} predicate conjuncts that can be added to $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow *}})$ from a given $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow 0}})$. The longest chain of such additions is at most ω_{max} (which would happen if only one new predicate conjunct is added at each step). Thus it is not possible to have a conjunct $P_q \in \widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow \omega_{max}+1}})$ and $P_q \notin \widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow \omega_{max}}})$. □

Theorem 5.28. *The set* $\widehat{U_{\Xi init\uparrow \omega_{max}}}$ *is computable and can be used to answer any safety question* $\Upsilon_{(r)}$.

Proof. By definition $\Upsilon_{(r)}$ can be true if and only if there is some $P_{\alpha i} \wedge P_{\beta i}$ in $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow *}})$ which is the pre-condition of a creating or non-creating command that grants r.

Procedure Q_ω given below computes $\widehat{U_{\Xi init\uparrow \omega_{max}}}$ from which $\widehat{f_{\Xi\omega}}(\widehat{U_{\Xi init\uparrow \omega_{max}}})$ can be trivially computed. Let $\widehat{U_{\Xi init\uparrow =k}}$ denote the set of configurations that can be reached from $U_{\Xi init}$ by using exactly k usage control commands. This is similar to Definition 5.19 except for requiring exactly k commands rather than k or less.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

```
Procedure $Q_\omega$:
Let $\widehat{U_{\Xi init\uparrow=0}} = \{U_{\Xi init}\}$;
Let $\widehat{U_{\Xi init\uparrow 0}} = \{U_{\Xi init}\}$;
Let $k = 1$;
While $k \leq |\widehat{P_\omega}|$ do begin
  $\widehat{U_{\Xi init\uparrow=k}} = \emptyset$;
  For each $U_{\Xi p} \in \widehat{U_{\Xi init\uparrow=k-1}}$ do begin
    Add all $U_{\Xi q}$ reachable from $U_{\Xi p}$
    in one command to $\widehat{U_{\Xi init\uparrow=k}}$
    without any duplicates;
  end for;
  $\widehat{U_{\Xi init\uparrow k}} = \widehat{U_{\Xi init\uparrow k-1}} \cup \widehat{U_{\Xi init\uparrow=k}}$;
  $k = k + 1$;
end while;
```

By the argument given in the proof of Lemma 5.2 in [4], each iteration of the for loop above is computable, and thereby Procedure Q_ω computes $\widehat{U_{\Xi init\uparrow\omega_{max}}}$. □

Computation of $\omega-$equivalent usage configurations in context of Example 4.8 is given in Appendix E, available in the online supplemental material.

6 Illustration of Expressiveness

We illustrate the expressiveness of $PreUCON_A^{id}$ using authorization policies applicable for Massive Online Open Course (MOOC) type learning portals that support unbounded number of subjects and objects. Though the attributes of the objects are exclusively meant to store object identifiers, this model can express interesting policies such as consumable rights, regenerative rights, separation of duty constraints, and data provenance over potentially infinite number of objects. To the best of our knowledge the safety decidable models in the prior literature do not support these policy features.

6.1 Application of $PreUCON_A^{id}$ in Learning Portals

An object schema $OS_{\omega\ eportal}$ and the set of usage rights UR for an example e-learning portal are defined as follows.

(i) $OS_{\omega\ eportal}$ = [$id : ID, student_{id} : ID, course_{id} : ID, exam_{id} : ID, faculty_{id} : ID, advisor : ID, nadvisor_1 : ID,\ nadvisor_2 : ID, examiner_1 : ID,\ examiner_2 : ID, referee : ID$]

(ii) $UR = \{BuyVoucher, TakeExam, ChangeAdvisor, EvaluateCopy, InductFaculty\}$

The following provides examples of policy statements and their expression in our authorization scheme.

(i) *Consumable and Regenerative Rights*: Students in a course can buy examination vouchers. A voucher is valid for taking examination one time.

UC_1: **Permit**$_{BuyVoucher}(s, o)$

PreCondition:

$(s.id \neq \phi) \wedge$
$(s.student_{id} \doteq s.id) \wedge$
$(s.course_{id} \neq \phi)$

PreUpdate:

create o;
$o.course_{id} := s.course_{id}$;
$o.student_{id} := s.id$;

UC_2: **Permit**$_{TakeExam}(s, o)$

PreCondition:

$(s.id \neq \phi) \wedge (o.id \neq \phi) \wedge$
$(s.course_{id} \doteq o.course_{id}) \wedge$
$(s.id \doteq o.student_{id})\wedge$
$(o.exam_{id} = \phi)$

PreUpdate:

$o.exam_{id} := s.course_{id}$;

A student with a voucher can take the exam. Each time he takes the exam, the $TakeExam$ right gets consumed and each time he buys a voucher, the $TakeExam$ right gets regenerated. Meaning of the predicates and attribute updates are kept intuitively consistent with their names.

(ii) *Relationships Among Objects*: Students may change their advisor two times during the course.

UC$_3$: Permit$_{Ch.Advisor}(s, o)$

PreCondition:

$(s.student_{id} \neq \phi) \wedge$
$(o.faculty_{id} \neq \phi) \wedge$
$(s.advisor \neq \phi) \wedge$
$(s.nadvisor_1 = \phi)$

PreUpdate:

$s.nadvisor_1 :=$
$o.faculty_{id}$;

UC$_4$: Permit$_{Ch.Advisor}(s, o)$

PreCondition:

$(s.student_{id} \neq \phi)\wedge$
$(o.faculty_{id} \neq \phi)\wedge$
$(s.advisor \neq \phi)\wedge$
$(s.nadvisor_1 \neq \phi)\wedge$
$(s.nadvisor_2 = \phi)$

PreUpdate:

$s.nadvisor_2 :=$
$o.faculty_{id}$;

A student s can choose a faculty member o as his new advisor and he can execute his right to change the advisor two times. This right is a consumable right, that is, a student cannot change his advisor more than two times.

(iii) *Dynamic Separation of Duty*: Each answer sheet must be evaluated by two different examiners.

UC_5: **Permit**$_{Ev.Copy}(s, o)$

PreCondition:

$(s.id \neq \phi) \wedge (o.id \neq \phi) \wedge$
$(s.course_{id} \doteq o.course_{id})$
$\wedge\ (o.exam_{id} \neq \phi) \wedge$
$(s.id \doteq s.faculty_{id}) \wedge$
$(o.examiner_1 = \phi) \wedge$
$(o.examiner_2 = \phi)$

PreUpdate:

$o.examiner_1 := s.id$;

UC_6: **Permit**$_{Ev.Copy}(s, o)$

PreCondition:

$(s.id \neq \phi) \wedge (o.id \neq \phi)\wedge$
$(s.course_{id} \doteq o.course_{id})$
$\wedge\ (o.exam_{id} \neq \phi) \wedge$
$(s.id = s.faculty_{id}) \wedge$
$(o.examiner_1 \dot{\neq} s.id) \wedge$
$(o.examiner_2 = \phi)$

PreUpdate:

$o.examiner_2 := s.id$;

The students cannot evaluate the answers and the faculty members cannot take the exam. Further, the second examiner must be different from the first examiner. With additional attributes we can enrich this policy with additional examiners and constraints on Teaching Assistants, Certification Authority etc.

(iv) *Decentralized Delegation of Authority*: A faculty member of a course can induct a new faculty member.

UC_7: **Permit** $_{InductFaculty}$ (s,o)

PreCondition:	**PreUpdate:** create o;
$(s.id \neq \phi) \wedge$	$o.course_{id} := s.course_{id}$;
$(s.faculty_{id} \doteq s.id) \wedge$	$o.referee := s.id$;
$(s.course_{id} \neq \phi)$	$o.faculty_{id} := o.id$;

This policy permits any faculty member of a course to induct a new faculty member and the new faculty member can have same rights as the other faculty members. The new faculty member can further induct additional faculty members. Such delegation of authorities can also be constrained with additional predicates.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

6.2 Decentralized Administration and Automation

Safety decidable $PreUCON_A$ models are amicable for decentralized administration and automation of security administration. In practice, security administration of large information systems are often managed with delegation of administrative responsibilities to multiple admins in a decentralized manner. The $PreUCON_A^{id}$ model supports administrators to add new administrators to share their duties as well as to delegate some of their duties to the existing administrators. In the learning portal example given in the Section 6.1, policies like inducting new faculty members and teaching assistants illustrate the decentralized administration within the learning portal. When the administrative rights are delegated to multiple administrators, it is important to ensure that the organization's central policies are not violated. In the the learning portal example, a student can become TA and a faculty member can also enroll in a course and become a student. However, the portal should not allow the students to evaluate their own answer sheets. Manually checking such policy violations is a difficult task. In the $PreUCON_A^{id}$ model, we can express such violations as unsafe usage right and apply the safety decision procedure to automatically check if any of the reachable usage configuration can permit such an unsafe usage right.

7 Discussion and Future Work

Besides the features mentioned in Section 6, the $PreUCON_A^{id}$ model also has its own limitations. For example, it cannot express policies like (i) a faculty member can claim incentives based on the number of exam copies he evaluates in a semester, and (ii) top 10 percent of students in the course are permitted to become student ambassadors for the high schools. Expression of these policies requires explicit arithmetic operations in the update functions which are lacking in $PreUCON_A^{id}$. Further, the attributes of the objects also need to be extended beyond the domain of object identifiers such as integer domains.

The inclusion of integer valued attributes in the object schema of $PreUCON_A^{id}$ and integer-valued functions in the pre-update part of the usage control commands would help in expressing usage control policies beyond $PreUCON_A^{id}$. However, determining the safety decidability status of such an integrated model is a difficult problem. One part of the difficulty essentially comes from identifying a safety decidable fragment of $PreUCON_A$ with integer valued functions in the pre-update part of the usage control commands. Another part of the difficulty comes from checking effect of combining the identifier attributes and the integer domain attributes in a usage control authorization scheme. We discuss both these issues separately below.

First, we discuss the issues in the safety analysis of $PreUCON_A^{Integer}$ with integer attributes without identifier attributes. In this model, the integer domain attributes of the objects can take potentially unbounded number of different values as the subjects can execute the usage control commands an unbounded number of times and each such execution can change the values stored in the attributes of the objects. In such models, the authorization predicates may become true after unbounded number of changes in the objects' attribute values. Whereas in the $PreUCON_A^{id}$ model, the pre-update part of the usage control commands allows only copying object identifiers from one attribute to another attribute. Further, arithmetic operations over the object identifiers are not allowed, therefore, safety analysis can be performed by tracking changes in the satisfiability of set of all the atomic predicates, in the usage control commands, which checks equality between stored values in the attributes of the objects in each of the new configurations. This approach will not work for $PreUCON_A^{Integer}$ since the attribute values are updated with arithmetic functions. We conjecture that $PreUCON_A^{Integer}$ without object creation itself can simulate the two-counter machine [35] and finding a safety decidable fragment of the $PreUCON_{Integer}$ without identifier attributes itself is a non-trivial problem.

Second part of the problem is to check the effect of combining two safety decidable models which we explore in detail in the following section.

7.1 Safety in $PreUCON_A^{id+finite}$ is Undecidable

$PreUCON_A^{id}$ can express various relationships between objects in systems with unbounded object creation and it is not so flexible in expressing policies having arithmetic nature as the update functions are constrained. On the other hand, safety decidable $PreUCON_A^{finite}$ with finite domain attributes [4] supports unconstrained update functions, therefore, it is flexible in expressing policies with arithmetic operations but has limitations in expressing relationships. In this section, we examine the possibilities of combining the features of these two safety decidable models $PreUCON_A^{id}$ and $PreUCON_A^{finite}$. Let us call the combined model as $PreUCON_A^{id+finite}$.

$PreUCON_A^{id+finite}$ has both infinite identifier attributes and finite domain attributes. The update constraints of identifier attributes remain the same as in $PreUCON_A^{id}$. The finite domain attributes can be updated without any restriction as in $PreUCON_A^{finite}$. We define a usage control authorization scheme $U_{\omega.f}$ in $PreUCON_A^{id+finite}$ with the combined features and analyze safety in the combined model.

Definition 7.1. *Usage control authorization scheme $U_{\omega.f}$ has three components as follows.*

(i) *an object schema $OS_{\omega.f}$,*
(ii) *a set of usage rights $UR_{\omega.f} = \{r_1, r_2, \ldots, r_m\}$,*
(iii) *a set of usage control commands $UC_{\omega.f} = \{UC_1, UC_2, \ldots, UC_n\}$.*

Usage rights remain the same as in U_ω. The changes in the remaining components of the scheme are as follows.

Definition 7.2. *The object schema $OS_{\omega.f}$ is of the form $[id : ID, a_2 : ID, a_3 : ID \ldots a_n : ID, b_1 : FD_1, b_2 : FD_2 \ldots b_m : FD_m]$, where the first n attributes are identifier domain attributes and the remaining m attributes are finite domain attributes.*

Definition 7.3. *The predicate P_γ is defined as any Boolean function that takes finite domain attributes of the subject, object pair and returns $true$ or $false$.*

The structure of a creating command UC_i in $U_{\omega.f}$ is as follows.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

Command_Name$_{create}(s, o)$

$$\begin{aligned}
&\textbf{PreCondition:} P_\alpha \bigwedge P_\beta \bigwedge P_\gamma \\
&\textbf{PreUpdate:}\ Createo; \\
&s.a_{i_1} := fa_{i_1}(s, o.id);\ s.b_{k_1} := fb_{k_1}(s); \\
&\quad \dots \qquad\qquad\qquad \dots \\
&s.a_{i_p} := fa_{i_p}(s, o.id);\ s.b_{k_r} := fb_{k_r}(s); \\
&o.a_{j_1} := fa_{j_1}(s, o.id);\ o.b_{l_1} := fb_{l_1}(s); \\
&\quad \dots \qquad\qquad\qquad \dots \\
&o.a_{j_q} := fa_{j_q}(s, o.id);\ o.b_{l_s} := fb_{l_s}(s);
\end{aligned}$$

In this command s is an input parameter and o is an output parameter. P_α and P_β are the α-predicate conjunct and the β-predicate conjunct, respectively, defined over the identifier domains attributes of the subject s and the $o.id$ attribute. P_γ is a conjunction of Boolean functions defined over the finite domain attributes of s.

The $create$ operation carries the usual meaning. The update functions whose names start with fa are identifier domain functions which return a value from the identifier attributes of s or return $o.id$. Likewise, the functions whose names start with fb are finite domain functions which take finite domain attributes of s as input and return a value from the appropriate finite domain. Due to space limitations the structure of the non-creating commands is given in Appendix F, available in the online supplemental material.

$PreUCON_A^{id+finite}$ can express relationships between infinite number of objects and unconstrained policies over finite security attributes. Though the safety in both sub-models $PreUCON_A^{id}$ and $PreUCON_A^{finite}$ are individually decidable, even if we add one finite attribute of $PreUCON_A^{finite}$ into $PreUCON_A^{id}$ the safety becomes undecidable in $PreUCON_A^{id+finite}$.

Theorem 7.4. *Safety in* $PreUCON_A^{id+finite}$ *with one finite domain attribute is undecidable.*

Similar to other proofs of undecidability of safety [2], [5], [13], we reduce the halting problem to the safety problem in $PreUCON_A^{id+finite}$. Our proof of undecidability of safety uses just one finite domain attribute. The proof details are given in Appendix G, available in the online supplemental material.

7.2 Future Work

The above result shows that simply combining two decidable safety models can lead to undecidable safety. A viable strategy might be to consider constraints on finite attributes that are combined with the safety decidable $PreUCON_A^{id}$ model developed in this paper. For example, DRM applications can require users to pay/repay certain amount for a certain number of usage of resources [1]. Such policies can be expressed with a counting functions with reset features.

The safety decidable $PreUCON_A^{id}$ with identifier attribute domains is restrictive. The model does not support feature like multiple updates on attributes, set-valued attributes and array-valued attributes. Safety analysis of a sub-model with the above mentioned feature may require a data flow analysis on different attributes of infinite number of objects and it is a non-trivial work. It would be interesting to find if safety in the sub-models of $PreUCON_A$ with above mentioned features are decidable.

Safety analysis of the on-going authorization model is also an important problem; they are applicable in protecting resources in reactive systems like web services. Besides the attribute domain and update functions the on-going authorization model has an additional layer of difficulty due to interleaved and nonterminating executions of usage control commands. Extending the decidability result to similar sub-models in $OnUCON_A$ is also non-trivial work. Lemma 5.2 in [4] would not hold as the usage control commands in the on-going authorization models may create unbounded number of new objects while executing one usage right. Note that an execution of usage control command may not terminate in $OnUCON_A$ unless the on-going authorization predicates become false or until the subject explicitly terminates the execution. Therefore, we may need to develop a different proof strategy for such models.

There are many avenues for future work in safety analysis of various sub-models in usage control besides pre-authorization model. Undecidability proofs, along with other results, in Theorems A.5 and A.6, available in the online supplemental material, may help as demarcating tools in developing various safety decidable sub-models of $UCON_{ABC}$ in future.

8 Conclusion

In this work, we presented a safety decidability result for $PreUCON_A^{id}$ model with object identifier attributes. The safety decidable model can support interesting policies that are useful in today's information systems with unbounded resources. We conjuncture that (i) adding array attributes with constant size, integer attributes, and type attributes into the object schema would not affect the safety decidability of the model, and (ii) adding unrestricted arithmetic update operations on integer attributes would make the models safety undecidable. Further, studying safety decidability of the UCON with obligations, conditions, combination of finite and infinite attributes are interesting problems to explore. However, detailed analysis on such extensions are outside the scope of this article.

Acknowledgments

This research is partially supported by US National Science Foundation Grants CNS-1111925 and CNS-1423481.

References

[1] J. Park and R. Sandhu, "The $UCON_{ABC}$ usage control model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, pp. 128–174, Feb. 2004.

[2] X. Zhang, R. Sandhu, and F. Parisi-Presicce, "Safety analysis of usage control authorization models," in *Proc. ACM Symp. Inf. Comput. Commun. Secur.*, Mar. 21–24, 2006, pp. 243–254.

[3] P. V. Rajkumar, "Formal and semi-formal methods for application specific usage control and security," PhD dissertation, School of Information Technology, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal, India, Aug. 2012.

[4] P. V. Rajkumar and R. Sandhu, "Safety decidability for pre-authorization usage control with finite attribute domains," *IEEE Trans. Depend. Secure Comput.*, vol. 13, no. 5, pp. 582–590, Sep./Oct. 2016.

[5] M. Harrison, W. Ruzzo, and J. Ullman, "Protection in operating systems," *Commun. ACM*, vol. 19, pp. 461–471, Aug. 1976.

[6] M. V. Tripunitara and N. Li, "The foundational work of Harrison-Ruzzo-Ullman revisited," *IEEE Trans. Depend. Secure Comput.*, vol. 10, no. 1, pp. 28–39, Jan. 2013.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.

[7] R. Lipton and L. Snyder, "A linear time algorithm for deciding subject security," *J. ACM*, vol. 24, pp. 455–464, Aug. 1977.

[8] R. Sandhu, "The schematic protection model: Its definition and analysis for acyclic attenuating schemes," *J. ACM*, vol. 35, no. 2, pp. 404–432, 1988.

[9] R. Sandhu, "Expressive power of the schematic protection model," *J. Comput. Secur.*, vol. 1, no. 1, pp. 59–98, 1992.

[10] V. Varadharajan and C. Calvelli, "Extending the schematic protection model—I. Conditional tickets and authentication," in *Proc. IEEE Symp. Res. Secur. Privacy*, May 16–18, 1994, pp. 213–229.

[11] V. Varadharajan, "Extending the schematic protection model—II. Revocation," *ACM SIGOPS Operating Syst. Rev.*, vol. 31, pp. 64–77, Jan. 1997.

[12] P. E. Ammann and R. Sandhu, "Safety analysis for the extended schematic protection model," in *Proc. IEEE Comput. Soc. Symp. Res. Secur. Privacy*, May 20–22, 1991, pp. 87–97.

[13] R. Sandhu, "The typed access matrix model," in *Proc. IEEE Symp. Secur. Privacy*, May 04–06, 1992, pp. 122–136.

[14] M. Soshi, M. Maekawa, and E. Okamoto, "The dynamic-typed access matrix model and decidability of the safety problem," *IEICE Trans. Fundam.*, vol. E87-A, pp. 190–203, Jan. 2004.

[15] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[16] D. F. Ferraiolo, R. Sandhu, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, pp. 224–274, Aug. 2001.

[17] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Trans. Inf. Syst. Secur.*, vol. 4, pp. 191–233, Aug. 2001.

[18] J. Joshi, E. Bertino, and A. Ghafoor, "An analysis of expressiveness and design issues for the generalized temporal role-based access control model," *IEEE Trans. Depend. Secure Comput.*, vol. 2, no. 2, pp. 157–175, Apr.–Jun. 2005.

[19] I. Ray and M. Toahchoodee, "A spatio-temporal role-based access control model," in *Proc. 21st Annu. IFIP WG 11.3 Work. Conf. Data Appl. Secur.*, Jul. 2007, pp. 211–226.

[20] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Trans. Inf. Syst. Secur.*, vol. 2, pp. 105–135, Feb. 1999.

[21] J. Crampton and G. Loizou, "Administrative scope: A foundation for role-based administrative models," *ACM Trans. Inf. Syst. Secur.*, vol. 6, pp. 201–231, May 2003.

[22] A. Schaad, J. Moffett, and J. Jacob, "The role-based access control system of a european bank: A case study and discussion," in *Proc. 6th ACM Symp. Access Control Models Technol.*, May 2001, pp. 3–9.

[23] S. Stoller, P. Yang, C. Ramakrishnan, and M. Gofman, "Efficient policy analysis for administrative role based access control," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, Oct. 29–Nov. 02, 2007, pp. 445–455.

[24] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, "A logical framework for reasoning about access control models," *ACM Trans. Inf. Syst. Secur.*, vol. 6, pp. 71–127, 2003.

[25] S. Osborn, "Information flow analysis of an RBAC system," in *Proc. 7th ACM Symp. Access Control Models Technol.*, Jun. 03–04, 2002, pp. 163–168.

[26] A. L. Ferrara, P. Madhusudan, and G. Parlato, "Security analysis of role-based access control through program verification," in *Proc. IEEE 25th Comput. Secur. Found. Symp.*, Mar. 2012, pp. 113–125.

[27] A. Armando and S. Ranise, "Scalable automated symbolic analysis of administrative role-based access control policies by SMT solving," *J. Comput. Secur.*, vol. 20, pp. 309–352, Jul. 2012.

[28] N. Li, J. Mitchell, and W. Winsborough, "Design of a role-based trust-management framework," in *Proc. IEEE Symp. Secur. Privacy*, May 12–15, 2002, pp. 114–130.

[29] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 9, pp. 391–420, Nov. 2006.

[30] K. Jayaraman, V. Ganesh, M. V. Tripunitara, M. C. Rinard, and S. J. Chapin, "Automatic error finding in access-control policies," in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2011, pp. 17–21.

[31] K. Jayaraman, M. V. Tripunitara, V. Ganesh, M. C. Rinard, and S. J. Chapin, "MOHAWK: Abstraction-refinement and bound-estimation for verifying access control policies," *ACM Trans. Inf. Syst. Secur.*, vol. 15, Apr. 2013, Art. no. 18.

[32] J. Shahen, J. Niu, and M. V. Tripunitara, "Mohawk+T: Efficient analysis of administrative temporal role-based access control (ATRBAC) policies, Vienna, Austria," in *Proc. 20th ACM Symp. Access Control Models Technol.*, Jun. 2015, pp. 15–26.

[33] Z. Zhigang, W. Jiandong, and M. Yuguang, "Study and safety analysis on $UCON_{onA}$ model," in *Proc. 1st Int. Workshop Database Technol. Appl.*, Apr. 25–26, 2009, pp. 103–106.

[34] S. Ranise and A. Armando, "On the automated analysis of safety in usage control: A new decidability result," in *Proc. 6th Int. Conf. Netw. Syst. Secur.*, Nov. 21–23, 2012, pp. 15–28.

[35] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. New York, USA: Pearson Education, Apr. 2012.

P.V. Rajkumar received the PhD degree from Indian Institute of Technology Kharagpur, India. He is an Assistant Professor of Computer Information Systems and Analytics with the University of Central Missouri, Warrensburg, MO 64093, USA. Previously, he served on the faculty at Texas Southern University, Houston, TX 77004. He also worked as a postdoctoral fellow at the Institute of Cyber Security, University of Texas at San Antonio, TX 78249. He is a member of the IEEE and ACM. His research interests are cyber security and formal methods.

Ravi Sandhu is the founding executive director of the Institute for Cyber Security, University of Texas at San Antonio where he holds an Endowed chair in cyber security. He is an inventor on 30 patents. He is a past editor-in-chief of the *IEEE Transactions on Dependable and Secure Computing*, a past founding editor-in-chief of the *ACM Transactions on Information and System Security*, and a past chair of the *ACM SIGSAC*. He founded ACM CCS, SACMAT, and CODASPY, and has been a leader in numerous other security conferences. His research has been focused on security models and architectures, including the seminal role-based access control models. His papers have accumulated more than 38,000 Google Scholar citations, including more than 8,000 citations for the RBAC96 paper. He is a fellow of the ACM, IEEE, and AAAS. He has received numerous awards including the IEEE Innovation in Societal Infrastructure award.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 24,2020 at 23:22:48 UTC from IEEE Xplore. Restrictions apply.