HAL
archives-ouvertes.fr

On the Decidability of the Safety Problem for Access Control Policies

Eldar Kleiner, Tom Newcomb

▶ **To cite this version:**

Eldar Kleiner, Tom Newcomb. On the Decidability of the Safety Problem for Access Control Policies. Automatic Verification of Critical Systems, Sep 2006, Nancy/France, pp.91-103. inria-00089496

HAL Id: inria-00089496
https://hal.inria.fr/inria-00089496

Submitted on 18 Aug 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the Decidability of the Safety Problem for Access Control Policies

E. Kleiner [1]

Oxford University Computing Laboratory
Oxford, OX1 3QD, UK

T. Newcomb [2]

Oxford University Computing Laboratory
Oxford, OX1 3QD, UK

Abstract

An access control system regulates the rights of users to gain access to resources in accordance with a specified policy. The rules in this policy may interact in a way that is not obvious via human inspection; there is, therefore, a need for automated verification techniques that can check whether a policy does indeed implement some desired security requirement.
Thirty years ago, a formalisation of access control presented a model and a safety specification for which satisfaction is undecidable. Subsequent research, aimed at finding restricted versions that obtain the decidability of this problem, yielded models without satisfactory expressive power for practical systems.
Instead of restricting the model, we reexamine the safety specification. We develop a new logic that can express a wide variety of safety properties over access control systems, and show that model checking is decidable for a useful fragment of this logic.

Keywords: Access control, model checking, temporal logic, CSP.

1 Introduction

Motivation. An *access control system* is a mechanism which regulates the rights of a set of users to gain access (e.g. to read or write) to some resources (e.g. secret files). This is done according to an *access control policy*, a set of rules which indicates in which circumstances a particular user may obtain a particular permission to a particular resource.

These policies can be large and dynamic, for example they might be updated every time a user or resource is created or deleted. For these reasons it is generally not clear exactly which behaviours a policy permits at any given time, and whether the policy does in fact implement some desired security specification. For example,

[1] Email: eldar.kleiner@comlab.ox.ac.uk
[2] Email: tom.newcomb@comlab.ox.ac.uk

A revised version of this paper will be electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

a system administrator would want to be sure that the policy rules do not interact in any way which allows unrestricted users to gain access to restricted resources.

There is, therefore, a need for automated tools that help an administrator assess the impact of an access control policy on the security of his system.

The Problem. In the seminal paper by Harrison, Ruzzo and Ullman [7], a formal model of access control was presented which has become known as HRU. A state of an HRU system is denoted by a set of objects, some of which are subjects, and a *protection matrix* giving the current access rights between pairs of subjects and objects. A policy is a set of commands, each parameterised by objects and specifying some possible transformation on the access matrix. Their language is able to express naturally the behaviour of real-world access control systems (e.g. UNIX).

They considered the following safety problem: given a set of policy rules, a generic access right a and an initial matrix, is it possible to reach a state in which a is granted to any subject. They then showed that this problem is undecidable using an encoding of a Turing machine tape into the matrix.

Attempted solutions. Since then, research has concentrated on finding restricted models for which this problem is decidable with minimum diminution of expressive power. For example, insisting that each command is *mono-operational* [7] (may only perform one single action, e.g. it may create an object, or grant an access right, but cannot do both in the same atomic step) or *mono-conditional* [6] (the enabling condition for each command is a single cell of the access matrix).

In [18], Sandhu and Suri introduced the *non-monotonic transformations* model, which is subsumed by [13]'s condition that allows object creation but forbids the creation of new subjects. Koch *et al.* [9] analysed a similar restriction of their graph-based access control framework.

Our observation. Although all these models obtain decidability of the HRU safety problem, unfortunately none of them is powerful enough to fully express many practical systems. This is because they gain decidability by placing restrictions on the type of policies that can be considered.

We instead reexamine the original problem in [7]. One source of undecidability is that the safety question is a property of an access control policy plus an initial matrix. It asks questions that always begin, 'From a given initial state, is it possible...' We believe that initial states are not so important when assessing the security of a policy. More common questions administrators ask about their access control systems are 'If a user doesn't have permission a, can he somehow obtain permission b?' and 'From any state with at least three users, can some user x grant a permission to some user y?' In other words, they tend to implicitly assume the system is already up and running.

Also, answering these questions can be viewed as answering the HRU safety problem for a whole range of initial states. For example, if we show that any user without right a cannot obtain right b, and if we know that the initial state of our access control system does not have a granted to any user, then we know that the system does not allow users to obtain right b.

This paper. We present a protection matrix model of access control similar

to HRU. Policy rules are expressed as parameterised commands that, provided some specified permissions meet some precondition, may create/destroy objects and grant/take permissions.

We then introduce a first-order linear temporal logic which is interpreted over finite runs of the access control system. The logic can quantify over all currently existing objects and possesses the temporal operator 'always.' The atomic propositions can make assertions about the state of particular permissions on the quantified variables or equality relationships between the variables. We call this logic *Safety Access Temporal Logic (SATL)* because it can express a variety of safety properties over access control systems.

The model checking problem for the whole logic is undecidable because it can express the HRU safety problem [7]. However, we show that the problem is decidable for a fragment of the logic which we call *Universal SATL*. Formulas in this fragment only have quantifiers $\forall$ which may only appear at the outermost level.

We have a prototype implementation of the finite abstract model used in the proof of decidability. It is written in CSP [15] for use in the model checker FDR [4]. We also give an example to illustrate the effectiveness of our approach.

Contributions. We describe a model of access control which is more expressive than that given in [7] because we allow testing for the absence of permissions, but technically simpler because we group atomic series of commands into single actions. We introduce a temporal logic that is able to express a wide range of safety properties over such models. The model checking problem for the entire logic is undecidable, but we believe this provides a framework for investigating what kinds of useful safety problems can be decided for access control policies.

This paper makes a significant start on that investigation. We present a fragment of this logic that is able to pose practical questions about access control policies such as those suggested above; another example is 'If there are at least three unprivileged users and one administrator, can the users conspire so that one of them is elevated into a privileged state?' We show that there exists an algorithm for the model checking problem over this fragment, and have built a proof-of-concept implementation.

As remarked above, this result also provides a procedure for checking the HRU safety problem for certain sets of initial states. This can be turned into an incomplete (but sound) procedure for checking the original HRU safety problem (i.e. for a single specified initial state) by attempting to find a set of initial states containing the specified state which is expressible in Universal SATL and for which the check succeeds.

Organisation. In Section 2 we present our formal model of access control. In Section 3 we introduce our temporal logic SATL, and demonstrate the undecidability of the model checking problem for the whole logic. This problem is shown to be decidable over the fragment Universal SATL in Section 4. In Section 5 we briefly describe our implementation. Conclusions, related work and future work are presented in Section 6.

2 A Model for Access Control

We assume an infinite universe of objects Σ and a finite set of access rights A. A *state* is a pair (O, M), where O is a finite subset of Σ and represents the set of currently existing objects, and M is a subset of $O \times O \times A$ representing the permissions which are currently granted. For example, a permission $(\mathsf{Frank}, \mathsf{passwd}, \mathsf{x})$ represents whether object Frank has access right x to object passwd, and might model whether a user called Frank is able to execute a program that changes his password.

An *access control policy* $\mathcal{P}$ is a finite set of commands, where a *command* $c(x_1, \ldots, x_n)$ is a 6-tuple of finite sets:

$$(c_{on}, c_{off}, c_{create}, c_{grant}, c_{take}, c_{destroy})$$

Intuitively, the first two components represent the precondition of the command (permissions that must be on and off) and the last four components represent the atomic action of the command (objects to create, permissions to grant and take, and objects to destroy). More formally, each of $c_{on}, c_{off}, c_{grant}, c_{take}$ are subsets of $F \times F \times A$, and $c_{create}, c_{destroy}$ are subsets of F. Here, F is the set of formal parameters $\{x_1, \ldots, x_n\}$, i.e. symbols which denote objects by which a command is parameterised. An *instance* of a command replaces each formal parameter in the command with a *distinct*[3] object from Σ.

We now formally specify the transition relation induced by an access control policy $\mathcal{P}$ over states as follows. $(O, M) \rightarrow (O', M')$ iff there exists some instance c of a command in the policy such that all of the following hold:

(i) The command is *applicable*:
- $c_{on} \cup c_{off} \subseteq O \times O \times A$.
  (Conditional permissions fall within the scope of the current state.)
- $c_{create} \cap O = \{\}$.
  (Objects to be created do not already exist.)
- $c_{grant} \cup c_{take} \subseteq O'' \times O'' \times A$, where $O'' = O \cup c_{create}$.
  (Permissions to be altered exist after c_{create} has been applied.)
- $c_{destroy} \subseteq O''$.
  (Objects to be destroyed exist after c_{create} has been applied.)

(ii) The *guard* of the command is met:
- $c_{on} \subseteq M$.
- $c_{off} \cap M = \{\}$.

(iii) The *next state* predicates are satisfied:
- $O' = (O \cup c_{create}) \setminus c_{destroy}$.
- $M' = ((M \cup c_{grant}) \setminus c_{take}) \cap (O' \times O' \times A)$.

Observe that a newly created object has no permissions associated with it by default, but the command may specify permissions to grant to the object in c_{grant}. Observe also that we do not include any initial state in the transition system. We take the view that a policy in fact has many possible initial states, and we allow the policy

[3] The 'distinct' restriction does not affect expressiveness. If one wanted to allow that two parameters mentioned in a command could represent the same object, one would add a similar command in which the two formal parameter names are identified.

designer to specify constraints or assumptions about initial states within our logic.

Our formalisation of access control differs from HRU in some ways that we hope simplifies our model. We make no distinction between subjects and objects: by giving a permission $(o, o, \mathsf{Subject})$ to objects o that should be regarded as subjects, one can generate the same expressive power. We also collect together all the sequential *primitive operations* that make up an atomic command in HRU and assume a command is a single structure that specifies the total effect. Finally, as it creates no extra technical overhead, we also allow commands to be enabled by permissions being *off* as well as on.

Example 2.1 We reproduce an example given in [22] called the Employee Information System. It features the employees of a company, some of whom are managers and/or directors and may award bonuses to other employees. We will use access rights Manager, Director, and Bonus.

	on	*off*	*create*	*grant*	*take*	*destroy*
$c_1(x,y)$	$(x, x, \mathsf{Director})$			(x, y, Bonus)		
$c_2(x,y)$	$''$				(x, y, Bonus)	
$c_3(x,y)$	$(x, x, \mathsf{Manager})$	$(y, y, \mathsf{Manager})$ $(y, y, \mathsf{Director})$		(x, y, Bonus)		
$c_4(x,y)$	$''$	$''$			(x, y, Bonus)	
$c_5(x,y)$	$(x, x, \mathsf{Director})$	$(y, y, \mathsf{Manager})$		$(y, y, \mathsf{Manager})$		
$c_6(x,y)$	$(x, x, \mathsf{Director})$ $(y, y, \mathsf{Manager})$				$(y, y, \mathsf{Manager})$	
$c_7(x,y)$	$(x, x, \mathsf{Manager})$		y			
$c_8(x,y)$	$(x, x, \mathsf{Manager})$	$(y, y, \mathsf{Manager})$ $(y, y, \mathsf{Director})$				y

Fig. 1. Employee Information System.

The company's policy states that directors can give out bonuses. This is expressed in Fig. 1 by the command $c_1(x, y)$ where a director x signals that he's awarded a bonus to an employee y. The command $c_2(x, y)$ shows that directors can also remove bonuses they have awarded. Similarly, $c_3(x, y)$ and $c_4(x, y)$ dictate that a manager may give or take bonuses to any employee who isn't a manager or a director. Commands $c_5(x, y)$ and $c_6(x, y)$ say that a director can demote from or promote to manager.

The original example in [22] included the ability for an employee x to appoint another employee y as his advocate. We omit this for simplicity but could easily model it by setting a permission $(x, y, \mathsf{Advocate})$. A feature not permitted in [22] but allowed in our framework is the ability to let the set of existing objects grow and shrink. Commands $c_7(x, y)$ and $c_8(x, y)$ expresses that managers may hire and fire employees.

3 Safety Access Temporal Logic

The formulas of *Safety Access Temporal Logic* (SATL) are:

$$\phi ::= \; x = y \mid x \neq y \mid (x, y, a) \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \Rightarrow \phi \mid \exists x.\phi \mid \forall x.\phi \mid \Box\phi$$

$$
\begin{array}{lll}
\pi \models o = o' & \text{iff} & o = o' \\
\pi \models (o, o', a) & \text{iff} & (o, o', a) \in M_1 \\
\pi \models \neg\phi & \text{iff} & \text{not } \pi \models \phi \\
\pi \models \phi \vee \psi & \text{iff} & \pi \models \phi \text{ or } \pi \models \psi \\
\pi \models \exists x.\phi & \text{iff} & \text{there exists some } o \in O_1 \text{ such that } \pi\restriction o \models \phi[^o/_x] \\
\pi \models \forall x.\phi & \text{iff} & \text{for all } o \in O_1 \text{ we have } \pi\restriction o \models \phi[^o/_x] \\
\pi \models \Box\phi & \text{iff} & \text{for all } i = 1, \ldots, n \text{ we have } \pi^i \models \phi
\end{array}
$$

Fig. 2. Satisfaction.

where x, y are drawn from some set of variables and a is an access right. To save ourselves exploring cases, we consider that $x \neq y$ is $\neg(x = y)$, $\phi \wedge \psi$ is $\neg(\neg\phi \vee \neg\psi)$, and $\phi \Rightarrow \psi$ is $\neg\phi \vee \psi$. We will sometime refer to formulas where variables have been instantiated with actual object names, i.e. the formula might have atomic propositions of the form $o = o'$ or (o, o', a) for $o, o' \in \Sigma$. When there are no quantifiers and all variables have been instantiated with object names we call it a *propositional formula*. Remember these are not formulas in SATL.

For a finite non-empty sequence of states $\pi = (O_1, M_1), \ldots, (O_n, M_n)$, we say π is a *path* of $\mathcal{P}$ if $(O_1, M_1) \rightarrow \cdots \rightarrow (O_n, M_n)$. We write π^i (where $i \in \{1, \ldots, n\}$) for the suffix of π starting at position i, i.e. $\pi^i = (O_i, M_i), \ldots, (O_n, M_n)$. We write $\pi\restriction o$ (where $o \in O_1$) for the longest prefix of π (possibly all of π) which has $o \in O_j$ for every state (O_j, M_j) in the prefix. *Satisfaction* between a sequence of states π and a formula is defined in Fig. 2. The notation $\phi[^o/_x]$ denotes a formula ϕ with all free occurences of x replaced by o. In this way, when satisfaction is applied to closed formulas, all variables will have been substituted with actual objects by the time they are evaluated, hence satisfaction finds object names o, o' in atomic propositions rather than variables.

Note that quantifiers range over currently existing objects only, and their temporal scope is restricted to the lifetime of the selected object. This is necessary to prevent the logic from being able to test properties of objects that do not exist, but produces a quirk of this logic: quantifiers do not distribute over the temporal operator, e.g. in general $\forall x.\Box\phi \not\equiv \Box\forall x.\phi$.

We say that a formula ϕ satisfies a policy $\mathcal{P}$, written $\mathcal{P} \models \phi$, when every path of $\mathcal{P}$ satisfies ϕ. The *model checking problem* for (some fragment of) SATL is the following: given any formula ϕ from the logic and any policy $\mathcal{P}$, does $\mathcal{P} \models \phi$? We will also require the notation $\mathcal{P}\restriction C \models \phi$ which means all paths of $\mathcal{P}$ that have all objects in the set C in continual existence satisfy ϕ.

The model checking problem for the whole of SATL is undecidable. This is because we are able to express the HRU safety problem which is shown in [7] to be reducible to the halting problem for Turing machines. The HRU safety problems asks: assuming a single fixed initial state, can a certain access right be eventually

granted. We express such problems using a SATL formula:

$$\forall x_1, \ldots, x_n.(\ \psi(x_1, \ldots, x_n) \wedge (\forall y. y = x_1 \vee \cdots \vee y = x_n) \\ \Rightarrow \Box\ \forall y_1, y_2. \neg(y_1, y_2, a)\ \ ).$$

Here, ψ expresses the exact state of the permissions between x_i variables in the initial state, and the $(\forall y \ldots)$ clause says that initially there are no other objects. The $\Box$ clause asserts that no right a should ever be granted between two objects.

4 Universal SATL

We define *Universal SATL* as the fragment of SATL which only contains the quantifier $\forall$ which may only occur at the outermost level, i.e. formulas of the form $\forall x_1, \ldots, x_n.\phi$ where ϕ is quantifier free. After looking at an example we will prove some propositions about our framework with the eventual goal of showing that the model checking problem for Universal SATL is decidable.

Example 4.1 The *manager conspiracy scenario problem* considered in [22] was: 'Can two managers conspire such that one of them gives a bonus to the other?' We can express that the scenario is not possible in Universal SATL with the formula

$$\forall x, y.(\ (x, x, \mathsf{Manager}) \wedge (y, y, \mathsf{Manager}) \wedge \neg(x, y, \mathsf{Bonus}) \wedge \neg(y, x, \mathsf{Bonus}) \\ \Rightarrow \Box\neg((x, y, \mathsf{Bonus}) \vee (y, x, \mathsf{Bonus}))\ ).$$

Our semantics for access control systems exhibits *data independence* [20] (or *parametric polymorphism*) with respect to the 'type' of objects Σ. This is because the only operation we assume on this type is equality testing. This induces a symmetry on objects which implies a bisimilarity on the transition system.

Proposition 4.2 *For all states (O, M) and (O', M') and for all bijections σ on Σ, we have*

$$(O, M) \to (O', M') \text{ iff } \sigma(O, M) \to \sigma(O', M')$$

where σ is lifted to states in the obvious way.

Proof. Straightforward from the definition of $\to$. $\Box$

We now reduce the problem involving a Universal SATL formula to a set of problems involving *propositional* SATL formulas. This allows us to henceforth restrict our attention to a single propositional formula involving object constants from a finite set $C \subset \Sigma$, and paths of the system that have objects C in continual existence.

Proposition 4.3 *The model checking problem $\mathcal{P} \models \forall x_1, \ldots, x_n.\phi$ is equivalent to the conjunction of a finite number of model checking problems of the form $\mathcal{P}\restriction C \models \psi$, where ψ is a propositional formula referring only to objects in a finite set C.*

Proof. Take any n distinct objects $C = \{o_1, \ldots, o_n\}$. We show that it is sufficient to consider the $\forall$ quantifier ranging over this finite set, thus reducing the first-order problem to a finite set of propositional problems. We can temporarily invent

notation and write

$$\mathcal{P} \models \forall x_1, \ldots, x_n.\phi \quad \text{iff} \quad \mathcal{P} \models \forall x_1 : C, \ldots, x_n : C.\phi.$$

Proving the forward direction is trivial. The backwards direction can be proved as follows. Take any path π of $\mathcal{P}$, and any objects $o'_1, \ldots, o'_n$ as instances for $x_1, \ldots, x_n$. By Proposition 4.2 we know that for every bijection σ, $\sigma\pi$ is also a path of $\mathcal{P}$, so we find such a σ that maps each o'_i into C. Assuming the right-hand side we know that $\sigma\pi \models \phi[^{\sigma o'_1,\ldots,\sigma o'_n}/_{x_1,\ldots,x_n}]$. Apply structural induction on ϕ to deduce $\pi \models \phi[^{o'_1,\ldots,o'_n}/_{x_1,\ldots,x_n}]$.

Finally, notice that $\mathcal{P} \models \forall x_1 : C, \ldots, x_n : C.\phi$ is equivalent to $\mathcal{P}\restriction C \models \forall x_1 : C, \ldots, x_n : C.\phi$ because $\forall$ only inspects paths where its instance object is in continual existence. □

To check the propositional formula on the system we employ an abstract transition system, where an abstract state records only relationships between the finite set of objects C; thus an abstract state is a subset of $C \times C \times A$ and represents any concrete state containing that exact relationship between the objects in C. Precisely, we use an abstraction function which takes a state (O, M) with $O \supseteq C$ and maps it to

$$\alpha(O, M) = M \cap (C \times C \times A).$$

Note that this mapping is not total — states that do not have all objects C in existence do not have abstractions.

We need a notion of transitions on our abstract systems. The usual requirement is that an abstract transition exists exactly when there is some corresponding concrete transition. This is not desirable as a definition because it is not directly computable: each abstract state represents an infinite number of concrete states and we cannot check them all.

Instead we use the following observation. Because our abstract states record no information about objects outside of C and because our systems are data independent with respect to objects, we can consider all objects outside of C to be homogeneous. Furthermore, each command only inspects and/or changes a finite number of objects. In a manner similar to [14], this allows us to 'collapse' the infinite set of objects to a finite set of objects, the size of which depends on the maximum number of such objects that might be required for any single transition.

This idea suggests that we compute transitions on the abstract state space by computing concrete transitions on a 'reduced' state space $(C \cup C') \times (C \cup C') \times A$. Here, C' is a subset of $\Sigma \setminus C$ of size m, where m is the the greatest number of formal object parameters in any one command (i.e. the most objects that can be 'involved' in any one transition).

To make this more formal, we define a translation from abstract states to concrete states:

$$\begin{aligned}\gamma(Q) = \{(O, M) \mid\ & O \supseteq C, O \subseteq C \cup C', \\ & M \subseteq O \times O \times A, \\ & M \cap (C \times C \times A) = Q\}.\end{aligned}$$

We define a transition $\rightarrow$ to exist between two abstract states Q and Q' exactly when there is some concrete transition from any state in $\gamma(Q)$ to any state in $\gamma(Q')$. This is computable because it deals only with finite structures. This allows us to talk about *abstract traces* $Q_1 \rightarrow \cdots \rightarrow Q_n$.

To relate the abstract and concrete systems we say that each path $(O_1, M_1) \rightarrow \cdots \rightarrow (O_n, M_n)$ of $\mathcal{P}\restriction C$ (i.e. a path of $\mathcal{P}$ with $O_i \supseteq C$ for all $i = 1, \ldots, n$) implies a *concrete trace* $\alpha(O_1, M_1) \rightarrow \cdots \rightarrow \alpha(O_n, M_n)$. We will show that the two systems have equivalent traces.[4]

Proposition 4.4 *All concrete traces are abstract traces*

Proof. We show this by proving $(O_1, M_1) \rightarrow (O_2, M_2)$ implies $\alpha(O_1, M_1) \rightarrow \alpha(O_2, M_2)$ and the result follows by induction.

Suppose the command c that generated the transition $(O_1, M_1) \rightarrow (O_2, M_2)$ was instantiated with objects $C \cup D \subseteq \Sigma$, where D is some set of objects disjoint from C. Let σ be any injection from $C \cup D$ to $C \cup C'$ which preserves C. We know such an injection exists because $|D| \leq m = |C'|$. We lift this injection to states as follows:

$$\sigma(O, M) = (\ \{\sigma(o) \mid o \in O \text{ and } o \in \operatorname{dom}\sigma\},\\ \{(\sigma(o_1), \sigma(o_2), a) \mid (o_1, o_2, a) \in M \text{ and } o_1, o_2 \in \operatorname{dom}\sigma\}\ ).$$

It now follows that $\sigma(O_1, M_1) \rightarrow \sigma(O_2, M_2)$ using the same command c except we instantiate with objects $C \cup C'$ instead of $C \cup D$. This can be seen from the definition of $\rightarrow$ by detailing cases. It can be seen also by the more informal observation that σ only removes objects which are not inspected/updated by the command instance, and renames other objects uniformly. We also have $\sigma(O_i, M_i) \in \gamma(\alpha(O_i, M_i))$ for $i = 1, 2$ as required. □

Proposition 4.5 *All abstract traces are concrete traces.*

Proof. Now let's suppose we have a sequence of abstract transitions $Q_1 \rightarrow \ldots \rightarrow Q_n$. We know that each transition exists because there is a concrete transition from some state $S_i \in \gamma(Q_i)$ to some state $S_i' \in \gamma(Q_{i+1})$. We call the command instance that creates this transition $c_i(C \cup C')$ to highlight that it is instantiated only with objects from $C \cup C'$. Our aim is to show that there is a concrete trace $(O_1, M_1) \rightarrow \cdots \rightarrow (O_n, M_n)$ with $(O_i, M_i) \in \gamma(Q_i)$ for each i.

In a reversal of the previous construction, we map the objects in C' back out into the whole of Σ, and we want to do this in such a way that each new command instance uses objects that do not overlap with those used by any other new command instance in the path. We therefore require a different injection $\sigma_i : C' \rightarrow \Sigma \setminus C$ for each command $c_i(C \cup C')$ which we use to form a new sequence of commands:

$$(O_1, M_1) \xrightarrow{c_1(C \cup \sigma_1 C')} (O_2, M_2) \quad \cdots \quad \xrightarrow{c_{n-1}(C \cup \sigma_{n-1} C')} (O_n, M_n).$$

[4] They are *not* bisimilar, a stronger equivalence between transition systems. For example, an abstract system might have an infinitely long behaviour which destroys an infinite number of objects; this could never happen in any concrete system.

We need to set up each state (O_i, M_i) in the trace in such a way that it captures the initial conditions on objects $\sigma_i(C')$ expected by each command $c_i(C \cup \sigma_i C')$ yet to be executed (we obtain these values from the states S_i) and the final conditions on objects $\sigma_i(C')$ expected by each command already executed (we obtain these values from the states S'_i). This can be done because the injections σ_i do not overlap, so the proposed states are:

$$(O_i, M_i) = Q_i \oplus \sigma_1 S'_1 \oplus \cdots \oplus \sigma_{i-1} S'_{i-1} \oplus \sigma_i S_i \oplus \cdots \oplus \sigma_{n-1} S_{n-1}.$$

Above, each injection σ_j applies to objects outside of C and permissions which are not just between C, so $\sigma_j S$ returns a structure[5] (O, M) with

$$\begin{aligned} &O \cap C = \{\}, \\ &M \cap (C \times C \times A) = \{\}, \text{ and} \\ &M \subseteq (C \cup \sigma C') \times (C \cup \sigma C') \times A. \end{aligned}$$

The operator $\oplus$ combines these structures (using union) to form a state, using Q_i to fill in the $C \times C \times A$ part.

The reader might notice that there are parts of the matrices in the final trace which are always blank (e.g. $\sigma_1 C' \times \sigma_2 C' \times A$) — they do not directly correspond to any information that could be extracted from states S_i and S'_i. While there is some freedom about what values these permissions can have, it is safest to set all these permissions off. This is because if one of these $o \in \sigma_i C'$ objects is destroyed, we would need (o, o', a) and (o', o, a) to be off in subsequent states for all objects o'. Conversely, if one was created, we'd need these permissions to be off in previous states. (Contrast this with destroying objects in C discussed below.)

It can be seen readily from the definition that $\alpha(O_i, M_i) = Q_i$. It remains to show that:

$$\begin{gathered} \underline{Q_i} \oplus \sigma_1 S'_1 \oplus \cdots \oplus \sigma_i S'_{i-1} \oplus \underline{\sigma_i S_i} \oplus \sigma_i S_{i+1} \oplus \cdots \oplus \sigma_{n-1} S_{n-1} \\ \xrightarrow[c_i(C \cup \sigma_i C')]{} \\ \underline{Q_{i+1}} \oplus \sigma_1 S'_1 \oplus \cdots \oplus \sigma_i S'_{i-1} \oplus \underline{\sigma_i S'_i} \oplus \sigma_i S_{i+1} \oplus \cdots \oplus \sigma_{n-1} S_{n-1} \end{gathered}$$

Because of the underlined terms, this can be deduced from $S_i \to S'_i$ by checking the cases in the definition of $\to$. Notice that no $o \in C$ ever appears in the destroy field of a command instance because the resulting state would not have an abstraction. This is important because otherwise we would require all permissions in the row and column of that object to be off in the subsequent state which is not guaranteed by our definition of (O_{i+1}, M_{i+1}). Otherwise the proof is straightforward. □

We now want to show that the trace equivalence between the abstract and concrete systems means that checking the abstract system will give us the same results as checking the concrete system. Note first that satisfaction can be applied to traces as well as paths.

[5] We cannot call these states, as we do not necessarily have $M \subseteq O \times O \times A$.

Proposition 4.6 *The abstract and concrete systems are indistinguishable by any propositional formula ϕ mentioning only objects in C.*

Proof. Observe that checking an atomic proposition used in ϕ on a concrete state returns the same truth value as checking it on the representative abstract state. This means that checking ϕ on a path returns the same truth value as checking ϕ on the associated trace. The result now follows from the trace equivalence of the abstract and concrete systems (Propositions 4.4 and 4.5). □

Theorem 4.7 *Model checking for Universal SATL is decidable, i.e. there exists a procedure that, given an access control policy $\mathcal{P}$ and a closed Universal SATL formula ϕ, answers whether $\mathcal{P} \models \phi$.*

Proof. Proposition 4.3 means we can instead consider the model checking problem $\mathcal{P}\restriction C \models \phi$ for a propositional formula ϕ mentioning only objects in the finite set C. By Proposition 4.6 we can checking the finite computable abstract system using traditional model checking algorithms for linear temporal logic [19] to deduce the truth of ϕ on the concrete system $\mathcal{P}\restriction C$. □

5 Automation

We can model the abstraction described in the proof of Theorem 4.7 in the process algebra CSP [15]. The CSP events represent commands instantiated by objects from $C \cup C'$. We model each permission in $(C \cup C') \times (C \cup C') \times A$ as a separate process, each able to accept events if they meet the precondition of the associated command and change state if appropriate. Processes representing permissions outside of $C \times C \times A$ do not retain state but remain in a permanent nondeterministic state in accordance with γ. All these permissions are then combined using the CSP parallel operator, and the refinement checker FDR [4] can be used to explore which states are reachable. More details are provided in [8]

Example 5.1 We complete our running example of the Employee Information System. We set $C = \{o_x, o_y\}$ to represent the two managers mentioned in the specification formula, and $C' = \{o'_1, o'_2\}$ because the maximum number of parameters in any one command is two. The abstraction was coded in CSP and checked using the refinement checker FDR. It gave a counter-example $\langle c_6(o'_1, o_x), c_3(o_y, o_x)\rangle$. We interpret this as the scenario where some director o'_1 demotes o_x; then o_y can give a bonus to o_x.

Removing c_6 from the system gives us a check that succeeds. Using our theorem, this proves the following about the Employee Information System: regardless of the number of employees, if directors cannot demote managers then it is not possible for two managers to conspire so that one of them awards the other a bonus.

6 Conclusions

Summary. We have introduced a first-order temporal logic for a protection matrix model of access control. Although the model checking problem for the whole logic is undecidable, we have identified a useful fragment of the logic for which the

problem is decidable. This allows us to check practical requirements of such systems without having to restrict the policy language.

We described how the HRU safety problem can be reduced to our decidable problem in some cases. We have also built a prototype implementation of the model checking procedure.

Related Work Guelev, Ryan and Schobbens [5] presented the RW formalism based on propositional logic for expressing access control policies and queries. The paper also presents an algorithm implemented in Prolog for calculating the ability of a fixed number of agents to achieve a certain goal in the presence of a fixed number of resources. In addition, a tool was provided which takes an RW script as input and converts the policy description into XACML [21]. Universal SATL can be compared to RW formulas which only use existential quantification.

Another related work is [2] where it is shown how access control mechanisms with a bounded number of subjects and resources can be expressed in CSP. The CSP models we use are to some extent similar.

Both these works reason only about bounded systems. They are therefore usable only for finding flaws and cannot provide general proofs of safety. Our result shows that RW formulas with only existential quantification can be checked on unbounded systems.

Future Work In practice, it is often not possible to prove security requirements like the ones we consider without also assuming that the system can never enter some inconsistent state. In such cases, one would like to 'strengthen' the check by specifying that, for example, 'there is never more than one administrator' or 'every file always has a unique owner.' Universal SATL is unable to express these invariants, but we are currently preparing decidability results about larger fragments of SATL containing formulas like

$$\forall x_1, \ldots, x_n.(\ (\phi \wedge INV) \Rightarrow \Box(\psi \wedge INV)\ ),$$

where INV expresses a desired global invariant of the system via restricted use of quantifiers.

The results presented in this paper were possible thanks to the fact that the model is *data-independent* with respect to the type of objects. This observation allowed us to use techniques developed for data-independent systems with arrays [10,14] and apply them to access control matrices. There are other results about array systems that we believe can be leveraged to analyse access control policies, in particular results about arrays with *reset* [16,11].

Writing CSP scripts for analysing access control polices manually might be tedious and error-prone. We therefore intend to develop a compiler which will produce CSP scripts from a more abstract description.

We are also interested in RBAC (role based access control) which are Turing-complete [3]. We believe our decidability results can be extended to model some RBAC polices in which the system is limited to a fixed number of roles but unrestricted otherwise.

Lastly, we hope that our results can shed more light on verification of Trust Management systems [1]. The results we present here, together with knowledge

gained in the security protocols field [17], can be combined in order to reason about such systems and hopefully strengthen existing decidability results [12].

References

[1] Blaze, M., J. Feigenbaum and J. Lacy, *Decentralized trust management*, in: *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1996, pp. 1081–6011.

[2] Bryans, J., *Reasoning about XACML policies using CSP*, Technical Report CS-TR-924, University of Newcastle (2005).

[3] Crampton, J., "Authorization and antichains," Ph.D. thesis, Birkbeck, University of London, London, England (2002).

[4] Formal Systems (Europe) Ltd, "Failures-Divergence Refinement: FDR2 User Manual," (1999). URL http://www.fsel.com

[5] Guelev, D., M. Ryan and P.-Y. Schobbens, *Model-checking access control policies.*, in: *ISC*, 2004, pp. 219–230.

[6] Harrison, M. and W. Ruzzo, *Monotonic protection systems*, in: A. J. R. DeMillo, D. Dobkin and R. Lipton, editors, *Foundations of Secure Computation*, Academic Press, 1978 pp. 337–363.

[7] Harrison, M., W. Ruzzo and J. Ullman, *Protection in operating systems*, Communications of the ACM (1976).

[8] Kleiner, E. and T. Newcomb, *Using CSP to decide safety problems for access control policies*, Technical Report RR-06-04, Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK (2006).

[9] Koch, M., L. Mancini and F. Parisi-Presicce, *Decidability of safety in graph-based models for access control*, in: *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security* (2002), pp. 229–243.

[10] Lazić, R., T. Newcomb and A. Roscoe, *On model checking data-independent systems with arrays without reset*, Theory and Practice of Logic Programming: Special Issue on Verification and Computational Logic **4** (2004).

[11] Lazić, R., T. Newcomb and A. Roscoe, *Polymorphic systems with arrays, 2-counter machines and multiset rewriting*, in: *Proceedings of INFINITY*, 2004, pp. 3–19.

[12] Li, N., J. Mitchell and W. Winsborough, *Beyond proof-of-compliance: security analysis in trust management*, J. ACM **52** (2005), pp. 474–514.

[13] Lipton, R. and L. Snyder, *On synchronization and security*, in: A. J. R. DeMillo, D. Dobkin and R. Lipton, editors, *Foundations of Secure Computation*, Academic Press, 1978 pp. 367–385.

[14] Newcomb, T., "Model Checking Data-Independent Systems With Arrays," Ph.D. thesis, Oxford University Computing Laboratory (2003).

[15] Roscoe, A., "The Theory and Practice of Concurrency," Prentice-Hall, 1998.

[16] Roscoe, A. and R. Lazić, *What can you decide about resetable arrays?*, in: *Proceedings of the 2nd International Workshop on Verification and Computational Logic (VCL 2001), Technical Report DSSE-TR-2001-3, pages 5–23* (2001), pp. 5–23.

[17] Ryan, P., S. Schneider, M. Goldsmith, G. Lowe and A. Roscoe, *Modelling and analysis of security protocols* (2001).

[18] Sandhu, R. and G. Suri, *Non-monotonic transformation of access rights*, in: *Proceedings of the IEEE Symposium on Security and Privacy*, 1992, pp. 148–163.

[19] Vardi, M. and P. Wolper, *An automata-theoretic approach to automatic program verification (preliminary report)*, in: *Proc. 1st Annual IEEE Symposium on Logic in Computer Science*, Washington, DC, 1986, pp. 332–344.

[20] Wolper, P. and V. Lovinfosse, *Verifying properties of large sets of processes with network invariants*, in: *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science **407** (1989), pp. 68–80.

[21] Zhang, N., M. Ryan and D. Guelev, *Synthesising verified access control systems in XACML*, in: *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering* (2004), pp. 56–65.

[22] Zhang, N., M. Ryan and D. Guelev, *Evaluating access control policies through model checking.*, in: *ISC*, 2005, pp. 446–460.