**Programming Research Group**

# USING CSP TO DECIDE SAFETY PROBLEMS FOR ACCESS CONTROL POLICIES

E. Kleiner and T. Newcomb

PRG-RR-06-04

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

**Abstract**

For the last three decades, since the seminal paper of Harrison *et al.* [17], it appeared that formal verification of access control mechanisms might not be feasible. Their work was the first to formalise safety analysis of such systems and showed it is undecidable under a model commonly known as HRU. Research, aiming to find restricted versions of HRU that gain the decidability of this problem, yielded models without satisfactory expressive power for practical systems.

We introduce a new protection model which subsumes HRU, giving it semantics informally and in CSP. In addition, we introduce new safety properties and show that, though in terms of security they are stronger properties than the one defined under HRU, they can be automatically decided under our model and thus under HRU.

## 1 Introduction

For the last three decades, researchers have employed the *hru safety*[1] problem definition for access control polices as was formalised by Harrison, Ruzzo and Ullman in [17] under a *protection matrix* model commonly referred to as HRU (see Sect. 4 for more details).

Given a set of policy rules, a generic access right $a$ and an initial matrix $M_0$; the hru safety problem is the question of whether it is possible to reach a state in which $a$ is granted to a subject that had not previously held it. That is, is there any sequence of administrative commands that *leaks* $a$. If no such sequence exists, we say that $M_0$ does not *leak* $a$.

It was subsequently shown that the problem of determining hru safety is in general undecidable. However, under appropriate conditions decidability is achieved and indeed the *mono-operational commands* condition was suggested as such. Since then, research has been put into finding restricted models for which this problem is decidable with minimum diminution of the expressive power. Harrison *et al.* [16] found that the hru safety problem for *monotonic* protection systems, where all the administrative command are *mono-conditional*, is decidable.

Lipton and Snyder [14] proved that protection systems whose commands do not use the *create subject* operation, regain the decidability of the problem. In [24], Sandhu and Suri introduced their model of non-monotonic transformations which is implied by [14].

Koch *et al.* [10] analysed similar safety property of their graph-based access control framework. They showed that it is decidable if each graph rule either adds or deletes a graph structure but does not do both. Since graph rules only allow adding subjects from a predefined finite set, this restriction is slightly stronger than [14]'s.

Unfortunately, none of these models is powerful enough to fully express many practical systems. Therefore, in this paper we define and investigate stronger safety properties.

We present a new protection model, KN, which can simulate any HRU *protection system* augmented with the ability to test for the absence of rights.

We then consider a new safety problem KN1. This can be interpreted as follows: 'Given a set of policy rules, a generic access right $a$ and a matrix $M$ specifying permissions

[1]This original term used in [17] was safety. We want to distinguish this problem from other safety properties and therefore use the term *hru safety* instead.

on some finite set of objects, does there exist a reachable state from any matrix *consistent* with $M$ that leaks $a$?' By consistent, we mean the initial matrix must agree with $M$ all permissions in $M$. We can think of this safety property as only partially specifying the initial matrix.

In practice, it is often not possible to prove useful security requirements without also assuming that the system can never be in 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.'

For this reason, we also consider the KN2 safety problem which modifies the KN1 problem by introducing a set of simple invariants such as 'for a given object $o$ no other object $o'$ has permission $(o, o', a)$ set.'

The main contributions we make in this paper are:

- We show that, although in terms of security the safety problems KN1 and KN2 are more rigorous than the hru safety property, they can be decided using finite abstractions in the FDR model checker [15].
- We observe that if a protection system $\mathtt{S}$ satisfies KN1 or KN2 for a generic right $a$ and any initial state that includes $M_0$, then it is also (hru) safe for $\mathtt{S}$, $a$ and $M_0$. We can therefore propose a fail-safe algorithm for deciding whether an initial state[2] of a given protection system under HRU (or KN) is (hru) safe. By fail-safe we mean that if the algorithm finds no leakage then this state is safe but it might find "false" leakages. However, since a leakage is a result of a finite sequence of actions, it is easy to check whether a leakage returned by the algorithm is "false" or not.

The models and algorithms presented in this paper were possible thanks to the fact that the HRU model is *data-independent* with respect to $S$ and $O$. In other words, the only operations a system can perform over members of $S$ and $O$ are: input, store, equality testing and some polymorphic operation such as tupling. This observation allowed us to use the decidability results of data-independent systems with arrays without reset [11].

Our paper is structured as follows. In Sect. 2 we introduce the KN model and ensure it is data independent in Sect. 3. Section 4 investigates the relationship between HRU and KN. In Sect. 5 we formalise three variations of the safety problem under KN and prove their decidability/undecidability in Sect. 6. We conclude and discuss related work in Sections 7 and 8. Lastly, we describe our future work plans in Sect. 9. APPENDIX A provides a brief overview of CSP, the syntax we use to formalise the models presented in this paper.

## 2 The KN Access Control Model

In this section we introduce our protection model, giving semantics informally and in CSP [9].

[2]In [17] the authors use the term *configuration* rather then *state*.

We assume a finite set of *access rights* $A$ and an infinite domain for *objects* $\Sigma$. This generates the set of *permissions* $P = \Sigma \times \Sigma \times A$. We may write such a permission $p$ as $(p_x, p_y, p_a)$ which will model whether object $p_x$ has access right $p_a$ to $p_y$.

A *matrix* is a pair $(O, M)$ of a finite set $O \subset \Sigma$, and a function

$$M : O \times O \times A \to \mathbb{B}.$$

A matrix denotes whether permissions within the domain of $M$ are on or off; permissions outside the domain of $M$ are off.

A system itself is defined by a set $T$ of *transitions*. A transition $t$ is a finite tuple of the form

$$t = (t_{on}, t_{off}, t_{reset}, t_{grant}, t_{take}),$$

where each of $t_{on}$, $t_{off}$, $t_{grant}$, and $t_{take}$ are subsets of $P$, and $t_{reset}$ is a subset of $\Sigma$. Transitions must satisfy the following property:

$$t_{on} \cap t_{off} = t_{grant} \cap t_{take} = \{\}.$$

A transition is intended to be interpreted as follows:

- If all the permissions in $t_{on}$ are on and all the permissions in $t_{off}$ are off, then the transition is enabled and may be executed.

- If the transition is executed, every permission involving anything from the set $t_{reset}$ is set to off, unless the permission is mentioned in the set $t_{grant}$. All the permissions in $t_{grant}$ are set to on, and all the transitions in $t_{take}$ are set to off, regardless of their state before.

Note that a transition can never grant an infinite number of permissions. This ensures that the resulting state is expressible as a finite matrix.

When coding our access control model KN into CSP, we make the set of events equal to the set of transitions. Each permission is modelled as a single process: when it is in the on state it blocks transitions that demand that it is off and turns itself off after a transition that asks so:

$$\begin{array}{l} \mathit{PERMISSION}_{ON}(p) = \\ \quad \Box_{t \in T}\, p \notin t_{off} \;\&\; t \to \big( \begin{array}{l} \mathit{PERMISSION}_{OFF}(p) \\ \blacktriangleleft p \in t_{take} \;\vee\; p_x \in t_{reset} \;\vee\; p_y \in t_{reset} \blacktriangleright \\ \mathit{PERMISSION}_{ON}(p)\; \big) \end{array} \end{array}$$

The definition of $\mathit{PERMISSION}_{OFF}(p)$ is similar.

We will put the entire $\Sigma \times \Sigma \times A$ matrix together using CSP parallel composition. (The composition is infinite, but this will not cause us any problems mainly because we will use only the traces model of CSP.) At this point we need to consider what the initial matrix of the system is — we will come to this later.

**Example 1.** *We reproduce an example given in [29] called the Employee Information System (EIS). It features the employees of a company, some of whom are managers and/or directors and may award bonuses to other employees.*

*We can model these objects in our* KN *framework. We will have access rights* Exists *and* Subject *that may be present at the diagonal to mark whether an objects exists, and if it does, whether it is an employee (subject) or a bonus. (A similar mechanism is used to capture HRU within* KN *— see Sect. 4.) We will have additional access rights* Manager, Director, *and* HasBonus.

| | *on* | | *off* | *reset* | *grant* | *take* |
|---|---|---|---|---|---|---|
| $t_1(x, a, b)$ | $(x, x,$ Exists$)$<br>$(x, x,$ Subject$)$<br>$(x, x,$ Director$)$ | $(a, a,$ Exists$)$<br>$(a, a,$ Subject$)$<br>$(b, b,$ Exists$)$ | $(b, b,$ Subject$)$ | | $(a, b,$ HasBonus$)$ | |
| $t_2(x, a, b)$ | " | " | " | | | $(a, b,$ HasBonus$)$ |
| $t_3(x, a, b)$ | $(x, x,$ Exists$)$<br>$(x, x,$ Subject$)$<br>$(x, x,$ Manager$)$ | (a, a, Exists)<br>(a, a, Subject)<br>(b, b, Exists) | $(a, a,$ Manager$)$<br>$(a, a,$ Director$)$<br>$(b, b,$ Subject$)$ | | $(a, b,$ HasBonus$)$ | |
| $t_4(x, a, b)$ | " | " | " | | | $(a, b,$ HasBonus$)$ |
| $t_5(x, a)$ | $(x, x,$ Exists$)$<br>$(x, x,$ Subject$)$<br>$(x, x,$ Director$)$ | $(a, a,$ Exists$)$<br>$(a, a,$ Subject$)$ | $(a, a,$ Manager$)$ | | $(a, a,$ Manager$)$ | |
| $t_6(x, a)$ | $(x, x,$ Exists$)$<br>$(x, x,$ Subject$)$<br>$(x, x,$ Director$)$ | $(a, a,$ Exists$)$<br>$(a, a,$ Subject$)$<br>$(a, a,$ Manager$)$ | | | | $(a, a,$ Mananger$)$ |
| $t_7(x)$ | $(x, x,$ Exists$)$<br>$(x, x,$ Subject$)$ | $(x, x,$ Manager$)$ | | | | $(x, x,$ Mananger$)$ |
| $t_8(x, n)$ | $(x, x,$ Exists$)$<br>$(x, x,$ Subject$)$<br>$(x, x,$ Manager$)$ | | $(n, n,$ Exists$)$ | $n$ | $(n, n,$ Exists$)$<br>$(n, n,$ Subject$)$ | |

Table 1: Employee Information System.

*The company's policy states that directors can give out bonuses. This is expressed in Fig. 1 by transitions of the form $t_1(x, a, b)$ where a director $x$ gives a bonus $b$ to an employee $a$. Transitions of the form $t_2(x, a, b)$ show that the director can also remove bonuses. Similarly, $t_3(x, a, b)$ and $t_4(x, a, b)$ dictate that a manager may give or take bonuses to any employee who isn't a manager or a director. Transitions $t_5(x, a)$ and $t_6(x, a)$ say that a director can demote from or promote to manager, and $t_7(x)$ says that a manager may resign.*

*The original example in [29] included the ability for an employee $s_1$ to appoint another employee $s_2$ as his advocate. We omit this for simplicity but could easily model it, for example by setting a permission $(s_1, s_2,$ Advocate$)$. A feature not permitted in [29] but allowed in our framework is the ability to let the set of existing objects grow. $t_8(x, n)$ expresses that managers may hire new employees. We could similarly add transitions that fire employees and that create/delete bonuses.*

*The set of transitions $T$ is formed by taking the transitions from Fig. 1 and instantiating $x, a,$ and $b$ with all possible values from $\Sigma$. The variable $n$ is chosen slightly differently as explained in Sect. 3.*

## 3 Data Independence

In the literature a system is data-independent with respect to a type $\Sigma$ if the only operation that can be performed on values of that type is equality. This property has been exploited to prove the decidability of many useful classes of model checking problems [19, 22, 27]. In this section we ensure our KN systems have this property by giving a semantic restriction and a sufficient syntactic condition.

We also allow a finite number of constants $O_0 \subseteq \Sigma$ from the type to be used which will let us specify the initial states of some permissions and check how they evolve over time. We will write $\overline{O_0}$ for $\Sigma \setminus O_0$ and restrict KN as follows:

1. Transitions cannot refer to objects in $\overline{O_0}$ explicitly by name. More precisely, if some objects in $O_0$ can perform a transition with, say, $n$ objects from $\overline{O_0}$, then they could also have performed it with any other $n$ objects from $\overline{O_0}$, so long as they also met the enabling conditions of the transition.

2. Objects in $O_0$ cannot be reset.

The second condition is necessary because without it a system would be able to refer directly to a particular row/column of the matrix which has been reset. Safety problems of such systems are harder [21] but it is future work to try and relax this condition (see Sect. 9.)

In order to formalise this definition, we will require a little mathematical machinery. We can lift a function $\varphi : \overline{O_0} \to S$ to transitions in the expected way: $\varphi t$ is the same as $t$ except $\varphi$ has been applied to everything from $\overline{O_0}$ appearing in $t$. Everything else in $t$, including objects in $O_0$, remain unchanged. We write $\varphi : \overline{O_0} \overset{\leftrightarrow}{\to} \overline{O_0}$ to mean that the function $\varphi$ is a bijection on the set $\overline{O_0}$.

Now we can state our definition of data independence more formally:

1. $T$ is closed with respect to all bijections $\varphi : \overline{O_0} \overset{\leftrightarrow}{\to} \overline{O_0}$.

2. For all $t \in T$, $t_{reset} \cap O_0 = \{\}$.

These conditions can be enforced by requiring that $T$ is defined as a union of set comprehensions in the following form:

$$
\begin{aligned}
\{\ & t_1(o_1, ..., o_n, o'_1, ..., o'_m), \quad \ldots, \quad t_l(o_1, ..., o_n, o'_1, ..., o'_m) \mid \\
& \quad o'_1 \leftarrow \overline{O_0}, \\
& \quad o'_2 \leftarrow \overline{O_0}, o'_2 \notin \{o'_1\}, \\
& \quad o'_3 \leftarrow \overline{O_0}, o'_3 \notin \{o'_1, o'_2\}, \\
& \qquad \vdots \\
& \quad o'_m \leftarrow \overline{O_0}, o'_m \notin \{o'_1, \ldots, o'_{m-1}\}\ \}
\end{aligned}
$$

where $o_1, \ldots, o_n \in O_0$, and each $t_i(\cdots)$ can only refer to objects using its parameters, and the sets $t_i(\cdots)_{reset}$ must not contain $o_1, \ldots, o_m$. In fact, we can relax these conditions: for example, it is permissible to miss out some or all of the $\notin$ constraints above, or to draw $o'_1, \ldots, o'_m$ from supersets of $\overline{O_0}$

**Example 2.** *In the EIS example (Example 1) no constants were used in the transitions which on the face of it means that the set $O_0$ does not need to contain any objects. However, we will want to mention objects in our safety property so $O_0$ will not be empty (see Example 4.)*

*We explained that the variables $x$, $a$ and $b$ are drawn from the set $\Sigma$. To satisfy our definition of data independence we should draw values for $n$ from the set $\overline{O_0}$. The fact that $n$ cannot take values in $O_0$ is not a problem as the matrix is always finite, the set of objects $\overline{O_0}$ contains an infinite supply of non-existing objects that $t_8(\cdots)$ can turn into existing objects.*

## 4 The Relationship Between KN and HRU

In this section we show how HRU [17] can be modelled in KN.

Conversely to KN, HRU identifies $S \subseteq O$ as the set of subjects. Therefore a state or a *configuration* in HRU is the triple $(S, O, M)$ of finite sets $S \subseteq O \subset \Sigma$ and a function $M : S \times O \times A$. A command (transition) in HRU takes the form:

```
command $\alpha(X_1, X_2, \ldots, X_n)$
  if $a_1$ in $(X_{s_1}, X_{o_1})$ and
    ⋮
    $a_m$ in $(X_{s_m}, X_{o_m})$
  then
    $op_1$
    ⋮
    $op_n$
end
```

Where each each $op_i$ is a *primitive operations* which can enter and delete rights as well as create and destroy subjects and objects.

**Example 3.** *This example illustrates how the EIS system introduced in Example 1 can be modelled using HRU with a slight adjustment. Since HRU cannot test for the absence of a right we introduce a new guard of the form "a* **notin** *$(X, X')$".*

$(s, s, \mathsf{Director})$ *and* $(s, s, \mathsf{Manager})$ *will indicate that a subject is a director or a manager respectively. As we cannot model objects' type by including a special right in the diagonal, we employ an additional access right* Types *ensuring that only one special-purpose subject in the initial matrix is granted the permission* $(s', s', \mathsf{Types})$. *The permission* $(s', o, \mathsf{Bonus})$ *is subsequently used to indicate that object $o$ is of type* Bonus.

We can model an HRU state in KN by adding two new attributes Exists and Subject to $A$. We include an attribute $(o, o, \mathsf{Exists})$ for objects $o$ that exist, and all other objects can be considered not to exist. Of the existing objects, another attribute $(o, o, \mathsf{Subject})$ could mark whether or not this object is a subject or not.

An HRU command can be simulated as follows:

```
• command $t_1(s, x, a, b)$
    if
      Types in $(s, s)$ and
      Bonus in $(s, b)$ and
      Director in $(x, x)$
    then
      enter HasBonus into $(a, b)$
  end
```

```
• command $t_2(s, x, a, b)$
    if
      Types in $(s, s)$ and
      Bonus in $(s, b)$ and
      Director in $(x, x)$
    then
      delete HasBonus from $(a, b)$
  end
```

```
• command $t_3(s, x, a, b)$
    if
      Types in $(s, s)$ and
      Bonus in $(s, b)$ and
      Manager in $(x, x)$ and
      Manager notin $(a, a)$ and
      Director notin $(a, a)$
    then
      enter HasBonus into $(a, b)$
  end
```

```
• command $t_4(s, x, a, b)$
    if
      Types in $(s, s)$ and
      Bonus in $(s, b)$ and
      Manager in $(x, x)$ and
      Manager notin $(a, a)$ and
      Director notin $(a, a)$
    then
      delete HasBonus from $(a, b)$
  end
```

```
• command $t_5(x, a)$
    if
      Director in $(x, x)$
    then
      enter Manager from $(a, a)$
  end
```

```
• command $t_6(x, a)$
    if
      Director in $(x, x)$ and
      Manager in $(a, a)$
    then
      delete Manager from $(a, a)$
  end
```

```
• command $t_7(x)$
    if
      Manager in $(x, x)$
    then
      delete Manager from $(x, x)$
  end
```

```
• command $t_8(x, n)$
    if
      Manager in $(x, x)$
    then
      create subject $n$
  end
```

Table 2: Modelling the EIS system in HRU commands

- We can encode conditions using $t_{on}$, although we would want to add to $t_{on}$ the permission $(o, o, \mathsf{Exists})$ for any object $o$ mentioned, and both $(s, s, \mathsf{Exists})$ and $(s, s, \mathsf{Subject})$ for any subject $s$ mentioned.

- The HRU primitive operations enter and delete can be modelled using the sets $t_{grant}$ and $t_{take}$.

- We can destroy an object by removing its Exists permission: i.e. we put $(o, o, \mathsf{Exists})$ in $t_{take}$. Similarly for subjects.

- We can create an object by taking one of the infinitely many $o \in \overline{O_0}$ which doesn't exist, clearing all its entries, and making it exist. We do this by having a transition $t(o, \ldots)$ for each $o \in \overline{O_0}$, and putting $(o, o, \mathsf{Exists})$ in $t_{off}$, $o$ in $t_{reset}$, and $(o, o, \mathsf{Exists})$ in grant. It is necessary to reset all permission involving $o$ in case $o$ was an object that previously existed and its entries contain junk. For subjects, we also grant $(o, o, \mathsf{Subject})$.

As HRU commands are parameterised by the subjects and objects they refer to, they create data-independent sets of transitions. (Notice how we ensured that objects in $O_0$ are not reset by only allowing objects in $\overline{O_0}$ to be created.)

In the HRU model, states are finite so we can encode them as a KN state $(O, M)$. In [17] the initial state of an HRU system is not included in the model, rather it is presented as part of the safety property. We have separated our notion of model and property in the same manner for comparison.

Notice that KN systems are more expressive than HRU systems in the following way: whereas HRU conditions can only insist that certain permissions are on, we are able to test whether permissions are off via the $t_{\mathit{off}}$ component of transitions.

There is one slight difference between the two frameworks in the way that safety properties can be expressed over the models: an HRU system may grant and take the same permission in the same command to model a temporary access right that can be detected by the safety property. We cannot do this in KN because our transitions are atomic, but can easily find another way to signal this, e.g.: permanently set a fresh access right $a$, while changing the safety property to look for $a$ instead.

## 5 Safety

As mentioned in Sect. 3, we are interested in a specific finite set of objects $O_0$ because we wish to impose some initial constraints upon them or because we wish to observe how these objects evolve. We call $O_0 \times O_0 \times A$ the observable permissions. If the system possesses some kind of symmetry (as it is bound to if it is data independent), we can assume that some or all of these observable objects can represent arbitrary objects without loss of generality.

We allow the initial state of the observable permissions to be completely specified by some *initial observable matrix* $(O_0, M_0)$, and we will only consider safety properties that talk about the observable permissions (hence the name 'observable'), for example 'is a certain observable permission ever granted?'

**Example 4.** *The* manager conspiracy scenario *considered in [29] was: 'can two managers conspire such that one of them gets a bonus?' Because of the symmetry in our model, if such a scenario is possible then (without loss of generality) a manager $S2$ will give a bonus $B1$ to a manager $S1$. We therefore consider an initial state with $O_0 = \{S1, S2, B1\}$ that models two managers and a bonus:*

| $M_0$ | $S1$ | $S2$ | $B1$ |
|---|---|---|---|
| $S1$ | Exists Subject Manager | | |
| $S2$ | | Exists Subject Manager | |
| $B1$ | | | Exists |

*Of course, such a problem is finite in the case that (1) we consider a specific initial state and (2) we don't allow the number of objects to grow beyond a certain bound. But the property we are checking makes no assumptions about the rest of the initial state and we have included transitions that allow objects to be added to the system, so traditional model checking procedures are not applicable.*

The question remains: what is the initial state of the unobservable permissions? We consider three variants which we call KN1, KN2, and KN$\perp$.

## 5.1 KN1

For our first variant KN1 we are interested in proving properties of the KN system independently of the unobservable permissions' initial states. One can view such a class of initial states pictorially as in Fig. 1.

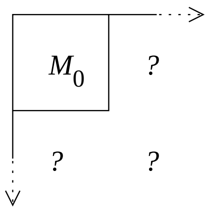

Figure 1: KN1.

We can encode a system starting with an arbitrary assignment to the unobservable permissions in CSP as follows:

$$\begin{aligned}\mathrm{KN1} \;=\; \underset{T}{\|}_{\,p\in P} \;\; & \big(\mathit{PERMISSION}_{ON}(p) \lessdot M_0(p) \gtrdot \mathit{PERMISSION}_{OFF}(p)\big) \\ & \qquad\qquad \lessdot p \in O_0 \times O_0 \times A \gtrdot \\ & \big(\mathit{PERMISSION}_{ON}(p) \sqcap \mathit{PERMISSION}_{OFF}(p)\big).\end{aligned}$$

KN1 does not enforce the restriction that there are only finitely many permissions initially on; we require this so that the system does not start with infinitely many users (remember we use permissions to mark when objects come into existence.) This is cumbersome to express in CSP and in fact we can assume it anyway. Because each transition can only reveal that a finite number of permissions are on, the finite-trace models of CSP we will be using can never yield a behaviour where infinitely many permissions must be initially on.

**Example 5.** KN1 *is a perfect model for the property discussed in Example 4. In this case,* KN1 *can nondeterministically initialise in any state satisfying the following property: there are a finite number of employees, but at least two managers, and there is at least one bonus which is not held by either of the aforementioned managers. For example, there is no constraint on the number of directors or other managers present in the company.*

## 5.2 KN2

In practice it is useful to consider the question marks from Fig. 1 to be constrained so that they are guaranteed not to start in particular states. We therefore assume a set of *safe permissions* $SP$ which are the only rights that may be initially held.

We will assume that these safe permissions are a desired invariant of the system, in that not only does the matrix start with just safe permissions granted, but it also should never grant non-safe permissions. For example, the safe permissions may be those that the administrator is willing to give out, and it would be considered an error if any non-safe permission could be granted. Therefore we will answer questions about these system of the following form: (1) can a non-safe permission be granted anywhere? (2) if not, is some safety problem about the observable permissions true?

As the matrix $(O_0, M_0)$ can completely specify the initial state of observable permissions, we assume that all observable permissions are safe. Also, in order to gain decidability results, it is necessary that these permissions respect the data-independence property. We therefore demand that for any permission $p \in SP$, renaming $\overline{O_0}$ objects in $p$ gives us another permission in $SP$. Formally speaking we can say:

1. $O_0 \times O_0 \times A \subseteq SP$, and
2. $SP$ is closed with respect to all bijections $\varphi : \overline{O_0} \overset{\leftrightarrow}{\to} \overline{O_0}$.

It is possible to write down a syntactic condition for such properties as was done for transitions in Sect. 3.

We encode the class of systems starting in such a state in CSP as follows.

$$\begin{aligned}
\mathrm{KN2} = \underset{T}{\|}_{p \in P} \quad & \big(\mathit{PERMISSION}_{ON}(p) \lhd M_0(p) \rhd \mathit{PERMISSION}_{OFF}(p)\big) \\
& \lhd p \in O_0 \times O_0 \times A \rhd \\
& \big(\mathit{PERMISSION}_{ON}(p) \sqcap \mathit{PERMISSION}_{OFF}(p) \\
& \quad \lhd p \in SP \rhd \mathit{PERMISSION}_{OFF}(p)\big).
\end{aligned}$$

The same caveat about the initial matrix being finite that applied to KN1 also applies to KN2.

**Example 6.** *We might consider the safety problem from Example 5 with the following additional constraint: suppose that there are initially no directors present in the company. We can add this constraint by setting the set of safe permissions to*

$$SP = (\Sigma \times \Sigma \times A) \setminus \{(o, o, \mathsf{Director}) \mid o \in \overline{O_0}\}.$$

*Our* KN2 *model will be able to model the company starting from a state where there are no directors or only one. We might then wish to check: (1) can a director be appointed? and (2) if not, is the manager conspiracy scenario possible.*

## 5.3 KN⊥

In our final variant KN⊥ we consider all the unobservable permissions to be off. This model bears the closest resemblance to the hru safety property [17]. We can code this in CSP as follows:

$$\begin{aligned} \mathrm{KN}\bot \;=\; \mathop{\|}_{T}{}_{p \in P} \quad & \mathit{PERMISSION}_{ON}(p) \\ & \nless\; p \in O_0 \times O_0 \times A \;\wedge\; M_0(p) \;\ngtr \\ & \mathit{PERMISSION}_{OFF}(p) \end{aligned}$$

**Example 7.** *If we use the matrix $(O_0, M_0)$ from Example 4 for the Employee Information System,* KN⊥ *will provide an accurate model of the company starting with* exactly *two managers and one bonus with initial state as prescribed by $M_0$.*

## 5.4 Relationships

It can be noted that KN1 and KN2 are abstractions of KN⊥ in the following sense:

**Theorem 8.** KN1 $\sqsubseteq_T$ KN2 $\sqsubseteq_T$ KN⊥.

*Proof.* By resolving nondeterminism within the definitions of KN1 and KN2. □ □

This theorem is important because of the following property of refinement: if $P \sqsubseteq_T Q$ and we show that all behaviours of $P$ satisfy some property $S$, then we know that all behaviours of $Q$ satisfy $S$ too.

None of the above CSP models can be model-checked using an explicit state exploration tool like FDR because the domain of objects $\Sigma$ is infinite. Using similar techniques that have previously been applied to non-resettable arrays [11], we are able to relate some of these processes to finite abstractions.

## 6 Decidability and Undecidability Results

In this section we examine the decidability of safety problems under our models KN1, KN2, and KN⊥.

### 6.1 Decidability of KN1 Safety Problems

As noted, KN1 is an infinite state system and therefore not amenable to traditional explicit state model-checking procedures. Here, we describe and motivate an abstraction AKN1 of KN1 that is finite state. We then relate the behaviours of KN1 and AKN1.

Recall that we are only interested how the permissions in the observable portion $O_0 \times O_0 \times A$ of the matrix evolve in KN1. We now argue that the enabledness of any transition is effectively independent of the state of all permissions outside of $O_0 \times O_0 \times A$. This is because of the following observations:

1. The observable portion of the matrix cannot distinguish by name different objects in $\overline{O_0}$ due to the data-independence condition.

2. After any finite run of the system, there will always remain an infinite number of objects in $\overline{O_0}$ who's initial state is as yet undetermined.

3. We can choose the state of these undetermined objects — we are considering all possible initial states for the unobservable permissions, so it is entirely possible that the objects in $\overline{O_0}$ were initialised in that state from the start.

The first two observations allows us to substitute objects in $\overline{O_0}$ for objects whose state is undetermined. The third observation means we can choose the states of these objects in such a way that enables the transition. This argument suggests that the evolution of the observable portion of the matrix is unimpeded by the unobservable portion. We therefore choose an abstraction that models the observable portion only:

$$\text{AKN1} = \underset{T}{\|}_{\,p \in O_0 \times O_0 \times A} \quad \begin{array}{l} \mathit{PERMISSION}_{ON}(p) \\ \lessdot M(p) \gtrdot \mathit{PERMISSION}_{OFF}(p). \end{array}$$

Note that although KN1 is finite state, each state may have an infinite number of transitions from it because $T$ is infinite. A model checker like FDR cannot handle such systems. We can, however, use standard data-independence techniques [22] to reduce $\Sigma$ (and therefore $T$) to a finite set. Such an argument would run as follows.

As AKN1 cannot distinguish between different objects in $\overline{O_0}$ we might as well assume that $\overline{O_0}$ is a singleton set. Therefore, the set of objects in the abstract model becomes $\Sigma_X = O_0 \cup \{X\}$. The symbol $X$ can be thought of as meaning 'some object in $\overline{O_0}$,' and can mean different objects even within the same transition.

Carrying on, the set $T$ is reduced to a finite set $T_X$ by replacing occurrences of objects in $\overline{O_0}$ with $X$. This can be done by applying the function $\varphi_X : \overline{O_0} \to \{X\}$ to each transition. For our sufficient syntactic condition for data independence, we just replace $\overline{O_0}$ with $\{X\}$. We use $T_X$ in place of $T$ in the definitions of $\mathit{PERMISSION}_{ON}(p)$ and $\mathit{PERMISSION}_{OFF}(p)$.

We now discuss formally how the behaviours of KN1 and AKN1 are related.

**Theorem 9.** *Let* $\langle t_1, \ldots, t_n \rangle \in \mathit{traces}\,[\![\text{KN1}]\!]$. *Then*

$$\langle \varphi_X\, t_1, \ldots, \varphi_X\, t_n \rangle \in \mathit{traces}\,[\![\text{AKN1}]\!].$$

*Proof.* We use induction on $i = 0, \ldots, n$ to show:

1. $\langle \varphi_X\, t_1, \ldots, \varphi_X\, t_i \rangle \in \mathit{traces}\,[\![\text{AKN1}]\!]$, and

2. the state of the $O_0 \times O_0 \times A$ permissions in the matrix in KN1 after the trace $\langle t_1, \ldots, t_i \rangle$ is equivalent to their state in AKN1 after the trace $\langle \varphi_X\, t_1, \ldots, \varphi_X\, t_i \rangle$.

For the empty trace ($i = 0$), the first statement is true by the healthiness conditions of the CSP traces model. The second statement is true initially because both processes initialise these permissions according to $M_0$.

Now we suppose $i > 0$, and that the induction hypothesis holds for $i - 1$. The transition $t_i$ must have been enable in KN1 because it executed. In particular, this means that all permissions in $t_{i\,grant} \cap (O_0 \times O_0 \times A)$ were on and those in $t_{i\,take} \cap (O_0 \times O_0 \times A)$ were off. Because permissions in $O_0 \times O_0 \times A$ are unaffected by $\varphi_X$, and because these permissions have the same state in AKN1 by the induction hypothesis, the transition will also be enabled in AKN1 after the trace $\langle \varphi_X t_1, \ldots, \varphi_X t_{i-1} \rangle$. We have shown that $\langle \varphi_X t_1, \ldots, \varphi_X t_i \rangle \in \mathit{traces}\, [\![\mathrm{AKN1}]\!]$.

Similarly, we can show that the second statement of the induction hypothesis holds: the function $\varphi_X$ doesn't affect objects in $O_0$, so the effects of $t_{i\,reset}$, $t_{i\,grant}$ and $t_{i\,take}$ on $O_0 \times O_0 \times A$ are identical in the two processes. □ □

**Theorem 10.** *Let* $\langle t_1, \ldots, t_n \rangle \in \mathit{traces}\, [\![\mathrm{AKN1}]\!]$. *Then there exist transitions* $t'_i \in T$ *with* $\varphi_X t'_i = t_i$ *for each* $i = 1, \ldots, n$ *such that*

$$\langle t'_1, \ldots, t'_n \rangle \in \mathit{traces}\, [\![\mathrm{KN1}]\!].$$

*Proof.* Similarly to the previous proof, we use induction on the length of the trace, strengthened with the additional hypothesis that AKN1 and KN1 always have matching states for the permissions $O_0 \times O_0 \times A$ after each pair of respective transitions.

By construction of $T_X$, we know that occurrences of $X$ in $t_i$ actually represent concrete objects in $\overline{O_0}$. Formally, there must be some $t''_i \in T$ such that $\varphi_X t''_i = t_i$.

Furthermore, we can use the data independence condition on $T$ to find another $t'_i \in T$ which replaces any $\overline{O_0}$ objects appearing in $t''_i$ with fresh $\overline{O_0}$ objects not already appearing in the trace $\langle t'_1, \ldots, t'_{i-1} \rangle$. This finite trace only contains finitely many objects, and $\overline{O_0}$ is infinite, so we can always find an appropriate bijection $\varphi : \overline{O_0} \overset{\leftrightarrow}{\rightarrow} \overline{O_0}$ which does this.

We now need to show that $\langle t'_1, \ldots, t'_n \rangle \in \mathit{traces}\, [\![\mathrm{KN1}]\!]$. By construction of $t'_i$, we know that

$$\begin{array}{rrcl} & t_{i\,on} \cap (O_0 \times O_0 \times A) & = & t'_{i\,on} \cap (O_0 \times O_0 \times A) \\ \text{and} & t_{i\,off} \cap (O_0 \times O_0 \times A) & = & t'_{i\,off} \cap (O_0 \times O_0 \times A). \end{array}$$

We also know from the additional induction hypothesis that the state of the permissions $O_0 \times O_0 \times A$ is identical in both AKN1 and KN1 after the traces $\langle t_1, \ldots, t_{i-1} \rangle$ and $\langle t'_1, \ldots, t'_{i-1} \rangle$ respectively. Therefore, as $t_i$ was not blocked by AKN1, we know that $t'_i$ will not be blocked by the $O_0 \times O_0 \times A$ portion of the parallel in KN1. Before we show that $t'_i$ is not blocked by the rest of the matrix, we point out that the above argument can also be applied to *reset*, *grant* and *take* to re-establish the additional induction hypothesis.

The question remains whether the permissions outside of $O_0 \times O_0 \times A$ in KN1 could be in a state to accept $t'_i$. These permissions are of the form $(o, o', a)$ or $(o', o'', a)$ where $o \in O_0$, $o', o'' \in \overline{O_0}$, and $a \in A$. Bare in mind that these permissions were started in an indeterminate state: either on or off. We claim that they are still in an indeterminate state after the trace $\langle t'_1, \ldots, t'_{i-1} \rangle$:

- First let's imagine a permission $(o', o'', a)$, where $o', o'' \in \overline{O_0}$. As both $o'$ and $o''$ have not appeared in the trace before, they could not have appeared in any *on*, *off*, *reset*, *grant* or *take* sets. This means that both branches of $\mathit{PERMISSION}_{ON}(p) \sqcap \mathit{PERMISSION}_{OFF}(p)$ would have been willing to execute any previous transition without change state, keeping the process in the state $\mathit{PERMISSION}_{ON}(p) \sqcap \mathit{PERMISSION}_{OFF}(p)$.

- In fact the same is true of permissions $(o, o', a)$ and $(o', o, a)$, where $o' \in \overline{O_0}$ but $o \in O_0$. As $o'$ is fresh, this permission cannot have appeared in any *on*, *off*, *grant* or *take* sets, and similarly the object $o'$ could not have been in a *reset* set. The object $o$ cannot appear in a *reset* set by our definition of data independence.

We can now resolve this nondeterminism how we like: we resolve it to *ON* for permissions in $t'_{i\,on}$, and *OFF* for permissions in $t'_{i\,off}$. We have now enabled the transition $t'_i$. $\square$ $\square$

These theorems allow us to deduce important relationships between KN1 and AKN1, for example:

**Corollary 11.**

$$\text{KN1} \setminus S \;=_T\; \text{AKN1} \setminus S_X$$

*where* $S$ *and* $S_X$ *are the subsets of* $T$ *and* $T_X$ *respectively that only mention objects in* $O_0$.

**Corollary 12.** *A permission in* $O_0 \times O_0 \times A$ *can be granted by* KN1 *if and only if it can be granted by* AKN1.

**Example 13.** *As noted in Example 5,* KN1 *is an appropriate model to check for the existence of the manager conspiracy scenario in the Employee Information System. The corresponding abstract system* AKN1 *has been coded in FDR, and we can specify that* $S1$ *doesn't obtain the bonus* $B1$ *by checking that*

$$CHAOS_C \sqsubseteq_T \text{AKN1}$$

*where*

$$C = \{t \in T_X \mid \neg\big((S1, B1, \mathsf{HasBonus}) \in t_{grant} \wedge (S2, S2, \mathsf{Exists}) \in t_{on}\big)\}$$

*By Theorems 9 and 10, we know that this refinement will hold if and only if* KN1 *is not able to perform a transition which grants the bonus* $B1$ *to* $S1$ *and requires* $S2$ *to exist (i.e. be involved in the transition.)*

*Checking this in FDR gives the following counterexample:* $\langle t_7(S1), t_3(S2, S1, B1)\rangle$. *This corresponds to the manager* $S1$ *resigning and then being given a bonus by* $S2$. *This is the same as the flaw found in [29].*

*Let's prevent managers from resigning by removing transitions* $t_7(x)$. *FDR now produces a counter-example* $\langle t_5(X, S1), t_3(S2, S1, B1)\rangle$. *The theory tells us that* $X$ *stands for 'some other object in* $\overline{O_0}$*,' so the trace depicts the scenario that a director from* $\overline{O_0}$ *fires* $S1$*; then* $S2$ *can give* $S1$ *the bonus.*

*Removing* $t_5$ *from the system gives us a check that succeeds. Using our theory, this proves the following about the Employee Information System with transitions* $t_5$ *and* $t_7$ *removed: from any initial state (regardless of the number of employees or bonuses), it is not possible for two managers to conspire so that one of them acquires a bonus that neither of them initially held.*

## 6.2 Decidability of KN2 Safety Problems

In this section we show that a finite abstraction for the KN2 problem exists.

Recall that the KN2 system was initialised according to $M_0$ on $O_0 \times O_0 \times A$, and other permissions $p$ would have to be off if $p$ was not a member of a set of safe permissions $SP$.

We now form a finite abstraction of KN2 starting from AKN1. We abstract our safe permissions $SP$ in the same way we abstracted $T$ to $T_X$ for KN1:

$$SP_X = \{\varphi_X p \mid p \in SP\}.$$

We restrict AKN1 so that transitions that require non-safe permissions to be set are disallowed:

$$\mathrm{AKN2}' = \mathrm{AKN1} \underset{\{t \in T_X \mid t_{on} \not\subseteq SP_X\}}{\|} STOP.$$

In order that the previous step remains sound during the computation, we throw an error if our $SP$ invariant is broken. An *error* event can be communicated if non-safe permissions are ever granted:

$$\mathrm{AKN2} = \mathrm{AKN2}'[{}^{error}/_{t} \mid t \in T_X, t_{grant} \not\subseteq SP_X].$$

We say that AKN2 is *error-free* if it is never able to communicate the *error* event.

**Theorem 14.** *If* AKN2 *is not error-free, then there exists a behaviour of* KN2 *where a non-SP permission is granted.*

*Proof.* Take a shortest trace $\langle t_1, \ldots, t_n, error\rangle$ of AKN2. From the definition of AKN2 we know that

1. Because of the *error* renaming, and the fact that this is a shortest trace with *error* occurring, there is a trace $\langle t_1, \ldots, t_{n+1}\rangle \in traces\,[\![AKN1]\!]$ with only $SP_X$ permissions in $t_{i\,grant}$ for $i = 1, \ldots, n$ and some non-$SP_X$ permission in $t_{n+1\,grant}$.
2. Because of the parallel composition with $STOP$, only $SP_X$ permissions appear in $t_{i\,on}$ for $i = 1, \ldots, n+1$.

From Theorem 10 we know that KN1 has a trace $\langle t'_1, \ldots, t'_{n+1}\rangle$ with $\varphi_X t'_i = t_i$ for each $i = 1, \ldots, n$. We now deduce from the two facts above:

1. There are only $SP$ permissions in $t'_{i\,grant}$ for $i = 1, \ldots, n$ and there are some non-$SP$ permissions in $t'_{n+1\,grant}$.
2. Only $SP$ permissions appear in $t'_{i\,on}$ for $i = 1, \ldots, n+1$.

It can now be seen that is also a trace of KN2 because the non-deterministic choices in KN1 can be resolved to $PERMISSION_{OFF}$ for all non-$SP$ permissions: the trace doesn't change the state of these processes (until after the final transition) and in this state the processes will not block any transition in the trace. $\square$ $\square$

**Theorem 15.** *If* AKN2 *is error-free, then we can reproduce Theorems 9 and 10 for* KN2 *and* AKN2*. That is:*

$$\langle t_1, \ldots, t_n \rangle \in \mathit{traces}\,[\![\mathrm{AKN1}]\!]$$

*if and only if*

$$\begin{array}{l} \textit{there exists } t'_1, \ldots, t'_n \in T \textit{ with } \varphi_X\, t'_i = t_i \textit{ for each } i = 1, \ldots, n \\ \textit{such that } \langle t'_1, \ldots, t'_n \rangle \in \mathit{traces}\,[\![\mathrm{KN1}]\!]. \end{array}$$

*Proof.* We can reuse the previous proof (omitting the *error* event at the end of the trace) to show that $\langle t_1, \ldots, t_n \rangle \in \mathit{traces}\,[\![\mathrm{AKN2}]\!]$ implies $\langle t'_1, \ldots, t'_n \rangle \in \mathit{traces}\,[\![\mathrm{KN2}]\!]$.

We now need to show that $\langle t_1, \ldots, t_n \rangle \in \mathit{traces}\,[\![\mathrm{KN2}]\!]$ implies $\langle \varphi_X\, t_1, \ldots, \varphi_X\, t_n \rangle \in \mathit{traces}\,[\![\mathrm{AKN2}]\!]$.

From Theorem 8 we know that $\langle t_1, \ldots, t_n \rangle \in \mathit{traces}\,[\![\mathrm{KN1}]\!]$, and Theorem 9 gives us $\langle \varphi_X\, t_1, \ldots, \varphi_X\, t_n \rangle \in \mathit{traces}\,[\![\mathrm{AKN1}]\!]$.

To see that this is also a trace of AKN2, we just need to observe that:

1. Each $(\varphi_X\, t_i)_{grant}$ contains only $SP_X$ permissions. For if this were not true, it would mean there is a behaviour of KN2 where a non-$SP$ permissions is granted. This would contradict with our assumption that AKN2 is error-free via Theorem 14.

2. Each $(\varphi_X\, t_i)_{on}$ contains only $SP_X$ permissions. Otherwise, there would be some $t_{i\,on}$ containing a non-$SP$ permission: this can't be the case because KN2 starts that permission in the off-state, and it remains in the off-state because KN2 is error-free as above.

The above two points show that the trace $\langle \varphi_X\, t_1, \ldots, \varphi_X\, t_n \rangle$ of AKN1 is unaltered by the parallel and renaming operators in the definition of AKN2. □ □

**Example 16.** *Let's return to the EIS example. In Example 13 we saw that the manager conspiracy scenario can be avoided by removing a couple of rules from the system. We now look at another approach to achieve this aim. We still dispense with rule $t_7$ that allows managers to resign, but we will keep $t_5$ which allows managers to be demoted by a director. Instead we insist that there are no directors in the initial system. We can do this using the* KN2 *model as described in Example 6.*

*We have coded up the finite abstraction* AKN2 *in FDR and the following check succeeded:*

$$\mathit{CHAOS}_C \sqsubseteq_T \mathrm{AKN2}$$

*(where $C$ is defined in Example 13.) This shows that* AKN2 *is error-free: it is therefore not possible for a director to be created in* KN2*. We can now use Theorem 15 to deduce from this refinement that the manager conspiracy scenario cannot occur in* KN2*. In words, the scenario does not occur whenever the system (without $t_7$) is started in any initial state satisfying the following two constraints: (1) the objects $S1, S2$, and $B1$ satisfy the matrix $M_0$, and (2) all other objects $o$ do not have $(o, o, \mathsf{Director})$ set.*

### 6.3 Undecidability of KN$\perp$ Safety Problems

Here we show that the safety problem for KN$\perp$ systems is undecidable. This is done using a reduction to the HRU safety problem shown to be undecidable in [17].

The safety problem for HRU is defined as follows: 'from a specific initial state, is it possible for a particular access right $a \in A$ to be granted anywhere in the system?' As described in Sect. 4, any HRU model can be simulated in our KN model. The KN$\perp$ process is able to express a specific initial state, and one object in $O_0$ could be used to model an arbitrary object. Therefore this safety problems for KN$\perp$ is undecidable.

However, we can use Theorem 8 combined with the decidability results about KN1 and KN2 presented in this paper to get an approximate decision procedure for KN$\perp$ safety problems.

**Example 17.** *Theorem 8 tells us that the positive results about the Employee Information System proved in Examples 13 and 16 also hold for* KN$\perp$. *This is despite the fact that arbitrarily many new objects can be created by transitions* $t_8(x, n)$.

## 7 Conclusions

We have introduced the novel satefy properties for protection systems and showed that they are decidable under KN and thus under HRU. We argue that in some circumstances, even if *hru safety* can be proven, these safety properties give better assurance of the security of an access control system. For example, proving that a policy is safe with KN2 guarantees correctness regardless of any change in the initial state, so long as the change is consistent with the initial conditions and invariants specified by the KN2 property.

It was shown that the analysis of *safety* properties under the RW model for an unbounded number of subjects and objects is decidable as long as specifications only use the existential quantifier[3]. The fact that we could show that safety analysis under an existing model is decidable, gives hope that the algorithms presented in the paper will be adequate for analysing practical systems.

We have also offered a sound but incomplete algorithm for deciding hru safety under HRU. It might be the case that the number of the "false" leakages can be reduced and this remains a topic for further investigation. However, by the undecidability result, it is clear that the "false" leakages potential cannot be eliminated. Although this algorithm is incomplete, in reality it might be valuable. As an example we refer to the safety problem of cryptographic protocols that has been extensively researched in the last decade. Although many of the proposed tools and algorithms in this realm are fail-safe, it turned out that in practice it was rare to find correct protocols these tools could not prove.

## 8 Related Work

Despite previous lack of an automatic algorithm for analysing the safety of access control policies, a few methods for evaluating such system have been developed. The most closely

[3]The hru safety problem can be expressed using the existential quantifier.

related work is [3] in which it is shown how access control mechanisms with a bounded number of subjects and resources can be expressed using CSP. The author demonstrates how concepts from XACML and such as separation of duty can be modelled in CSP in addition to several *safety* properties which can be verified using FDR. The CSP models presented in our paper are to some extent similar but allows us to reason about *safety* properties of unbounded systems.

Fisler *et al.*[7] developed a tool called *Margrave* which takes role-base access control policies specified in XACML and represent them as multi-terminal decision diagrams (MTBDDs). By exploring the MTBDDs, Margrave can check whether a policy preserves a given property and generate a counter-example if it does not. Margrave also supports a change-impact analysis in which it consumes two policies that span a set of changes and summarizes the differences. Margrave does not consider dynamic changes of the policy (administrative model) and hence, in terms of safety, its analysis is limited. The models presented in our paper can be used for modelling RBAC polices with administrative models that bound the number of roles. Analysing more general RBAC policies remains a subject of future work.

Guelev, Ryan and Schobbens [8] presented the RW formalism, base 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 coalition of a fix number of agents to achieve a certain goal in the presence of a fixed number of resources. It is therefore usable only for finding flaws and cannot provide general proofs of correctness of a safety of a policy. In addition, a tool was provided which takes an RW script as input and converts the policy description into XACML [28].

Ahmed and Tripathi [1] demonstrated how static verification can be employed to show that security constraints do not violate any desired security property.

Schaad and Moffett [26] showed how access control policies complying to RBAC96 and ARBAC97 together with a set of separation of duty properties can be translated into Alloy. They demonstrated how Alloy can then be used to check whether these policies do not breach the SoD constraints.

## 9 Future Work

The results presented in this paper are the foundation of the work we are seeking to carry out in the future.

KN lacks some features which are common in practical systems such as Groups and the Unique-permission constraint (e.g. in Unix only one subject can own a resource at a specific point in time.) For clarity we omitted these in this paper. In the future we intended to offer a complete model.

It was proven that safety analysis under KN1 and KN2 is decidable. We would be interested in expanding these models' expressiveness without losing this property. For example, KN2 allows, under certain conditions, some permissions outside $(O_0, M_0)$ to be set off. We would be interested in relaxing these conditions and considering cases in which permissions outside $(O_0, M_0)$ are set on.

Similarly to research conducted in this area, we believe we might be able to propose restricted versions of HRU and prove that the hru safety problem regains decidability by reduction to KN2. Another avenue for finding decidable subsets of HRU is the work by Lazić and Roscoe [21] who investigated the decidability of reachability specifications in finite control programs which are data-independent in type $X$, and which contain resettable arrays indexed by type $X$ and storing values from a type $Y$. Using these result we can generalise [14]'s result to KN. We are interested in simulating such system in CSP and using this technique for identifying even more expressive subsets of HRU under which the hru safety problem is decidable.

Though performance was out of the scope of this paper, it seems that KN2 can be explored efficiently especially when considering *monotonic* protection systems. However, we realise that our CSP model can be optimized by utilising symmetry and *chasing* [20]. After tackling these, we are interested in analysing the models' complexity and evaluating real systems.

We acknowledge that 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 the CSP scripts from a more abstract description. At the same time, a simple language for expressing policy specifications needs to be designed and supported by the proposed compiler.

There have been a few attempts to analyse Trust Management systems [2]. The models propose in this paper together with the knowledge gained in the security protocols field [23] can be combined in order to reason about such system. The only decidability result we are aware of in this realm is due to Li *et al.* [13] who considered safety analysis in the policy language RT[$\twoheadleftarrow, \cap$]. It can be shown that RT[$\twoheadleftarrow, \cap$] is subsumed by HRU and by KN2. Thus, the results presented in our paper can be used to shad more light on the decidability of TM systems.

It was shown in [5, 6] how a mapping function ($\phi$) can be used to generate automatically models of XML based security protocols specified by SOAP and WS-Security. We would like to extend $\phi$ to automatically produce CSP models of polices given by XACML. Once a TM model is completed, we would be interested in extending $\phi$ for supporting WS-Security protocols with incorporated SAML assertions [4].

Lastly, we are interested in extending our result to RBAC (role based access control) policies. In [18] Munawer and Sandhu have suggested that the HRU model can be simulated using RABC96 [25]. It follows that the safety problem under RBAC is undecidable as well. As mentioned above, the models presented in this paper can be extended for modelling RBAC polices in which the system is limited to a fixed number of roles but unrestricted otherwise. Recent results [12] reveal the requirements under which a system with an array with a linear order operator is decidable. We hope that these can be used for uncovering less strict requirements to achieve decidability of the safety problem of RBAC systems.

## References

[1] Tanvir Ahmed and Anand R. Tripathi. Static verification of security requirements in role based cscw systems. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 196–203, New York, NY, USA, 2003. ACM Press.

[2] M. Blaze, J. Feigenbaum, and J Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, number 96-17, 1996.

[3] Jeremy Bryans. Reasoning about XACML policies using CSP. Technical Report CS-TR-924, University of Newcastle, July 2005.

[4] S. Cantor, J. Kemp, R. Philpott, and E. Maler. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Committee Specification sstc-core, March 2005, 2005.

[5] E. Kleiner and A.W. Roscoe. Web Services Security: a preliminary study using Casper and FDR. In *Automated Reasoning for Security Protocol Analysis (ARSPA 04)*, Cork, Ireland, 2004.

[6] E. Kleiner and A.W. Roscoe. On the relationship of traditional and Web Services Security protocols. In *Mathematical Foundations of Programming Semantics (MFPS 05)*, Birmingham, UK, 2005.

[7] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 196–205, New York, NY, USA, 2005. ACM Press.

[8] Dimitar P. Guelev, Mark Ryan, and Pierre-Yves Schobbens. Model-checking access control policies. In *ISC*, pages 219–230, 2004.

[9] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[10] Manuel Koch, Luigi V. Mancini, and Francesco 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*, pages 229–243, London, UK, 2002. Springer-Verlag.

[11] R.S. Lazić, T.C. Newcomb, and A.W. Roscoe. On model checking data-independent systems with arrays without reset. *Theory Pract. Log. Program.*, 4(5-6):659–693, 2004.

[12] R.S. Lazić, T.C. Newcomb, and A.W. Roscoe. Polymorphic systems with arrays, 2-counter machines and multiset rewriting. In *Proceedings of INFINITY*, 2004.

[13] Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *J. ACM*, 52(3):474–514, 2005.

[14] R. Lipton and L. Snyder. On synchronization and security. In A. Jones R. DeMillo, D. Dobkin and R. Lipton, editors, *Foundations of Secure Computation*, pages 367–385. Academic Press, 1978.

[15] Formal Systems (Europe) LTD. *Failure-Divergences Refinement FDR2 Manual*, 1997.

[16] Harrison M. and W. Ruzzo. Monotonic protection systems. In A. Jones R. DeMillo, D. Dobkin and R. Lipton, editors, *Foundations of Secure Computation*, pages 337–363. Academic Press, 1978.

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

[18] Q. Munawer and R. Sandhu. Simulation of the augmented typed access matrix model (atam) using roles. In *INFOSECU99 International Conference on Information Security*, 1999.

[19] A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop, 9–11 June, 1998*, pages 84–95. IEEE, 1998.

[20] A. W. Roscoe and M. H. Goldsmith. The perfect "spy" for model-checking cryptoprotocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers University, September 1997.

[21] A.W. Roscoe and R.S. Lazić. What can you decide about resetable arrays? In *Proceedings of the 2nd International Workshop on Verification and Computational Logic (VCL 2001)*, September 2001.

[22] R.S. Lazić. *A semantics study of data-independence with applications to the mechanical verification of concurrent systems.* PhD thesis, University of Oxford, Oxford, UK, 1999.

[23] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A.W. Roscoe. Modelling and analysis of security protocols, 2001.

[24] R. S. Sandhu and G. S. Suri. Non-monotonic transformation of access rights. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 148–163, 1992.

[25] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[26] Andreas Schaad and Jonathan D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *SACMAT '02: Proceedings*

*of the seventh ACM symposium on Access control models and technologies*, pages 13–22, New York, NY, USA, 2002. ACM Press.

[27] Pierre Wolper. Expressing interesting properties of programs in propositional temporal logic. In *POPL '86: Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 184–193, New York, NY, USA, 1986. ACM Press.

[28] Nan Zhang, Mark Ryan, and Dimitar P. Guelev. Synthesising verified access control systems in xacml. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 56–65, New York, NY, USA, 2004. ACM Press.

[29] Nan Zhang, Mark Ryan, and Dimitar P. Guelev. Evaluating access control policies through model checking. In *ISC*, pages 446–460, 2005.

## APPENDIX A: Quick Guide to CSP

- $STOP$ - does nothing.
- $a \rightarrow P$ - initially willing to communicate $a$; if it performs $a$, afterwards it behaves like $P$.
- $P \lhd b \rhd Q$ - behaves like $P$ if the boolean condition $b$ is true, otherwise behaves like $Q$.
- $b \;\&\; P$ - guard $P$ with the boolean condition $b$; equivalent to $P \lhd b \rhd STOP$.
- $P \sqcap Q$ - nondeterministically behaves like either $P$ or $Q$.
- $\underset{X}{\|}\,_{i \in A} P(i)$ - put the processes $P(i)$, for each $i \in A$, together synchronising on the events in $X$.
- $P \setminus X$ - behaves like $P$ except the events $X$ have been hidden and are therefore uncontrollable by the environment of $P$.
- $P[^{e'}/_{e}]$ - Behaves like $P$ except events $e$ are renamed to $e'$.
- *traces* $[\![P]\!]$ is the set of all finite sequences of events that are prefixes of behaviours of $P$.
- $P \sqsubseteq_T Q$ means that $P$ is refined by $Q$: the traces of $P$ include the traces of $Q$.
- $Chaos[C]$ is a process whose traces are exactly the finite sequences over the set of events $C$.