The Knowledge Base Paradigm Applied to Delegation Revocation

Marcos Cramer
TU Dresden

Zohreh Baniasadi
University of Luxembourg

Pieter Van Hertum
KU Leuven

Abstract

In ownership-based access control frameworks with the possibility of delegating permissions and administrative rights, delegation chains will form. There are different ways to treat delegation chains when revoking rights, which give rise to different revocation schemes. In this paper, we investigate the problem of delegation revocation from the perspective of the *knowledge base paradigm*. A *knowledge base* is a formal specification of domain knowledge in a rich formal language. Multiple forms of inference can be applied to this formal specification in order to solve various problems and tasks that arise in the domain. In other words, the paradigm proposes a strict separation of concerns between information and problem solving. The knowledge base that we use in this paper specifies the effects of the various revocation schemes. By applying different inferences to this knowledge base, we can solve the following tasks: to determine the state of the system after a certain delegation or revocation; to interactively simulate the progression of the system state through time; to determine whether a user has a certain permission or administrative right given a certain state of the system; to verify invariants of the system; and to determine which revocation schemes give rise to a certain specified set of desired outcomes.

1 Introduction

In ownership-based frameworks for access control, it is common to allow principals (users or processes) to grant both permissions and administrative rights to other principals in the system. Often it is desirable to grant a principal the right to further grant permissions and administrative rights to other principals. This may lead to delegation chains starting at a *source of authority* (the owner of a resource) and passing on certain permissions to other principals in the chain [19, 22, 7, 25].

Furthermore, such frameworks commonly allow a principal to revoke a permission that she granted to another principal [16, 26, 7, 3]. Depending on the reasons for the revocation, different ways to treat the chain of principals whose permissions depended on the second principal's delegation rights can be desirable [16, 8]. For example, if one is revoking a permission given to an employee because he is moving to another position in the company, it makes sense to keep the permissions of principals who received them from this employee; but if one is revoking a permission from a user who has abused his rights and is hence distrusted by the user who granted the permission, it makes sense to delete the permissions of principals who received them from this user. Any algorithm that determines which permissions to keep intact and which permissions to delete when revoking a permission is called a *revocation scheme*.

Hagström et al. [16] have presented a framework for classifying possible revocation schemes along three dimensions: the extent of the revocation to other grantees (propagation), the effect on other grants to the same grantee

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

(dominance), and the permanence of the negation of rights (resilience). Since there are two options along each dimension, there are in total eight different revocation schemes in Hagström et al.'s framework. This classification was based on revocation schemes that had been implemented in database management systems [15, 14, 5, 4].

In this paper, we investigate the problem of delegation revocation from the perspective of the *knowledge base paradigm*, a declarative programming paradigm based on the idea of strictly separating information and problem solving. A *knowledge base* is a formal specification of domain knowledge in a rich formal language. Multiple forms of inference can be applied to this formal specification in order to solve various problems and tasks that arise in the domain. The IDP system is an implementation of the knowledge base paradigm with associated formal language FO($\cdot$), an extension of first-order logic [11].

In this paper we present an application of the knowledge base paradigm to delegation revocation, realized in the IDP system. We have written a formal specification of the eight revocation schemes in Hagström et al.'s framework, which formally described their effects. By applying different inferences to this specification, we can solve various tasks that can be useful both for implementing a system which allows for these revocation schemes and for supporting a user of such a system: Given a certain state of the system and a certain action (delegation or revocation), one inference determines the state of the system after this action. Another inference can interactively simulate the progression of the system state through time. A third determines whether a user has a certain permission or administrative right given a certain state of the system. A fourth inference allows to verify that the system satisfies certain invariants. Finally, there is an inference that allows a user to specify a set of desired outcomes (e.g. that a certain user should no longer have a certain right while another user is unaffected) and determine which actions give rise to this outcome.

The paper is structured as follows: We introduce Hagström et al.'s delegation revocation framework in Section 2. In Section 3 we motivate and describe the knowledge base paradigm, the IDP system and its specification language FO($\cdot$). Section 4 presents the application of the knowledge base paradigm to Hagström et al.'s framework. Section 5 discusses related work and section 6 concludes.

2 The delegation revocation framework

In this section we present Hagström et al.'s [16] delegation revocation framework in a slightly simplified way.

Let P be the set of principals (users or processes) in the system, let O be the set of objects for which authorizations can be stated, and let A be the set of access types, i.e. of actions that principals may perform on objects. For every object $o \in O$, there is a *source of authority* (*SOA*), for example the owner of file o, which is a principal that has full power over object o and is the ultimate authority with respect to accesses to object o. For any $a \in A$ and $o \in O$, the SOA of o can grant the right to access a on object o to other principals in the system, and can also delegate the right to grant access and to grant this delegation right. Therefore two kinds of *permissions* are distinguished: Permission α means that only access right is granted, while permission δ means that both access and delegation rights are granted. Additionally, Hagström et al.'s framework allows for negative authorizations, which can be used to block a principal's access or delegation rights without deleting any authorization. Due to space limitations, we restrict ourselves to the fragment of Hagström et al.'s framework consisting of positive authorizations only. In a technical report to this paper, negative authorizations are taken into account [**?**].

We assume that all authorizations in the system are stored in an authorization specification, and that every authorization is of the form (i, j, a, o, π), where $i, j \in P$, $a \in A$, $o \in O$, and $\pi \in \{\alpha, \delta\}$ is a permission. The meaning of this authorization is that principal i is granting some permission concerning access type a on object o to principal j. If b is T, the permission contains the right to delegate the permission further.

Note that there is no interaction between the rights of principals concerning different access-object pairs (a, o), so we can consider a and o to be fixed for the rest of the paper. Therefore we simplify an authorization (i, j, a, o, π) to (i, j, π).

2.1 Delegation chains and connectivity property

Hagström et al. use the notion of a principal p *having a permission* π without formally defining it. The intended meaning of this can be formalized using the notion of a *rooted delegation chain*:[1]

Definition 1 *A* rooted delegation chain *for principal i with respect to permssion π is a sequence $p_1, \ldots, p_n$ of principals satisfying the following properties:*

[1]The idea to formalize Hagström et al.'s notion of *having a permission* in this way was first proposed by Aucher et al. [1].

1. p_1 *is the source of authority.*
2. p_n *is* i*.*
3. *For every integer* k *with* $1 \leq k < n$*, an authorization* (p_k, p_{k+1}, π_k) *is in place.*
4. *For every integer* k *with* $1 \leq k < n-1$*,* $\pi_k = \delta$*.*
5. $\pi_n = \pi$*.*

Definition 2 *We say that a principal* p has permission π *iff there is a rooted delegation chain for principal* p *with respect to permission* π*.*

Hagström et al.'s framework allows an authorization (i, j, π) to be in the authorization specification only if i has delegation right. This is called the *connectivity property*, and it can be viewed as an invariant that any system based on Hagström et al.'s framework needs to satisfy:

Connectivity property: *For every authorization* (i, j, π) *in the authorization specification,* i *has permission* δ*.*

2.2 The three dimensions

Hagström et al. [16] have introduced three dimensions according to which revocation schemes can be classified. These are called *propagation*, *dominance* and *resilience*:

Propagation. The decision of a principal i to revoke an authorization previously granted to a principal j may either be intended to affect only the direct recipient j or to affect all the other users in turn authorized by j. In the first case, we say that the revocation is *local*, in the second case that it is *global*.

Dominance. This dimension deals with the case when a principal losing a permission in a revocation still has permissions from other grantors. If these other grantors' are *dependent* on the revoker, she can dominate these grantors and revoke the permissions from them. This is called *strong* revocation. The revoker can also perform a *weak* revocation, where permissions from other grantors to a principal losing a permission are kept.

In order to formalize this dimension, we need to define what we mean by a principal's delegation rights to be *independent* of another principal:

Definition 3 *A principal* j has delegation rights independent of *a principal* i *with respect to permission* π *iff there is an rooted delegation chain for* j *with respect to* π *that does not contain the principal* i*.*

Resilience. This dimension distinguishes revocation by removal of positive authorizations from revocation by negative authorizations which just inactivate positive authorizations. Given that we concentrate on the fragment of Hagström et al.'s framework without negative revocations, we will not explain this dimension in detail.

Since there are two options along each of the three dimensions, Hagström et al. defined eight different revocation schemes. The technical report of this paper contains an illustration of these revocation schemes [**?**].

3 The knowledge base paradigm and the FO(·) KB project

Declarative systems have proven their merit in many application domains: From planning to scheduling to security contexts, many challenges have been tackled using declarative approach. For example, Barker et al. [2] used a rule-based approach to determine access rights in an access control system. Advantages of such an approach are readability and maintainability of a specification. However, only one task is supported in a rule based system, or in any other decalarative system. Every system has its own syntactical style, terminology, conceptualization, and designated style of inference (rule based systems do chaining, databases do querying, answer set programming generates answer sets, etc.). Yet, in all of them, propositions need to be expressed. Take, e.g., "each lecture takes place at some time slot". This proposition could be an expression to be deduced from a formal specification if the task was a verification problem, or to be queried in a database, or it could be a constraint for a scheduling problem. It is, in the first place, just a piece of information and we see no reason why depending on the task to be solved, it should be expressed in a different formalism.

The *knowledge base* (*KB*) *paradigm* [13] was proposed as an answer to this. The KB paradigm applies a strict separation of concerns to information and problem solving. A KB system allows information to be stored in a knowledge base, and provides a range of inference methods. With these inference methods various types of problems and tasks can be solved using the same knowledge base. As such the knowledge base is neither a program nor a description of a problem, it cannot be executed or run. It is nothing but information. However,

this information can be used to solve multiple sorts of problems. Many declarative problem solving paradigms are mono-inferential: they are based on one form of inference. In comparison, the KB paradigm is multi-inferential.

The FO(·) KB project is a research project in which an implementation of the KB paradigm is being developed. Its aim is to integrate different useful language constructs and forms of inference from different declarative paradigms in one rich declarative language and a KB system. So far, it has led to the KB language FO(·) [12] and the KB system IDP [11], which are used in this paper.

3.1 The specification language FO(·)

FO(·) refers to a class of extensions of first-order logic (FO). The language of the current version of the IDP system (IDP 3) is FO(T, ID, Agg, arit, PF): FO extended with types, inductive definitions, aggregates, arithmetic and partial functions (see [12, 21]). In this work, we will only work (and as such, introduce) a subset of this language: $FO(T, ID)$: typed first-order logic with inductive definitions. Abusing notation, we will use FO(·) as an abbreviation of this language.

3.2 An FO(·) specification

A specification of domain knowledge in FO(·) can consist of 3 types of building blocks: a vocabulary Σ, a theory T and a (partial) structure $\mathcal{S}$.

The *vocabulary* declares the symbols used in the associated theories and structures. It is a set Σ of type symbols (denoted as Σ_T) and predicate symbols (denoted as Σ_P). Every predicate P of arity n has a fixed type $[\tau_1, \ldots, \tau_n]$, where $\tau_1, \ldots, \tau_n$ are type symbols. Variables, atoms and first-order formulas are defined as usual.

A *theory* is a set of first-order formulas and inductive definitions. An inductive definition Δ in FO(·) is a set of rules δ of the form $P(\bar{t}) \leftarrow \varphi$, with φ a first-order formula. We call $P(\bar{t})$ the head ($head(\delta)$) and φ the body ($body(\delta)$) of the rule. The symbols that occur in the head of a rule δ in Δ are called the defined symbols in Δ. All other symbols that occur in Δ are called the parameters of Δ. The semantics used for inductive definitions is the well-founded semantics; as argued in [12], this captures the intended meaning of all forms of inductive definitions commonly used in mathematics and computer science. Informally a structure $\mathcal{S}$ satisfies Δ if the interpretation of a defined predicate P in the well-founded model of $\mathcal{S}$, constructed relative to the restriction of $\mathcal{S}$ to the parameters of Δ, is exactly the relation $P^{\mathcal{S}}$.

The following example illustrates the use of an inductive definition in a theory by presenting the defintion of "reachable" in FO(·).

Example 1 *Assume a vocabulary containing a type $Node$, and 2 predicates: $Edge(Node, Node)$ and $Reachable(Node, Node)$. Informally, Edge states that there is an edge between two nodes, while Reachable states that there is a path of edges between 2 notes. We define what reachability means in terms of the edges, using an inductive definition Δ in* FO(·)*:*

$$\left\{ \begin{array}{l} \forall x : Reachable(x, x). \\ \forall x\ y : Reachable(x, y) \leftarrow \exists z : Reachable(x, z) \wedge Edge(z, y). \end{array} \right\}$$

Given a vocabulary Σ, a *partial structure* gives an interpretation to (a subset of) the elements of Σ. Before we define formally what an interpretation is, we define the concept of a *partial set*, which is a generalisation of a set in a 3-valued context: A *partial set* on domain D is a function from D to $\{\mathbf{t}, \mathbf{u}, \mathbf{f}\}$. A partial set is two-valued (or total) if $\mathbf{u}$ does not belong to its range. A *(partial) structure* $\mathcal{S}$ consists of a domain D_τ for all types τ in Σ_T and an assignment of a partial set $P^{\mathcal{S}}$ to each predicate or function symbol $P \in \Sigma_P$, called the *interpretation* of P in $\mathcal{S}$. The interpretation $P^{\mathcal{S}}$ of a predicate symbol P with type $[\tau_1, \ldots, \tau_n]$ in $\mathcal{S}$ is a partial set on domain $D_{\tau_1} \times \ldots \times D_{\tau_n}$. In case the interpretation of a predicate P in $\mathcal{S}$ is a two-valued set, we abuse notation and use $P^{\mathcal{S}}$ as shorthand for $\{\bar{d} | P^{\mathcal{S}}(\bar{d}) = \mathbf{t}\}$.

We call a partial structure *total* if the interpretation $P^{\mathcal{S}}$ of every predicate symbol $P \in \Sigma_P$ is a total set. Note that with the abuse of notation just explained, a total structure as we have defined it can be identified with a first-order structure as it is usually defined.

Given two partial structures $\mathcal{S} = (D, \mathcal{I})$ and $\mathcal{S}' = (D, \mathcal{I}')$, we write $\mathcal{S} \leq_p \mathcal{S}'$ (and say S *is more precise than* S', or S' *expands* S) iff for every predicate symbol $P \in \Sigma_P$ with type $[\tau_1, \ldots, \tau_n]$ and every tuple $\bar{d} \in D_{\tau_1} \times \ldots \times D_{\tau_n}$ such that $P^{\mathcal{I}}(\bar{d}) \neq \mathbf{u}$, we have $P^{\mathcal{I}'}(\bar{d}) = P^{\mathcal{I}}(\bar{d})$.

3.3 The reasoning engine

In the FO($\cdot$) KB project, a implementation of a KB System was developed: the IDP system [11]. IDP takes an FO($\cdot$) specification (that is, a combination of vocabularies, theories and/or structures) and can do a number of reasoning tasks, by applying a suitable form of inference on this specification. Below, we present the inferences that we need in this paper:

Modelexpand($T, \mathcal{S}$): Input: theory T and partial structure $\mathcal{S}$. Output: either a total structure I such that I is a model of T and $\mathcal{S} \leq_p I$, or *UNSAT* if there is no such I. Modelexpand [24] is a generalization for FO($\cdot$) theories of the modelexpansion task as defined in Mitchell et al. [20].

Allmodels($T, \mathcal{S}$): Input: theory T and partial structure $\mathcal{S}$. Output: the set of all total structures I such that I is a model of T and $\mathcal{S} \leq_p I$.

Query($\mathcal{S}, E$): Input: a (partial) structure $\mathcal{S}$ and a set expression $E = \{\overline{x} \mid \varphi(\overline{x})\}$. Output: the set $A_Q = \{\overline{x} \mid \varphi(\overline{x})^{\mathcal{S}} = \mathbf{t}\}$.

Progression($T, \mathcal{S}_i$): In [6], LTC theories (Linear Time Calculus) are proposed, a syntactic subclass of FO($\cdot$) theories that allow to naturally model dynamic systems. An LTC theory consists of three types of constraints: constraints about the initial situation, invariants, and "bistate" formulas that relate the state on the current point in time with that of the next. Note that the specification presenten in Subsection 4.1 below is an LTC theory.

The *Progression inference*: Input: an LTC theory T and a structure $\mathcal{S}_i$ that provides information about the state of the system on a time point t. Output: a structure $\mathcal{S}_{t+1}$ that represents the next state (or a next possible state) at time point $t+1$. Repeating this process, we can compute all subsequent states, effectively simulating the dynamic system defined by T.

4 Delegation revocation in the KB paradigm

In this section, we explain how the KB paradigm can be applied to delegation revocation. For this purpose, we show how the delegation revocation framework defined in Section 2 can be specified in FO($\cdot$), and how inferences on this specification can solve various tasks that arise in the domain. Some of these tasks are tasks that any system implementing the delegation revocation framework needs to solve, while others are tasks that support a user of such a system.

We have built a prototype in IDP in which this application of the KB paradigm is realized. This prototype also covers the negative authorizations and negative revocation schemes that this paper does not explain due to space restrictions. This prototype can be downloaded at `http://icr.uni.lu/mcramer/downloads/hagstrom-R&DS.zip` and run in IDP 3.

4.1 The FO($\cdot$) specification of the delegation revocation framework

In this subsection, we describe how Hagström et al.'s delegation revocation framework, which we defined semi-formally in Section 2, can be formally specified in FO($\cdot$).[2] In this subsection, we use IDP syntax for FO($\cdot$): The symbols `&`, `|`, `~`, `!` and `?` mean $\wedge$, $\vee$, $\neg$, $\forall$ and $\exists$ respectively, and `<-` means $\leftarrow$ (in inductive definitions).

The FO($\cdot$) specification models the change of the authorization specification over time. We allow for four types of objects: `Time`, `principal`, `scheme` and `permission`. Time points are integers. There is a constant `SOA` of type `principal` that denotes the source of authority. The type `scheme` consists of the four delete revocation schemes (`WLD`, `WGD`, `SLD` and `SGD`) and two schemes for granting the two different kinds of permissions (`grant_access` and `grant_deleg`). The permissions are `access` and `deleg`. Positive authorizations are modelled by the predicate `pos_auth`. The authorizations cannot be modelled as objects, because they change over time, while FO($\cdot$) assumes a constant domain of objects.

As IDP only works with finite domains, the type `Time` actually just consists of a finite set of consecutive integers. There is a constant `Start` for the first time point. The unary partial function `Next` maps a time point `t` to the next time point `t+1`, as long as `t` is not the last time point included in the domain.

[2] For a version of the specification that includes the formal specification of the vocabulary, that also covers negative authorizations, and that contains comments that clarify various details, see the file `MainTheory.idp` of the prototype that can be downloaded at `http://icr.uni.lu/mcramer/downloads/hagstrom-R&DS.zip`.

The predicate `pos_auth` for positive authorizations takes four arguments: `pos_auth(t,i,j,a)` means that at time `t`, a positive authorization from principal `i` to principal `j` for permission `a` is in place. There is a tertiary predicate `pos_auth_start` for specifying the positive authorizations that are in place at the first time point.

Changes in the authorization specification are always triggered by some action by a principal: `action(t,s,i,j)` means that at time `t`, principal `i` performs an action of the (revocation or grant) scheme `s` affecting principal `j`. These actions can lead to authorizations being deleted and/or new authorizations being included in the authorization specification. `delete(t,i,j,a)` means that between time points `t-1` and `t`, the positive authorization from `i` to `j` for permission `a` gets deleted. `new(t,i,j,a)` means that between time points `t-1` and `t`, a new positive authorization from `i` to `j` for permission `a` gets added to the authorization specification.

`pos_auth` is defined inductively by setting its values at the first time point `Start` to the start configuration specified by `pos_auth_start`, and by modifying its values between time `t` and `t+1` according to the changes specified by `delete` and `new`:

```
{       pos_auth(Start,p1,p2,a)<-pos_auth_start(p1,p2,a).
        pos_auth(Next(t),p1,p2,a)<-pos_auth(t,p1,p2,a) & ~delete(Next(t),p1,p2,a).
        pos_auth(Next(t),p1,p2,a)<-new(Next(t),p1,p2,a). }
```

The predicate `chain(t,i,a)` expresses that at time `t`, there exists a rooted delegation chain for principal `i` with respect to permission `a`. In Section 2, rooted delegation chains are defined by quantifying over sequences of principals. This is in effect a second-order quantification, which is not possible in the first-order language FO(·). However, `chain(t,i,a)` can be equivalently defined through an inductive definition as follows:[3]

```
{       chain(t,SOA,deleg).
        chain(t,p1,a) <- ?p2: chain(t,p2,deleg) & pos_auth(t,p2,p1,a). }
```

The predicate `can_delegate(t,i)` expresses that principal `i` has permission `del`. The predicate `ind(t,i,j,a)` models the independence of principal `i` from principal `j` with respect to a permission `a`, and the access right of principal `i`. These two predicates are defined as follows:

```
{       can_delegate(t,i) <- ?a1: chain(t,i,deleg). }
{       ind(t,SOA,p,a) <- ~ SOA = p.
        ind(t,p1,p2,a) <- ~ p1 = p2 & ?p: ind(t,p,p2,a) & pos_auth(t,p,p1,a).}
```

The different effects of the different deletion revocation schemes are captured by the definitions of the predicates `delete` and `new`. `delete` is defined via an inductive definition with four clauses:

```
{       delete(Next(t),i,j,a)<- pos_auth(t,i,j,a) & action(t,s,i,j) &
                (s=WLD | s=SLD | s=WGD | s=SGD).
        delete(Next(t),j,k,a)<-pos_auth(t,j,k,a) & ~ can_delegate(Next(t),j).
        delete(Next(t),k,j,a)<-pos_auth(t,k,j,a) & action(t,SLD,i,j) & ind(t,k,i,a1).
        delete(Next(t),z,w,a)<-pos_auth(t,z,w,a) & action(t,SGD,i,j)
                & delete(Next(t),p,w,a1) & ind(t,z,i,a2). }
```

The first clause just states that in any deletion revocation scheme from i to j, the positive authorization from i to j is deleted. The second clause defines the propagation of deletion by specifying that any positive authorization from i to j gets deleted if i is losing its delegation right. The last two clauses capture the meaning of *strong* vs. *weak* dominance by specifying the additional deletions that are needed in strong revocation schemes.

The predicate `new` takes care that new authorizations are added either when an action to grant a new permission takes place (first clause of the definition of `new`) or when a local revocation requires the addition of a new authorization (second clause):

```
{       new(Next(t),i,j,a) <-
                ? ds:( (ds=grant\_{}access & a = access) | (ds=grant\_{}deleg & a = deleg
                action(t,ds,i,j) & can_delegate(t,i).
        new(Next(t),i,k,a) <- ?j s:(s=WLD | s=SLD) & action(t,s,i,j) &
                (? z: pos_auth(t,z,k,a) & can_delegate(t,z)) &
                ~(? z: pos_auth(t,z,k,a) & can_delegate(Next(t),z)). }
```

[3]Note that first-order logic with inductive definitions has an expressivity that lies strictly between the expressivity of first-order and second-order logic.

Informally, the second clause of this definition says that if in a local revocation scheme revoking a positive authorization from principal `i` to principal `j`, `j` is losing its delegation right, then every positive authorization from `j` to another principal `k` must be replaced by a positive authorization of the same authorization type from `i` to `k`. This new authorization from `i` to `k` ensures that the propagation defined in the second clause of the definition of `delete` does not continue beyond `j`.

The predicate `access_right(t,i)` means that principal `i` has access right at time `t`:

```
{       access_right(t,p) <- chain(t,p,a). }
```

4.2 Using inferences to solve various tasks

In this subsection we explain how different logical inferences, when applied to the FO($\cdot$) specification explained above, can solve various tasks that can be useful both for implementing a system which allows for delegation revocation and for supporting a user of such a system.

Let us first consider tasks that a system that implements Hagström et al.'s delegation revocation framework needs to solve. Given a certain state of the system, defined by which authorizations are currently included in the authorization specification, and a given action (a delegation or revocation performed by some principal), the new state of the system after this action needs to be determined. This task can be performed using the Modelexpand inference as follows: Let T be the FO($\cdot$) specification of the delegation revocation framework. Let $\mathcal{S}$ be a partial structure with the following properties:

- The time domain of $\mathcal{S}$ contains only the two time points 0 and 1.
- $\mathcal{S}$ assigns to the predicate `pos_auth_start` the set of all authorizations currently included in the authorization specification.
- $\mathcal{S}$ assigns to the predicate `action` the given action at time 0.
- The value of all other predicates is undefined in $\mathcal{S}$.

In this case, Modelexpand(T,$\mathcal{S}$) is a total structure that expands $\mathcal{S}$ and that is a model of T. Being a total structure, it assigns to `pos_auth` a set A of quadruples of the form (t, i, j, a), where t is a time point (0 or 1), i and j are principals and a is a permission. Then the set $A' := \{(i, j, a) \mid (1, i, j, a) \in A\}$ is the set of authorizations that constitutes the authorization specification after the action. (Note that since all predicates other than `pos_auth_start` and `action` are defined in T through an inductive definition, there is a unique model of T that expands $\mathcal{S}$, so that the result of this inference is deterministic.)

This way we can determine the effect of a single action. It could be iterated by setting the value of `pos_auth_start` to be A' for the next iteration of this procedure. But the IDP system also supports an inference, namely the Progression inference, that is designed for this kind of temporal progression of a structure based on a theory with a type for time. The input structure of this inference provides information about the state of the system on a time point t; in our case that is the authorization specification at a given time. The output is a structure that represents the state of the authorization specification at time point $t + 1$. So the step of extracting A' from A that was required for iterating the above inference is no longer needed. So this inference can more straightforwardly be iterated, giving rise to an interactive simulation of the progression of the system state through time.

Of course, a system implementing Hagström et al.'s framework does not only need to determine how the authorization specification changes over time, but also needs to determine whether a principal requesting access or performing a certain administrative action actually has access right or the right to perform the action in question. This can be done with the Query inference: For example, if $\mathcal{S}$ is the partial structure that assigns to the predicate `pos_auth_start` the set of all authorizations currently included in the authorization specification, Query($\mathcal{S}$,{`i` | `access_right(Start,i)`}) returns the set of principals that have access right according to the current authorization specification.

When designing a system, one can avoid an erroneous design by specifying invariants that the system must satisfy at any moment during the execution of the system, and verify that these invariants are actually satisfied by the system. In the case of a system based on Hagström et al.'s delegation revocation system, an example of an invariant that the system must satisfy is the connectivity property defined in subsection 2.1. We must, of course assume, that the system starts in a state that satisfies the connectivity property. All that remains to be shown, then, is that if the connectivity property holds at some time point t, it must also hold at the next time point $t + 1$.

One way that this can be done is by calling an automated theorem prover to prove this implication. However, this is not always viable, as the theory may be too complex for an automated theorem prover to be able to find a proof of the invariant. This is the case for our specification.

Another possibility is to prove that the invariant holds in fixed structures. In our case, we can fix a partial structure $\mathcal{S}$ with time points 0 and 1 and without any information about the predicates. In this case, the only information that we are fixing is the number of principals. We can then prove that the invariant holds for a fixed number of principals by establishing, using Modelexpand(T',$\mathcal{S}$), that there is no total structure expanding $\mathcal{S}$ that is a model of the theory T' consisting of our specification T together with the statement that the connectivity property holds at time point 0 but not at time point 1. With this method, we have verified the connectivity property for any system with n principals for $n \leq 8$.

Despite this limitation to very small domains, this limited verification can be useful for avoiding erroneous design, as errors tend to already show up at relatively small domains. It should be added that the logical methodology of the KB paradigm lends itself well to the usage of interactive theorem provers common in software verification in order to fully verify invariants over complex specifications. The integration of IDP and interactive theorem provers is, however, still future work.

Finally, let us turn to a task that supports a user of a system based on Hagström et al.'s delegation revocation framework: A principal i may want to reach a certain outcome, e.g. that a given principal j should no longer have a certain right while another principal k is unaffected. i may want to find out all revocation schemes that lead to the desired outcome. This can be achieved by computing Allmodels(T_2,$\mathcal{S}$), where $\mathcal{S}$ is the structure that assigns the current authorization specification to `pos_auth_start`, and T_2 is the theory consisting of our specification of the delegation revocation framework together with the statement that the action at time point 0 is performed by i, and the statement that the desired outcome holds at time point 1. The values of the predicate `action` at time point 0 in the models returned by Allmodels(T_2,$\mathcal{S}$) are the actions that i can perform in order to get the desired outcome.

Furthermore, i may want to reach a certain outcome for some given principals while minimally influencing the permissions of other principals. In that case, i can define a cost function, e.g. that every change in a permission of a principal has a cost of 1, and search the models returned by Allmodels(T_2,$\mathcal{S}$) for the one with the minimal cost.

The IDP prototype that we have built can perform all the different tasks described in this subsection.

5 Related work

While the KB paradigm and its implementation IDP, are fairly young, its applicability has been investigated and illustrated in multiple domains. In [18], the connection with Business Rules was investigated. Business Rules are well-represented in industry for knowledge-intensive applications and as such were used as a comparison to evaluate the KB paradigm. A typical Business Rules application, the EU-Rent Car-Rental company, was modelled in FO($\cdot$), and two use cases were investigated.

In [23], the authors looked at applications of the IDP system, for interactive configuration systems, where the system is used to guide a user through a search space, looking for a valid configuration. The advantages of an explicit modeling of domain knowledge in configuration were large: the adaptability in case the domain knowledge changes and the fact that the same specification of knowledge could be reused in different tasks being the most important. This work was extended in [17], where eight different reasoning tasks used in a configuration system were identified and implemented using logical inferences on 1 knowledge base, containing all domain knowledge.

While the KB paradigm has not been previously applied to the problem of delegation revocation, other logical methods have been applied to this access control problem: Aucher et al. [1] presented a formalization of Hagström et al.'s eight revocation schemes in a dynamic variant of propositional logic that resembles imperative programming languages. Furthermore, they extended their formalization with a notion of trust. Their formalization only supports the tasks of determining the state of the system after a certain action and of determining whether a user has a certain permission given a certain state of the system; the other tasks described in this paper are not supported by their formalization.

Two of the authors of the current paper have defined a modified version of Hagström et al.'s delegation revocation framework as well as *Trust Delegation Logic*, a logic of trust designed for studying the reasons for performing different revocation schemes defined [8]. This work was motivated by problems we discovered with Hagström et al.'s delegation revocation framework when we produced the first FO($\cdot$) specification of the framework, e.g. the problem that variations in the timing of the actions of various principals can have undesirable

side-effects. These problems are documented in [8], [10] and [9], and a systematic methodology to avoid problems of this kind is studied in [9]. As this example shows, formally specifying something in FO($\cdot$) can generally help understanding it better and uncovering problematic features.

6 Conclusion

In this paper, we have explained the benefits of the knowledge base paradigm when applied to delegation revocation. The knowledge base paradigm proposes a strict separation between knowledge and problem solving. In our application, the knowledge is represented by an FO($\cdot$) specification of Hagström et al. [16] delegation revocation framework. By applying various logical inferences to this specification, multiple tasks that arise when implementing or using a delegation revocation system were solved. This way, the same information was reused for solving various problems.

Our work constitutes a proof of concept, and we hope that it will inspire other researchers in computer security to consider the possibility of applying the methodology of the knowledge base paradigm to their research.

References

[1] Guillaume Aucher, Steve Barker, Guido Boella, Valerio Genovese, and Leendert van der Torre. Dynamics in Delegation and Revocation Schemes: A Logical Approach. In Yingjiu Li, editor, *Data and Applications Security and Privacy XXV*, volume 6818 of *Lecture Notes in Computer Science*, pages 90–105. Springer Berlin, 2011.

[2] Steve Barker. The next 700 access control models or a unifying meta-model? In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 187–196. ACM, 2009.

[3] Steve Barker, Guido Boella, Dov Gabbay, and Valerio Genovese. Reasoning about delegation and revocation schemes in answer set programming. *Journal of Logic and Computation*, 2014.

[4] E. Bertino, P. Samarati, and S. Jajodia. An extended authorization model for relational databases. *Knowledge and Data Engineering, IEEE Transactions on*, 9(1):85–101, Jan 1997.

[5] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A Non-timestamped Authorization Model for Data Management Systems. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, CCS '96, pages 169–178, New York, NY, USA, 1996. ACM.

[6] Bart Bogaerts, Joachim Jansen, Maurice Bruynooghe, Broes De Cat, Joost Vennekens, and Marc Denecker. Simulating Dynamic Systems Using Linear Time Calculus Theories. *TPLP*, 14(4–5):477–492, 7 2014.

[7] Ajay Chander, Drew Dean, and John C. Mitchell. Reconstructing trust management. *Journal of Computer Security*, 2004.

[8] Marcos Cramer, Diego Agustín Ambrossio, and Pieter van Hertum. A Logic of Trust for Reasoning about Delegation and Revocation. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, pages 173–184, 2015.

[9] Marcos Cramer and Giovanni Casini. Postulates for Revocation Schemes. In *International Conference on Principles of Security and Trust*, pages 232–252. Springer, 2017.

[10] Marcos Cramer, Pieter Van Hertum, Ruben Lapauw, Ingmar Dasseville, and Marc Denecker. Resilient Delegation Revocation with Precedence for Predecessors is NP-Complete. In *Proceedings of the Computer Security Foundations Symposium (CSF) 2016*, in print.

[11] Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker. Predicate Logic As a Modeling Language: The IDP System. In Michael Kifer and Yanhong Annie Liu, editors, *Declarative Logic Programming*, pages 279–323. Association for Computing Machinery and Morgan & Claypool, New York, NY, USA, 2018.

[12] Marc Denecker and Eugenia Ternovska. A Logic of Nonmonotone Inductive Definitions. *ACM Trans. Comput. Log.*, 9(2):14:1–14:52, April 2008.

[13] Marc Denecker and Joost Vennekens. Building a Knowledge Base System for an Integration of Logic Programming and Classical Logic. In María García de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *LNCS*, pages 71–76. Springer, 2008.

[14] Ronald Fagin. On an Authorization Mechanism. *ACM Trans. Database Syst.*, 3(3):310–319, September 1978.

[15] Patricia P. Griffiths and Bradford W. Wade. An Authorization Mechanism for a Relational Database System. *ACM Trans. Database Syst.*, 1(3):242–255, September 1976.

[16] Åsa Hagström, Sushil Jajodia, Francesco Parisi-Presicce, and Duminda Wijesekera. Revocations – A Classification. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 44–, Washington, DC, USA, 2001. IEEE Computer Society.

[17] Pieter Van Hertum, Ingmar Dasseville, Gerda Janssens, and Marc Denecker. The KB Paradigm and Its Application to Interactive Configuration. In Marco Gavanelli and John H. Reppy, editors, *Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings*, volume 9585 of *Lecture Notes in Computer Science*, pages 13–29. Springer, 2016.

[18] Pieter Van Hertum, Joost Vennekens, Bart Bogaerts, Jo Devriendt, and Marc Denecker. The effects of buying a new car: an extension of the IDP Knowledge Base System. *TPLP*, 13(4–5-Online-Supplement), 2013.

[19] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A Logic-based Approach to Distributed Authorization. *ACM Transaction on Information and System Security*, 2003.

[20] David G. Mitchell and Eugenia Ternovska. A Framework for Representing and Solving NP Search Problems. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 430–435. AAAI Press / The MIT Press, 2005.

[21] Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Well-founded and Stable Semantics of Logic Programs with Aggregates. *TPLP*, 7(3):301–353, 2007.

[22] Roberto Tamassia, Danfeng Yao, and William H. Winsborough. Role-Based Cascaded Delegation. In *Proceedings of the 9th ACM symposium on Access control models and technologies* , 2004.

[23] Hanne Vlaeminck, Joost Vennekens, and Marc Denecker. A logical framework for configuration software. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 141–148. ACM, 2009.

[24] Johan Wittocx, Maarten Mariën, and Marc Denecker. The IDP system: A model expansion system for an extension of classical logic. In Marc Denecker, editor, *LaSh*, pages 153–165. ACCO, 2008.

[25] Danfeng Yao and Roberto Tamassia. Compact and Anonymous Role-Based Authorization Chain. *ACM Transactions on Information and System Security*, 2009.

[26] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A rule-based framework for role-based delegation and revocation. *ACM Transactions on Information and System Security*, 2003.