A General Logic-Based Authorization Model

W.R. Cook,
Department of Computer Science, University of Texas,
Austin, TX 78074, email: wcook@cs.utexas.edu

V.W. Marek
Department of Computer Science, University of Kentucky,
Lexington, KY 40506, email: marek@cs.uky.edu *

June 21, 2011

Abstract

The system-security literature contains numerous papers on authorization which are written according to the following scheme. The authors select a specific situation commonly occurring in a business setting and then describe a formalization of a specific policy that allows for solving the problem of the access to information (in varying granularity: collections of objects, tables, records or even specific fields in records). We discuss several such policies and show how a first-order logic and its extension by means of the fixpoint operation can be used to formalize all these policies.

Keywords

Security policies, access control, first-order logic, first-order logic with fixpoints

1 Introduction

The question of what, actually, constitutes the notion of access policy has been a subject of considerations for computer science and, in particular computer security researchers for over 30 years. The literature of the subject is voluminous and many notable papers have been published or circulated [BL73, BN89, GW76, SFK00, OGM08] among the others. Nevertheless a

*Partially supported by JPL contract 1401954

more principled analysis of the literature of the subject indicates that many papers follows the following scheme: the researchers look at a specific business situation, with its generally accepted constraints, and then formalize a solution tailored to that situation.

In the meanwhile, however, a significant change occurred. While originally the part of the system responsible for the security of the Database Management System (DBMS) was thought as a separate entity, the modern DBMS stores, besides the user-owned data, include the data regarding the users and their properties, as well as the data about the users being present at the system, relationship of the users to the data and other additional information. A consequence of this is that the security properties of the DBMS are themselves expressible in the query language of the database. This position have been taken in [Co03, CG04] and [CG04] and the current paper is a continuation of this work. Specifically, we discuss the security properties of a database as predicates expressible in first-order predicate calculus corresponding to the database treated as a finite relational structure (as this notion is treated in logic) or its extension by transitive closure operation. We take a slightly idealized point of view by not limiting the syntactic form of the definition (while it is likely the definition would be in a simpler fragment of logic) beyond the obvious requirement that the predicate relates the users and resources. We find that two languages (first-order logic and the extension of first-order logic by means of the fixpoint operation) are natural candidates for the logic-based language for formulation of security policies.

The contents of this paper are as follows. In Section 2 we discuss a number of well-known access policies. We also briefly discuss two systems of logic and their semantics. In Section 3 we show how the policies of Section 2 are formalizable in systems of logic. We also show one additional custom-made policy (but certainly corresponding to reality) as formalizable in the proposed framework. In Section 4 we discuss the issue of semantics of access control policies. Finally Section 5 discusses consequences of our approach and also issues related to time.

2 Background

In this section we describe a number of well-known policies that are, actually employed in existing systems. We also refer the reader to [CC04] for the so-call Common Criteria standard for security assurance.

2.1 Mandatory access control

Mandatory access control (MAC) also known as Bell-La Padula model [BL73] is used to prevent information leaking. MAC is a policy based on the idea that database (whose nature is immaterial) has the set of simple *access levels*, usually four values: U, C, S, and TS (but easily generalizable to more levels, or even partially ordered set of levels) that are ordered as follows:

$$U < C < S < TS.$$

Each user, and each document is *marked* with such level. The (simplified) access policy for such system is described by two requirements:

1. (no "write-down") The user with a level L can not *write* to documents of documents of strictly lower level.

2. (no "read-up") The user with a level L can not *write* to documents of documents of strictly higher level.

Additional tasks such as *append*, *learn_scheme* etc. may be added to this mode of access control.

2.2 MAC with Boolean compartments

A modification of MAC is one where the users and documents are marked with *compartments*. Compartments are use for the following business-logic situation. A document is marked by a set of its *topics*. Likewise, a user is marked by the vector of topics she can learn about. Formally, we have a set of *topics* (*compartments*) T. Each user u is labeled with a set $T_u \subseteq T$. Likewise, each document has an associated a set $T_d \subseteq T$. The user u can access the document d if, in addition to the MAC requirements, $T_d \subseteq T_u$.

An example of such situation could be the set of compartments: $\langle \mathit{TargetSharePrice}, \mathit{SizeOfInvestments} \rangle$. When the user *victor* having the security level *confidential* has an empty associated set of compartments and unclassified document *doc1* has the associated set $\langle \mathit{TargetSharePrice} \rangle$, the user *victor* will not get the access to the document *doc1* inspite the fact that *victor*'s security level is higher than that of *doc1*. But the user *william* with the security level *confidential* and the set of compartments $\langle \mathit{TargetSharePrice}, \mathit{SizeOfInvestments} \rangle$ will be able to access the document *doc1*. We observe that a natural implementation of compartments is with two tables with the following schemes: $\langle \mathit{user\_name}, \mathit{compartment} \rangle$ and $\langle \mathit{resource\_name}, \mathit{compartment} \rangle$. Then, the requirement is that whenever the

resource has compartment, then the user must have same compartment (for *all* compartments).

More complex policies where one assigns access levels to compartments are, of course, possible. We note, though, that the implementation still involved tables stored in the database.

2.3 Discretionary Access Control (DAC)

Discretionary access control [GW76] approach, used within relational DBMS, is an access control mechanism allowing for a variety of interpretations. Here, we discuss a simplified version of DAC with the granularity of objects set to tables. We will discuss a graph-theoretic representation of DAC. The multigraph G_T associated with a table o has always an initial node from a special node called *system*. When a table o is created by user (owner) u a number of edges from *system* to u are created. Those edges, are of the form

$$\langle system, u, o, p, g \rangle$$

Here p is one of the permissions (*write, read, update* etc.). The special value g is used in *granting process*. A user v may create an edge to some other user v'

$$\langle v, v', o, p, g \rangle$$

This means that the user v grants to the user v' a permission p on the table o. If g is 1, granting option is given by v to v'. The value p (where p is a permission, as discussed above) induces a subgraph $G_{o,p}$ of G_o. To see if a user u' has a permission p on table o we check if there is a path Q in graph $G_{o,p}$ with the following properties:

1. The beginning of the path Q is the node *system*
2. All the edges on the path Q are of the form $\langle v, v', o, p, g \rangle$
3. All the g values on the edges (*with the possible exception of the last one*) are equal to 1

Let us see what the conditions (1)-(3) mean. In fact one of the following two situations must happen:

1. Either the node u' belongs to the transitive closure H of the singleton set $\{system\}$ under the relation

$$\{\langle v, v' \rangle : \langle v, v', o, p, 1 \rangle\}$$

2. Or, u' belongs to the image of the set H defined above under the relation
$$\{\langle v, v' \rangle : \langle v, v', o, p, 0 \rangle\}$$

It should be clear that to express the DAC in a logical language we will need an extension of first-order logic where we can define an operator of transitive closure of a relation. However if we impose a constraint that the chain of permissions is of length at most k for some prespecified k then such limited form of DAC is first-order definable.

2.4 Roles, and DAC with roles

A (simple) role [SFK00] is a collection of users. Such collections correspond to specific elements of business logic. For instance, in financial organizations one talks about role of *tellers* (or even more specifically, tellers in a specific branch), while in the university setting one deals with *graduate faculty*, *faculty senate*, or *parking committee*. In the context of DAC, roles are treated in a manner analogous to the users, with the role getting specific privileges. We not in passing that a user may be a member of several roles and that this may lead subtle problems with the semantics, esp. when the privileges of various roles are contradictory (for instance with respect to temporal aspects of privileges.) More complex systems of roles are *hierarchies of roles*. An example of such situation is in banking industry where the *staff* role consists of *associate, loan_officer, teller* and *vice_president*. One can formally represent hierarchies of roles in formal systems such as *mereology* [Le16] or *inheritance hierarchies* [To86]. The presence of hierarchies of roles, where the dependence graph is a directed acyclic graph labeled with role names and the *child_of* relationship is represented by the part_of relation requires, like in the case of simple DAC, transitive closure.

2.5 Logics

We will be using two systems of logic in our description of security policies. One of these is standard first-order logic which will be denoted below by FOL. The second one, the fixpoint logic [Li04] denoted below by $FOL(FP)$, is less known but allows for fixpoint (transitive closure) operation.

2.5.1 First-order logic

By first-order logic (FOL) over signature σ we mean a predicate logic with a predicate symbol for each relational symbol R of σ. We assume that there

are no function symbols (operations such as + or · can be treated as built-in). A relational database can be treated as an interpretation of suitably chosen signature (with a relational symbol for each table). Given such first-order logic and a structure $\mathcal{A}$, we denote by $\mathcal{A} \models \varphi[v]$ the fact that variable assignment v satisfies formula φ in structure $\mathcal{A}$. Generally, FOL is studied in courses of logic. Here we refer to [Sh01] for standard description of FOL.

2.5.2 First-order logic with fixpoint operation, $FOL(FP)$

We will also consider an extension of FOL that admits the transitive closure (i.e. fixpoints of reachability relations) as "first-class citizens". Given a signature σ, $FOL(FP)_\sigma$ is a logic with an additional modal operator FP. Given a formula $\varphi(X, Y)$ with two free variables X and Y (additional free variables are possible), $FP_{X,Y}(\varphi(X, Y))$ is also a formula. Its meaning is the following: for every choice of remaining parameters, say $\vec{z}$, the formula $FP_{X,Y}\varphi(X, Y, \vec{z})$ is satisfied by pairs $\langle a, b\rangle$ belonging to the transitive closure of the binary relation R_φ defined by $\{\langle a, b\rangle : \varphi[a, b, \vec{z}]\}$. We will drop the subscript X, Y (and not mention parameters $\vec{z}$) when these are determined by the context. This logic is a fragment of the logic LFP discussed in [Li04], Ch. 10, and in fact its fragment *trcl* discussed in Section 10.6 of [Li04]. The intuition is that we have the relation $R_\varphi(X, Y)$, and then the extent of the relation $FP\varphi(X, y)$ is the collection of all tuples derived by the following DATALOG program:

$$fp(X, Y) \leftarrow R(X, Y)$$
$$fp(X, Y) \leftarrow fp(X, Z), fp(Z, Y)$$

(Here R is $R_{\varphi,X,Y}$.)

Formally, (and without direct reference to DATALOG), we interpret $FP(\varphi)$ as the transitive closure of R_φ. It should be clear that the inductive definition of satisfaction for formulas of the first-order logic FOL_σ can be extended to formulas of $FOL(FP)$. All one needs to do is to add the condition that the variable assignment $\vec{a}$ satisfies $FP(\varphi)$ if the pair $\langle \vec{a}(X), \vec{a}(Y)$ belongs to the transitive closure of R_φ (the parameters in R_φ are passed from $\vec{a}$). Moreover, let us observe that since we deal with finite relational structures (after all the database are finite relational structures) the standard algorithms for computation of transitive closure ([CLR90]) apply in our situation. It should also be observed that it is known that due to the compactness of the logic FOL, $FOL(FP)$ does not reduce to FOL [AU79]. Thus, in principle, $FOL(FP)$ properly extends FOL.

3 Logic-Based Authorization

Let S be a schema that describes a set of legal database instances. We identify S with the set of all instances of S. Let $D \in S$ be a *database*. The nature of the schema S and databases D is, for a moment immaterial; we can think about D as a depository of documents, a relational database, or yet another collection of objects, where S constrains the structure of D. Example schemas include many-sorted algebra [KK71] or triple-stores [RDF04]. We assume that any particular database D is finite, but the set of all possible databases S may be infinite.

Let O be a set of operations that can be performed upon the database. Assume that the operations can refer to elements of the database and also include additional values v. Example operations are $read(d)$, $write(d, v)$, $delete(d)$, or $insert(v)$. Thus d refers here to some object (document, tuple) present in D, and v refers to an object that is placed in D or written into a specific object d of D. In what follows we will also sometimes consider *permissions* a set of permissions which are the string part of an operation with only one database argument. For example, the permission *read* is associated with $read(d)$ operations.

let U be a table in D representing *users*. Again, the natures of the users is left open for now.

A *policy* is a formula $\varphi(U, O, S)$ (i.e. one involving predicate symbol U describing users, the predicate symbol O describing objects, and possibly other predicate symbols that describe other tables in D. As any formula, for specific user u, object o and other relations (tables) interpreting symbols occurring in φ P returns the Boolean value 1 or 0 depending whether the formula φ is satisfied or not satisfied in D.

A policy is understood as specifying, for a given database $D \in S$, whether a user in $u \in U$ can perform operation $o \in O$ on $d \in D$. The above is a very general notion of an authorization policy.

It is also useful in practice to include time as an additional argument in the policy. For instance we may want to be able to declare the policy permitting tellers to access client's accounts but only on Mondays through Fridays, and between 9a.m. and 3p.m. Other information, such as *geolocations*, *rank in the organization*, and other attributes may be assumed to be part of D and could be used in a specification of a policy.

One key point is that policies are usually infinite, meaning that they cannot be expressed as a finite table or access control matrix. This is because the set of databases, the set of operations, and the set of times are all potentially infinite. We want to describe policies that apply to all possible

databases and operations, not just describe policies for a specific database. Any approach based on finite enumeration of cases, will be fundamentally less expressive than the general form of policy describe above.

On the other hand, it is important to be able to compute particular instances of the policy efficiently. In particular, given a specific database D, user u, operation o it must be possible to compute the value assigned by the policy on $\langle u, o, d\rangle$ (given the finite database D). In this case all the inputs are specific finite values.

We want to represent uniform policies compactly and efficiently using some form of first-order predicate logic or its extensions. In such way we will also get a clean semantics, inherited from the corresponding logic-based language.

We will now show how the four policies discussed in Section 2 can be formalized in appropriately chosen logic.

3.1 Expressing MAC in FOL

We will first consider the case of MAC without compartments. The operations will be of two types only: $read(d)$ and $write(d)$. In this model the data being written is not relevant to the authorization decision, so it is suppressed here. The level access of users are represented as a binary relation D_{user_perm} in the database. Here a pair (u, l) belongs to the relation D_{user_perm} if user u has the level access l. Users u range over users; the levels range over the set $\{U, C, S, TS\}$. We assume that for each user u there is exactly one pair (u, l) in the relation D_{user_perm}. Likewise, the database contains a binary relation D_{doc_perm}. In this relation, a pair (d, l) in D_{doc_perm} expresses the fact that the document d has an access level l. In Section 2.1 we defined two basic policies. Now we express them as formulas that need to be true in the database. Let us first look at "no write-down" condition. We define a policy predicate $MAC_w(u, write(d), D)$ as follows:

$MAC_w(u, write(d), D)$ if and only if

$$\exists l_1, l_2 : (D_{user\_perm}(u, l_1) \wedge D_{doc\_perm}(d, l_2) \wedge (l_1 < l_2 \vee l_1 = l_2)$$

The policy predicate MAC_w expresses the "no write-up" policy. We will require that for a user u to modify the document d using operation *write(d)* on database D under MAC, the predicate MAC_w must be true in the database (treated as a first-order structure) on the tuple $(u, write(d), D)$. The relation $<$ was defined above in Section 2.1 and is stored in a table.

The second requirement ("no read-up") can be expressed by a similar predicate $MAC_r(V, D)$ as follows.

$MAC_r(u, read(d), D)$ if and only if

$$\exists l_1, l_2 : (D_{user\_perm}(u, l_1) \wedge D_{doc\_perm}(d, l_2) \wedge (l_2 < l_1 \vee l_1 = l_2)$$

Now the overall mandatory access control policy is

$$MAC(u, o, D) = MAC_r(u, o, D) \vee MAC_w(u, o, D)$$

We observe that since there is precisely one entry in the relation (table) *user_perm* for each user u (and similarly exactly on entry in the relation *doc_perm* for each document d, the right-hand-sides of definitions of relations MAC_r and MAC_w can be changed to universal formulas. For instance, an equivalent form of the definition of MAC_r is:

$MAC_w(u, write(d), D)$ if and only if

$$\forall l_1, l_2 : (D_{user\_perm}(u, l_1) \wedge D_{doc\_perm}(d, l_2) \wedge (l_2 < l_1 \vee l_1 = l_2)$$

3.2 Expressing MAC with compartments in FOL

We can also use FOL to define MAC with compartmentalized knowledge. The number of compartments does not matter, but our access predicates will have an additional placeholder for each compartment. In our example in Section 2.2, we had two compartments: *TargetSharePrice*, and *SizeOfInvestments*. The database is extended with two predicates D_{user_comp} and D_{doc_comp} of type (u, c) where c is a compartment. Unlike the level predicates, the compartment predicates can define multiple compartments for a user or document. The idea is that if the document d and the user have any compartments in common then the user can access the document. We show the modified definition of $CMAC_r$, leaving to the reader the suitable modification of $CMAC_w$.

$CMAC_r(u, read(d), D)$ if and only if $MAC_r(u, read(d), D)$

$$\wedge \ \forall c : (D_{doc\_comp}(u, c) \Rightarrow D_{user\_comp}(d, c))$$

Note that the definition of MAC with compartments is a natural extension of the original definition of MAC. We also observe that a subtler policies that control the assignment of users and document to compartments are also possible. In this case the set of operations would be extended to include operations *add(u,c)*, *remove(u,c)*, *add(d,c)*, *remove(d,c)*. To specify a useful policy controlling compartments, a form of role or category manager is needed. This kind of extension is described in the next section.

3.3 Expressing RBAC in FOL

Role-based security (RBAC) is a NIST standard. A *role* acts as the nexus between users and permissions and provides a level of abstraction so that consistent permissions can be assigned to groups of users. There is a hierarchy of RBAC models with different levels of complexity. In the simple version given below, the focus is on users, roles, and permissions. We have a set of permissions P (which could be implemented by a unary table with the same name), the set of users U, the set of roles R, and the set of objects O. The permissions on objects are given to roles, not to users. That is, we have a table PR so that $PR \subseteq P \times O \times R$, and a table specifying the membership of users in roles UR (i.e. $UR \subseteq U \times R$.) With these tables (thus predicate symbols describing them) we define:

$$\mathrm{RBAC}(u, p, o) \text{ if and only if } \exists r : \mathrm{UR}(u, r) \wedge \mathrm{PR}(p, o, r)$$

The RBAC policy allows an operation p to be performed if the user u is authorized for a role r that is authorized to perform the operation. Thus the specific users are isolated from the explicit permissions by means of roles. A more complex role-based policy (based on a hierarchy of roles) will be discussed in Section 3.5

3.4 Formalizing DAC in $FOL(FP)$

To formalize DAC, we define three operations *read(d), write(d), delete(d)* with associated permissions *read, write, delete*. The objects in the database could be tables, rows, or individual cells of the database, depending on the intended granularity of permissions. The logic that will be used to formalize the access policy is $FOL(FP)$, since, as described in Section 2.3, we need to refer to transitive closure.

The data used in discretionary access control is stored in 5-ary relation D_A consisting of tuples (records) $\langle u_1, u_2, d, q, g\rangle$ where U_1, u_2 are users, d is a database object, q is the name of the permission (see above), and g belongs to *Bool* (intuitively is that the permission to do q on t is given by u_1 to u_2 without, or with grant option depending whether g is 0 or 1.) Thus we have a 5-ary predicate letter $DAC(u_1, u_2, d, q, g)$. given a database object d and the type of access q, we define a number of auxiliary binary predicates.

First, we define the predicate *weak-grant* by

$$\textit{weak-grant}(u_1, u_2, d, q) \text{ if and only if } \exists g : DAC(u_1, u_2, d, q, g)$$

and another predicate *full-grant* by

$$\textit{full-grant}(u_1, u_2, d, q) \text{ if and only if } DAC(u_1, u_2, t, q, 1)$$

The predicate *weak-grant* tells us which users are connected by permission q on object d (with, or without grant option). The predicate *full-grant* limits *weak-grant* to pairs of users that are connected by permission q with grant option.

Next we let $\textit{fp-grant}(u_1, u_2, d, q)$ to be $FP(\textit{full-grant}(u_1, u_2, d, q))$, where the fixpoint is computed with respect to the variables u_1 and u_2 (since our logic is $FOL(FP)$ it is a formula of our logic). We now form the predicate $\textit{grant}(u_1, u_2, d, q)$ as follows:

$$\begin{aligned}\textit{grant}(u_1, u_2, q) \text{ if and only if } (&\textit{fp-grant}(u_1, u_2, d, q)\\ &\vee \exists u_3 : (\textit{fp-grant}(u_1, u_3, d, q) \wedge \textit{weak-grant}(u_3, u_2, d, q)).\end{aligned}$$

Intuitively $\textit{grant}(u_1, u_2, d, q)$ means that there is a path from u_1 to u_2 where on that path all transitions correspond to grants of permission of q with further grant option, possibly with the exception of the last one (this is the meaning of the second term of disjunction on the right-hand side).

Once we defined the accessibility predicate *grant*, we can define the ternary predicate DAC defining the set of users that have the access to the object d with the permission q as follows:

$$DAC(u, q, d) \text{ if and only if } \textit{grant}(\textit{system}, u, d, q).$$

Here *system* is the user described in Section 2.3. One can easily prove by induction on the length of the path that the predicate DAC correctly describes the DAC policy.

The set of operations and permissions can be extended to allow users to grant or revoke permissions, e.g. $\textit{grant}(\textit{read}, d)$ with permission *grant*. The right to grant permissions is also controlled by access control policies. We observe that stronger predicate describing the set of users that have not only the access with permission q but also can grant such permission to others, can also be defined by:

$$SDAC(u, , q, d) \text{ if and only if } \textit{full-grant}(\textit{system}, u, d, q).$$

3.5 Formalizing hierarchical RBAC in $FOL(FP)$

We will now show how the $FOL(FP)$ can be used to formalize role-based access control with a hierarchy of roles. While it is possible to formalize both delegation and hierarchy of roles within a single formalism, in this section we formalize the discretionary access control with the hierarchy of roles only, leaving the merging of two mechanisms (delegation and role-hierarchy) to the reader.

The natural ordering of roles (the reverse inclusion relationship) that defines the relationship "part_of" can be naturally represented by the table of its Hasse diagram (i.e. "immediate_part_of"). Such table is much smaller than the full diagram of the reverse inclusion relation, and the full diagram can be reconstructed from such relation by means of transitive closure operation. We now proceed similarly to our construction of Section 3.3. Let UR, PR be tables defined in that Section. Additionally, we implement a binary table HDR that contains the information about the Hasse diagram of the "part_of" relation. As before we identify the table names with the corresponding predicate symbols of the language. We then can define

$$\text{HRBAC}(u, p, o) \text{ if and only if } \exists r_1, r_2 : \text{UR}(u, r_1) \wedge FP(HDR)(r_1, r_2) \wedge \text{PR}(p, o, r).$$

3.6 Formalizing MAC with roles

The logic $FOL(FP)$ can also be used to represent the discretionary access control with roles. We discussed in Section 2.4 the concept of roles within the relational model of databases. As observed in Section 2.4 there are slight differences depending whether the roles form a flat set, or they form a proper hierarchy. When roles are flat, an axiom expressing the fact that once a role carries some privileges each user of that role caries the same privileges. This property is an integrity constraint; and it can be expressed as a logical formula. The formula $in(V, R)$ is a binary predicate with the meaning of $in(u, r)$ as follows: "the user u has a role r"

$$\forall v, r, t, p, g : (access(r, t, p, g) \wedge in(v, r) \Rightarrow (access(v, t, p, g)).$$

Here the predicate *access* is the access predicate defined in Section 3.4.

The presence of hierarchical roles requires adding similar axioms but not only for users and roles but also for roles being part of other roles.

3.7 Other security models and specific policies

Other security models (for instance Biba model, also known as Reverse Bell- La Padula model, a model for authoritative information access) can be represented in *FOL* by predicates definable very similarly to those defined in Section 3.1. Likewise, the Brewer-Nash policy (Chinese Wall) [BN89] can easily be expressed as a logic-based access predicate. We will not discuss these models here, referring the reader to [CFMS94, Go06].

We want, however, to illustrate the power of logic-based policies with another example of a policy easily expressible in such formalism. Moreover, we will look at a different granularity of a policy, this time accessing records and not tables. Our goal now is to formalize the access predicate for the following policy for access of personal records: *The personal record of an employee can be viewed by:*
(a) Employee her/himself
(b) The manager of the department where the employee works
(c) Personnel of HR department (unless the employee whose record is viewed is employed in HR department itself, in which case only first two clauses apply)

To formalize this policy we will make assumptions about tables stored in the database. Specifically, we assume that the *employee* table has within its scheme the attributes: *id* (employee id), *position* (to denote the position of the employee in the department), and *did* - the department name. Likewise, we assume that the scheme of *emp_record* contains the attributes *did* and *id* to denote the department and employee id, resp.
Since we use the language of logic, we will use the attribute name as a function symbol, writing, for instance, $id(v)$ instead of $v.id$. The variable r ranges (in this example) over personal

Here is the logic-based representation of the policy described above:

$$\begin{aligned} access(v,r) \text{ if and only if } & [id(v) = id(r)] \vee \\ & [did(v) = did(r) \wedge position(v) = mgr] \vee \\ & [(did(v) = HR) \wedge (did(r) \neq HR)] \end{aligned}$$

4 Semantics of policies

The use of logic as means to define access predicates (thus policies for access) as done in Sections 2 and 3 provides an additional benefit of a providing semantics to access policies. Specifically, both first-order logic *FOL* and its

extension $FOL(FP)$ have semantics, that is for any formula $\varphi(\vec{X})$ and a database D there is a unique relation R_φ consisting of the tuples (variable assignments $\vec{x}$ such that $D \models \varphi[\vec{x}]$. This is the meaning that we assign to the logic-based policy. We also observe that such approach gives a sufficient condition for persistence of policies. Every formula is built from predicate symbols that describe database relations (tables). When these tables are *not* updated the logical value of $\varphi[\vec{x}]$ does not change either (as is easily shown by induction on the complexity of the formula φ). Therefore, if these tables are not updated; the *access* predicate does not change as the result of the update.

We observe that when (as discussed in Section 1 the login information is a part of the database (as in all modern database systems), the very fact of login becomes an update and the access predicate may change. The consequences of this fact are discussed in [OGM08].

5 Conclusions and Further Work

While we discussed definition of access policies using some logics (first-order logic, and its extension by means of fixpoint operation) these are not the only logics that can be used to define policies. Besides of logics such as monadic second-order logic (that is *FOL* but with variables for subsets of the universe and quantifiers for such variables) an important class of logics that potentially could be used for access control are temporal logics, especially with operators for duration of actions. For instance we may want to log out an inactive user (i.e. update the table of logged-in users) after a specific time of inactive behavior. This requires introduction of temporal constants with a varying interpretation such as *today*, *now*, *time_anchor* (last time the user was active) etc. With sufficient care (and limiting only to addition, so Presburger arithmetic algorithm could be used) such extension of first-order logic is certainly possible.

References

[AU79] Aho, A.V. and Ullman, J.D. Universality of data relational languages. 6$^{\text{th}}$ *ACM Symposium on Principles of Programming Languages*, pages 110–119, 1979.

[BL73] Bell, D.E. and La Padula, L.J. Secure Computer Systems, Mathematical Foundations. *Mitre Corporation*, 1973.

[BN89] Brewer, D.F.C, and Nash, M.J. The Chinese Wall security policy. *Proceedings of IEEE Security and Privacy*, 1989.

[CC04] Common Criteria for Information Technology Security Evaluation, CCIMB-2004-01-002, 2004.

[CFMS94] Castano, S., Fugini, M.G., Martella, G. and Samarati, P. *Database Security*, Addison Wesley, 1995.

[Co03] Cook, W. Policy-Based Authorization, Unpublished Manuscript, 2003.

[CG04] Cook, W.R., and Gannholm, M.R. Rule-based database security system and method. U.S. Patent 6,820,082, 2004.

[CLR90] Cormen, T.H., Leiserson, C.E., and Rivest, R.L. *Introduction to Algorithms*, MIT Press/Mc Graw Hill, 1990.

[Go06] Gollmann, D. *Computer Security*, Wiley, 2006.

[GW76] Griffith, P. and Wade, B. An authorization mechanism for relational database systems. *ACM Transactions on Database Systems* 1:242-255, 1976.

[KK71] Kreisel, G. and Krivine, J.L. Elements of Mathematical Logic, North Holland, 1971.

[Le16] Leśniewski, S. Foundations of the General Theory of Sets I (in Polish). (Eng. trans. by D. I. Barnett: Foundations of the General Theory of Sets. I, in S. Leśniewski, Collected Works, Vol. 1, pages 129-173 Kluwer, 1992), 1916.

[KQ+10] Kitchin, D., Quark, A., Cook, W., and Misra, J. The ORC Programming Language. Unpublished manuscript, 2010.

[Li04] Libkin, L. *Elements of Finite Model Theory*, Springer-Verlag, 2004.

[OGM08] Olson, L.E., Gunter, C.A., and Madhusudan, P. A Formal Framework for Reflective Database Access Control Policies. *Proceedings of CCS'08*, 2008.

[SFK00] Sandhur, R., Ferraido, D.F., and Kuhn, D.R. NIST Model for Role-Based Access Control, Towards a Unified Standard, *ACM Workshop on Rule-Based Access Control*, pages 46–63, 2000.

[RDF04] Resource Description Framework, `http://www.w3.org/RDF/`, 2004.

[Sh01] Shoenfield, J.R. *Mathematical Logic*, A.K. Peters/ASL (reprint), 2001.

[To86] Touretzky, D.S., *The Mathematics of Inheritance Systems*, Pitman/Morgan Kaufmann, 1986.

Use your QR-barcode reader to get to the e-repository of my papers.