21

A LOGICAL FORMALIZATION FOR SPECIFYING AUTHORIZATIONS IN OBJECT-ORIENTED DATABASES

Yun Bai and Vijay Varadharajan

Abstract In this paper, we propose a logic formalization to specify authorizations in object-oriented databases. Our formalization has a high level language structure, and it can be used to specify various types of authorizations associated with object oriented databases. Formal syntax and semantics are also provided for the formalization.

Keywords: Authorization policy, formal specification, logic reasoning, object-oriented database, security

1. INTRODUCTION

Authorization specification in object-oriented databases has being investigated by many researchers [4, 6, 8, 9]. However, most of the work to date suffers from a lack of formal logic semantics to characterize different types of inheritance properties of authorization policies among complex data objects. In this paper, we address authorizations in object-oriented databases from a formal logic point of view. In particular, we propose a logical language that has a clear and declarative semantics to specify the structural features of object-oriented databases and authorizations associated with complex data objects in databases. Our formalization characterizes the model-theoretic semantics of object-oriented databases and authorizations associated with them. A direct advantage of this approach is that we can formally specify and reason about authorizations on data objects without loosing inheritance and abstraction features of object-oriented databases. We first propose a logical language for specifying object-oriented databases. This language has a high level syntax and its semantics shares some features of Kifer *et al*'s F-logic [7]. We then

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35508-5_22

V. Atluri et al. (eds.), *Research Advances in Database and Information Systems Security*

extend this language for authorization specification. The semantics of the resulting language is defined in such a way that both the inheritance property in an object-oriented database (OODB) and authorization rules among different data objects can be formally justified.

The paper is organized as follows. In section 2, we first propose a formal language $\mathcal{L}$ that can be used to specify object-oriented databases. A model-theoretic semantics of $\mathcal{L}$ is also provided in this section. In section 3, we extend $\mathcal{L}$ to language $\mathcal{L}^a$ by combining authorization specification associated with data objects into a database. The semantics of $\mathcal{L}^a$ is also provided. Finally, we discuss some related work and our future work in section 4.

2. FORMAL LANGUAGE L FOR OBJECT-ORIENTED DATABASES SPECIFICATION

2.1 SYNTAX OF L

The *vocabulary* of language $\mathcal{L}$ which is used to specify object-oriented database consists of:

1. A finite set of *object variables* $\mathcal{OV} = \{o, o_1, o_2, \cdots\}$ and a finite set of *object constants* $\mathcal{OC} = \{O, O_1, O_2, \cdots\}$. We will simply name $\mathcal{O} = \mathcal{OV} \cup \mathcal{OC}$ as *object set.*
2. A finite set $\mathcal{F}$ of function symbols as *object constructors* or *methods* where each $f \in \mathcal{F}$ takes objects as arguments and maps to an object or a set of objects.
3. Auxiliary symbols $\Rightarrow$ and $\mapsto$.

An *object proposition* is an expression of the form

$$\begin{aligned} O \textbf{ has method } & f_1(\cdots) \Rightarrow \Pi_1, \\ & \cdots, \\ & f_m(\cdots) \Rightarrow \Pi_m, \\ & f_{m+1}(\cdots) \mapsto \Pi_{m+1}, \\ & \cdots, \\ & f_n(\cdots) \mapsto \Pi_n. \end{aligned} \tag{1}$$

In ((1)) O is an object from $\mathcal{O}$ and $f_1, \cdots, f_n$ are function symbols. Each function symbol f takes objects as arguments and maps to some Π that is an object or a set of objects. In an object proposition, a method with the form $f(\cdots) \Rightarrow \Pi$ indicates that f's arguments represent the types of actual objects that should be taken in any instance of this object proposition, and f returns a

set of types of the resulting object/objects. On the other hand, a method with the form $f(\cdots) \mapsto \Pi$ indicates that f takes actual objects as arguments and returns an actual object or a set of objects.

An *isa proposition* of $\mathcal{L}$ is an expression of one of the following two forms:

$$O \textbf{ isa member of } C, \tag{2}$$

$$O \textbf{ isa subclass of } C, \tag{3}$$

where O and C are objects from $\mathcal{O}$, i.e., O and C may be object constants or variables. Clearly, isa propositions (1.2) and (1.3) explicitly represent the hierarchy relation between two objects. An isa proposition without containing any object variables is called *ground isa proposition.*

We call an object or isa proposition a *data proposition.* A data proposition is called *ground data proposition* if there is no object variable occurrence in it. We usually use notation ϕ to denote a data proposition. We assume that any variable occurrence in a data proposition is universally quantified.

A *constraint proposition* is an expression of the form

$$\phi \textbf{ if } \phi_1, \cdots, \phi_k, \tag{4}$$

where $\phi, \phi_1, \cdots, \phi_k$ are data propositions. A constraint proposition represents some relationship among different data objects. With this kind of proposition, we can represent some useful deductive rules of the domain in our database. A *database proposition* is an object proposition, isa proposition, or constraint proposition.

We can now formally define our object-oriented database as follows.

Definition 23 *An* object-oriented database Σ *is a triplet* (Γ, Δ, Ω), *where* Γ *is a finite set of ground object propositions,* Δ *is a finite set of ground isa propositions, and* Ω *is a finite set of constraint propositions.*

Example 11 *We consider a simplified domain about research people in a computer science department. The structure of such domain is illustrated as the following figure.*

In Figure 21.1, line arrows indicate subclass relations while dot-line arrows indicate membership relations in the database.

Using our language $\mathcal{L}$, *our database* $\Sigma = (\Gamma, \Delta, \Omega)$ *is specified as follows: (1) the set of ground object propositions* Γ *consists of:*

$$\begin{aligned} ResPeople \textbf{ has method } name &\Rightarrow String, \\ age &\Rightarrow Integer, \\ firstdegree &\Rightarrow \text{'Bachelor'}, \end{aligned} \tag{5}$$

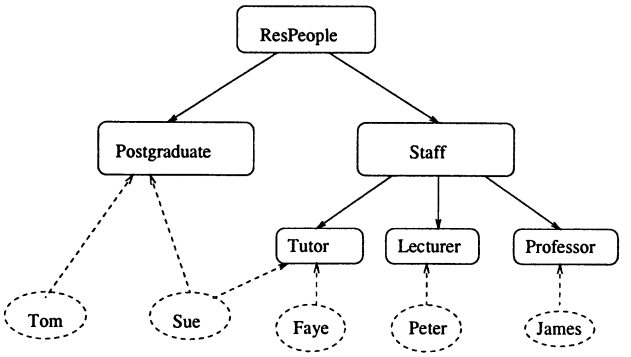

Figure 21.1 A research people database.

$$Postgraduate \textbf{ has method } id \Rightarrow Integer,$$
$$degree \Rightarrow String,$$
$$area(Staff) \Rightarrow String, \quad (6)$$
$$\cdots \quad (7)$$
$$Tutor \textbf{ has method } topsalary \mapsto \text{'\$45,000'}, \quad (8)$$
$$\cdots \quad (9)$$
$$Tom \textbf{ has method } name \mapsto \text{'Tom'},$$
$$age \mapsto 21,$$
$$id \mapsto 96007,$$
$$degree \mapsto \text{'Master'},$$
$$area(Peter) \mapsto \text{'Database'}, \quad (10)$$

(2) the set of ground isa propositions Δ *consists of:*

$$Tom \textbf{ isa member of } Postgraduate, \quad (11)$$
$$Sue \textbf{ isa member of } Postgraduate, \quad (12)$$
$$Sue \textbf{ isa member of } Tutor, \quad (13)$$
$$Faye \textbf{ isa member of } Tutor, \quad (14)$$
$$Ann \textbf{ isa member of } Lecturer, \quad (15)$$
$$James \textbf{ isa member of } Professor, \quad (16)$$
$$Postgraduate \textbf{ isa subclass of } ResPeople, \quad (17)$$
$$Staff \textbf{ isa subclass of } ResPeople, \quad (18)$$
$$Tutor \textbf{ isa subclass of } Staff, \quad (19)$$
$$Lecturer \textbf{ isa subclass of } Staff, \quad (20)$$

$$Professor \textbf{ isa subclass of } Staff, \tag{21}$$

(3) Ω consists of two constraint propositions:

$$\begin{aligned} y \textbf{ isa member of } Staff \\ \textbf{if } x \textbf{ isa member of } Postgraduate, \\ x \textbf{ has method } area(y) \mapsto z, \end{aligned} \tag{22}$$

$$\begin{aligned} y \textbf{ has method } research \mapsto \{..., z, ...\} \\ \textbf{if } x \textbf{ isa member of } Postgraduate, \\ x \textbf{ has method } area(y) \mapsto z, \end{aligned} \tag{23}$$

where x, y and z are object variables, and notation $\{..., z, ...\}$ means that set $\{..., z, ...\}$ includes element z while other elements in the set are not interested in here.

In Σ, Γ and Δ represent explicit *data object descriptions and hierarchical relations among these objects, while Ω describes constraints of the domain which characterize some* implicit *data objects an d their properties. By using these rules in Ω and facts in $\Gamma \cup \Delta$, we actually can derive new data objects with some clear properties. For instance, in the above database, we do not give explicit description about object Peter. But from proposition ((10)) and ((11)) about object Tom, we can derive the facts that Peter is a member of Staff and has a research field* 'Database'.

2.2 SEMANTICS OF L

In this subsection, we define the semantics of our database language $\mathcal{L}$ by following a similar way in classical logic.

Definition 24 *Let $\mathcal{L}$ be an object-oriented database language we defined earlier. A* structure *of $\mathcal{L}$ is a tuple $I = (U, \mathcal{F}_I, \subseteq_U, \in_U, \Rightarrow_I, \mapsto_I)$, where*

1. *U is a nonempty set called the* universe *of I that representing the set of all* actual *objects in the domain.*

2. *For each n-ary function symbol f in $\mathcal{F}$, there exists a n-ary function f_I: $U^n \to U$ in $\mathcal{F}_I$. For $n = 0$, f_I is an element of U.*

3. *$\subseteq_U$ is a partial ordering on U and $\in_U$ is a binary relation on U. We require that if $a \in_U b$ and $b \subseteq_U c$, then $a \in_U c$.*

4. *For symbol $\Rightarrow$ in $\mathcal{L}$, $\Rightarrow_I$ is a map: $\Rightarrow_I$: $U \to (H_0, \cdots, H_i, \cdots)$, where each H_i is a set of $(i+1)$-ary* anti-monotonic *functions with respect to*

ordering $\subseteq_U$[1] *such that for every* $h_i \in H_i$,

$$h_i : U^{i+1} \rightarrow \mathcal{P}\uparrow(U), \tag{24}$$

in which $\mathcal{P}\uparrow(U)$ *is the set of all* upward-closed *subsets of* U *with respect to* $\subseteq_U$.

5. *For symbol* $\mapsto$ *in* $\mathcal{L}$, $\mapsto_I$ *is a map:* $U \rightarrow (G_0, \cdots, G_k, \cdots)$, *where* G_k *is a set of* $(k+1)$*-ary functions such that for every* $g_k \in G_k$,

$$g_k : U^{k+1} \rightarrow U \cup \mathcal{P}(U), \tag{25}$$

in which $\mathcal{P}(U)$ *is the set of all subsets of* U.

Now let us look at this definition more closely. In a structure I, U represents all possible *actual* objects in the domain. That is, each element in U is a real object in our world. Then a function symbol (i.e. object constructor) f in $\mathcal{F}$ is corresponding to a function f_I in $\mathcal{F}_I$. Note that f takes object constants or variables in $\mathcal{O}$ as arguments while f_I takes elements in U as arguments and returns an element of U. The function of ordering $\subseteq_U$ is to represent the semantics of isa subclass proposition in $\mathcal{L}$. For example, $a \subseteq_U b$ is the counterpart of proposition A **isa subclass of** B, while A and B are elements in $\mathcal{O}$ (i.e. object constants or variables) and are mapped to a and b, which are the elements of U respectively. We also write $a \subset_U b$ if $a \subseteq_U b$ but $a \neq b$. The semantics of isa membership proposition in $\mathcal{L}$ is provided by $\in_U$ in I in a similar way.

The semantics of $\Rightarrow$, however, is not quite straightforward. As we mentioned earlier, a method with the form $f(\cdots) \Rightarrow \Pi$ actually defines the function type of f. That is, f takes objects that represent types of actual objects and returns an object (or a set of objects) that indicates the type (or types) of resulting actual object (or objects). Suppose f is a i-ary function. Then the semantics of $\Rightarrow$ is provided by mapping $\Rightarrow_I^{(i)}$[2] that maps the resulting object represented by $f(\cdots)$ to a $(i+1)$-ary function $h_i : U^{i+1} \rightarrow \mathcal{P}\uparrow(U)$, where the first ith arguments in U^{i+1} are objects that corresponds to the i arguments taken by f, and the $(i+1)$-th argument in U^{i+1} is the object that corresponds to the object associated with function $f(\cdots)$ in the proposition (we also call the *host object* of f). In $f(\cdots) \Rightarrow \Pi$, Π denotes the type/types of resulting object/objects for which we use a subset of U to represent all the possible actual objects that have type/types indicated by Π.

---

[1] That is, if $u, v \in U^{i+1}$ and $v \subseteq_U u$, then $h_i(v) \supseteq h_i(u)$.

[2] In expression $\Rightarrow_I : U \rightarrow (H_0, \cdots, H_i, \cdots)$, we use notation $\Rightarrow_I^{(i)}$ to denote the i-th component of $\Rightarrow_I$, i.e. $\Rightarrow_I^{(i)} : U \rightarrow H_i$.

It is important to note that we require the subset of U to be *upward-closed* with respect to ordering $\subseteq_U$. A subset V of U is upward-closed if for $v \in V$ and $v \subseteq_U v'$, then $v' \in V$. The purpose of this requirement is that if V is viewed as a set of classes, upward closure ensures that for each class $v \in V$, V also contains all the superclasses of v, which will guarantee the proper inheritance property of types.

A similar explanation for $\mapsto_I$ can be given for the semantics of $\mapsto$. We now show that $\Rightarrow_I$ actually provides the type of the corresponding $\mapsto_I$.

To simplify our formalization, we will use *Herbrand universe* in any structures of $\mathcal{L}$. That is, the Herbrand universe U_H is formed from the set of all object constants in $\mathcal{OC}$ and the objects built by function symbols on these object constants.

Definition 25 *Let* $I = (U_H, \mathcal{F}_I, \subseteq_I, \in_I, \Rightarrow_I, \mapsto_I)$ *be a structure. We define* entailment *relation* $\models$ *as follows.*

1. *For a ground isa membership proposition,* $I \models O$ **isa member of** C *if* $O \in_{U_H} C$, *and for an isa subclass proposition,* $I \models O$ **isa subclass of** C *if* $O \subseteq_{U_H} C$[3].

2. *For a ground object proposition,*

$$I \models O \textbf{ has method } f_1(\cdots) \Rightarrow \Pi_1, \quad f_n(\cdots) \Rightarrow \Pi_n, \quad \cdots,$$

   if the following conditions hold:

   - *for each* $f(O_1, \cdots, O_p)$ *where* f *is in* $\{f_1, \cdots, f_n\}$, $\Rightarrow_I^{(p)} (O')(O_1, \cdots, O_p, O) = \Pi$, *where* $f_I(O_1, \cdots, O_p) = O'$, *and*

3. *For a ground constraint proposition,* $I \models \phi$ **if** $\phi_1, \cdots, \phi_k$ *if* $I \models \phi_1, \cdots, I \models \phi_k$ *implies* $I \models \phi$.

4. *For any proposition* ψ *including object variables,* $I \models \psi$ *if for every instance* ϕ *of* ψ *(i.e.* ϕ *is obtained from* ψ *by substituted each variable in* ψ *with some element of* U_H*),* $I \models \phi$.

Now we can formally define the model of a database Σ as follows:

[3]Note that under Herbrand universe U_H, an object constant is mapped to itself in U_H.

Definition 26 *A structure M of $\mathcal{L}$ is a* model *of a database $\Sigma = (\Gamma, \Delta, \Omega)$ if*

1. *For each proposition ψ in $\Gamma \cup \Delta \cup \Omega$, $M \models \psi$.*

2. *For each object proposition ϕ, if $M \models \phi$, then $M \models \phi'$ where ϕ' is obtained from ϕ by omitting some methods of ϕ.*

3. *For any isa proposition O* **isa member of** *C and object propositions C* **has method** *$f(\cdots) \Rightarrow \Pi$ and C* **has method** *$f(\cdots) \mapsto \Pi$,*
   (1) $M \models O$ **isa member of** *C and $M \models C$* **has method** *$f(\cdots) \Rightarrow \Pi$ imply $M \models O$* **has method** *$f(\cdots) \Rightarrow \Pi$;*
   (2) $M \models O$ **isa member of** *C and $M \models C$* **has method** *$f(\cdots) \mapsto \Pi$ imply $M \models O$* **has method** *$f(\cdots) \mapsto \Pi$.*

4. *for any isa proposition O* **isa subclass of** *C and object proposition C* **has method** *$f(\cdots) \mapsto \Pi$, $M \models O$* **isa subclass of** *C and $M \models C$* **has method** *$f(\cdots) \mapsto \Pi$ imply $M \models O$* **has method** *$f(\cdots) \mapsto \Pi$.*

Condition 1 in the above definition is the basic requirement for a model. Condition 2 allows us to partially represent an object with only those methods that are of interest in a given context. Condition 3 is a restriction to guarantee necessary inheritance of membership, whereas Condition 4 is needed for the purpose of subclass value inheritance.

Let Σ be a database and ϕ be a database proposition. If for every model M of Σ, $M \models \phi$, we also call that ϕ *is entailed* by Σ, denoted as $\Sigma \models \phi$.

3. DATABASES WITH AUTHORIZATIONS

3.1 SYNTAX OF L^A

The *vocabulary* of $\mathcal{L}^a$ includes the vocabulary of $\mathcal{L}$ together with the following additions:

1. A finite set of *subject variables* $\mathcal{SV} = \{s, s_1, s_2, \cdots\}$ and a finite set of *subject constants* $\mathcal{SC} = \{S, S_1, S_2, \cdots\}$. We denote $\mathcal{S} = \mathcal{SV} \cup \mathcal{SC}$.

2. A finite set of *access-rights variables* $\mathcal{AV} = \{r, r_1, r_2, \cdots\}$ and a finite set of *access-right constants* $\mathcal{AC} = \{R, R_1, R_2, \cdots\}$. We denote $\mathcal{A} = \mathcal{AV} \cup \mathcal{AC}$.

3. A ternary predicate symbol *holds* taking arguments subject, access-right, and object/method respectively.

4. Logic connectives $\wedge$ and $\neg$.

In language $\mathcal{L}^a$, a fact that a subject S has access right R for object O is represented using a ground atom $holds(S,R,O)$. A fact that S has access right R for object O's method $f(\cdots) \hookrightarrow \Pi$ is represented by ground atom $holds(S,R,O|f)$.

In general, we define an *access fact* to be an atomic formula $holds(s,r,o)$ (or $holds(s,r,o|f)$, where $o|f$ indicates a method associated with object o.) or its negation. A ground access fact is an access fact without any variable occurrence. We view $\neg\neg F$ as F. An *access fact expression* in $\mathcal{L}^a$ is defined as follows: (i) each access fact is an access fact expression; (ii) if ψ is an access fact expression and ϕ is an isa or object proposition, then $\psi \wedge \phi$ is an access fact expression; (iii) if ψ and ϕ are access fact expressions, then $\psi \wedge \phi$ is an access fact expression. A ground fact expression is a fact expression with no variable occurrence in it. An access fact expression is *pure* if it does not have an isa proposition occurrence in it.

Based on the above definition, the following are access fact expressions:

$$holds(S,R,O) \wedge O \textbf{ isa subclass of} C,$$
$$\neg holds(S,R,o) \wedge o \textbf{ isa member of} C$$

where o is an object variable.

Now we are ready to define propositions in language $\mathcal{L}^a$. Firstly, $\mathcal{L}^a$ has the same types of database propositions as $\mathcal{L}$, i.e. object proposition, isa proposition and constraint proposition. It also includes the following additional type of *access proposition*:

$$\psi \textbf{ implies } \phi \textbf{ with absence } \gamma, \tag{26}$$

where ψ is an access fact expression, and ϕ and γ are pure access fact expressions. Note that ψ, ϕ and γ may contain variables. In this case, as before, ((26)) will be treated as a set of access propositions obtained by replacing ψ, ϕ and γ with their ground instances respectively.

There is a special form of access proposition ((26)) when γ is empty. In this case, we rewrite ((26)) as

$$\psi \textbf{ provokes } \phi, \tag{27}$$

which is viewed as a *causal* or *conditional* relation between ψ and ϕ. For instance, we may have an access proposition like:

$$holds(s,r,c) \wedge o \textbf{ isa subclass of } c$$
$$\textbf{provokes } holds(s,r,o).$$

This access proposition expresses that for any subject s, access right r and objects o and c, if s has access right r on c and o is a subclass of c, then s also has access right r on o. This is also an example of access inheritance.

On the other hand, there is also a special form of ((26)) when ψ is empty. In this case, we rewrite ((26)) simply as

$$\textbf{always } \phi. \tag{28}$$

For example, we can express a fact that the database administrator (DBA) should have any access right on any object as follows:

$$\textbf{always } holds(DBA, r, o).$$

It is clear that our access propositions ((26)), ((27)) and ((28)) provide flexibility to express different types of authorization policies on objects. However, to ensure the proper inheritance of access policies on different objects, some specific types of access policies are particularly important for all databases. The set of these kinds of authorization policies is referred to as the *generic authorization scheme* for databases. Consider

$$\begin{aligned} holds(s, r, o) \ \textbf{implies} \ & holds(s, r, o|f) \\ \textbf{with absence} \ & \neg holds(s, r, o|f). \end{aligned} \tag{29}$$

Intuitively, ((29)) says that if s has access right r on object o, then s also has access right r on each of its methods under the assumption that $\neg holds(s, r, o|f)$ is not present.

We also have the following two generic access propositions:

$$\begin{aligned} holds(s, r, c) \wedge o \ \textbf{isa subclass of} \ & c \\ \textbf{implies} \ & holds(s, r, o) \\ \textbf{with absence} \ & \neg holds(s, r, o), \end{aligned} \tag{30}$$

and

$$\begin{aligned} holds(s, r, c|f) \wedge o \ \textbf{isa subclass of} \ & c \\ \textbf{implies} \ & holds(s, r, o|f) \\ \textbf{with absence} \ & \neg holds(s, r, o|f). \end{aligned} \tag{31}$$

((30)) and ((31)) guarantee the proper inheritance of access policies on subclasses.

Finally, the following two propositions ensure the membership inheritance of access policies.

$$\begin{aligned} holds(s, r, c) \wedge o \ \textbf{isa member of} \ & c \\ \textbf{implies} \ & holds(s, r, o) \\ \textbf{with absence} \ & \neg holds(s, r, o), \end{aligned} \tag{32}$$

and

$$holds(s,r,c|f) \wedge o \textbf{ isa member of } c \textbf{ implies } holds(s,r,o|f) \textbf{ with absence } \neg holds(s,r,o|f). \quad (33)$$

Now we can formally define our database with associated authorizations as follows. We will refer to this kind of database as *extended object-oriented database.*

Definition 27 *An* extended object-oriented database *in* $\mathcal{L}^a$ *is a pair* $\Lambda = (\Sigma, \Xi)$*, where* $\Sigma = (\Gamma, \Delta, \Omega)$ *is the database as defined in Definition 1, and* $\Xi = GA \cup A$ *is an* authorization description *on* Σ *where* GA *is a collection of generic authorization propositions ((29)) - ((33)), and* A *is a finite set of user-defined access propositions.*

3.2 SEMANTICS OF L^A

Now we consider the semantics of language $\mathcal{L}^a$. To define a proper semantics of our access proposition ((26)), we need to employ a *fix-point semantics* that shares the spirit of fix-point semantics used for *extended logic programs* [2, 5].

Formally, a *structure* I^Λ of $\mathcal{L}^a$ is a pair (I^Σ, I^Ξ), where I^Σ is a structure of $\mathcal{L}$ as defined in Definition 2 and I^Ξ is a finite set of ground literals with forms $holds(S,R,O)$, $holds(S,R,O|f)$, $\neg holds(S,R,O)$ or $\neg holds(S,R,O|f)$. Now we can define the entailment relation $\models_\lambda$ of $\mathcal{L}^a$.

Definition 28 *Let* $I^\Lambda = (I^\Sigma, I^\Xi)$ *be a structure of* $\mathcal{L}^a$*. We define the* entailment relation $\models_\lambda$ *of* $\mathcal{L}^a$ *as follows.*

1. *For a database proposition* ψ, $I^\lambda \models_\lambda \psi$ *iff* $I^\Sigma \models \psi$.
2. *For a pure ground access fact expression* $\psi \equiv F_1 \wedge \cdots \wedge F_k$*, where each* F_i *is a ground access fact,* $I^\Lambda \models_\lambda \psi$ *iff for each* i, $F_i \in I^\Xi$.
3. *For a ground access fact expression* ψ, $I^\Lambda \models_\lambda \psi$ *iff for each isa or object proposition* ϕ *occurring in* ψ, $I^\Sigma \models \phi$*, and for each ground access fact* ϕ' *occurring in* ψ, $\phi' \in I^\Xi$.
4. *For an access fact expression* ψ, $I^\Lambda \models_\lambda \psi$ *iff for each ground instance* ψ' *of* ψ, $I^\Lambda \models_\lambda \psi'$.

Now we are in the position to formally define a model of $\Lambda = (\Sigma, \Xi)$.

Definition 29 *Consider an extended database* $\Lambda = (\Sigma, \Xi)$ *and a structure* $I^\Lambda = (I^\Sigma, I^\Xi)$. *Let* Ξ' *be an authorization description obtained from* Ξ *in the following way:*

(i) by deleting each access proposition ψ **implies** ϕ **with absence** γ *from* Ξ *if for some* F_i *in* γ, $F_i \in I^{\Xi}$[4];

(ii) by translating all other access propositions ψ **implies** ϕ **with absence** γ *to the form* ψ **provokes** ϕ, *or to the form* **always** ϕ *if* ψ *is empty.*

Definition 30 *Consider an extended database* $\Lambda = (\Sigma, \Xi)$ *and a structure* $I^\Lambda = (I^\Sigma, I^\Xi)$. *Let* Ξ' *be an authorization description obtained from* Ξ *as described in Definition 5.* $I^\Lambda = (I^\Sigma, I^\Xi)$ *is a* model *of* $\Lambda = (\Sigma, \Xi)$ *if and only if*

(i) I^Σ *is a model of* Σ;

(ii) I^Ξ *is a smallest set satisfying the following conditions:*

(a) for each access proposition **always** ϕ *in* Ξ', $I^\Lambda \models \phi$;

(b) for each access proposition of the form ψ **provokes** ϕ *in* Ξ', *if* $I^\Lambda \models \psi$, *then* $I^\Lambda \models \phi$.

Due to space limit, we are unable to specify reasoning about authorizations in object-oriented databases and the theorems about these reasoning. Refer to our full paper [1] for reasoning about authorizations and a case study for describing authorization on data object using our formalization.

4. DISCUSSIONS AND FUTURE WORK

Here, we briefly review some related work, discuss the different approaches used in object-oriented database security, and outline our future work.

In [3], a security model for object-oriented databases was proposed. This model consists of a set of policies, a structure for authorization rules and an algorithm to evaluate access requests. The database is composed of objects that include a collection of facts and a collection of relevant rules. An object binds knowledge rules to database facts. The database is specified by the OSAM* [10, 11] model, in which the generic properties are defined through a *generalization association* and the set of attributes of a class is defined by an *aggregation association.* Derived classes(subclasses) are viewed as generic. Class

[4]Recall that $\gamma \equiv F_1 \wedge \cdots \wedge F_k$ is a pure access fact expression, i.e. each F_i $(1 \leq i \leq k)$ is a ground access fact.

inheritance properties suggest that access to some attributes of a class also implies access to the corresponding values in its subclass. Generally, there are three types of access policies:

1. A user who has access to a class is allowed to have similar type of access in the corresponding subclasses to the attributes inherited from that class.

2. Access to a complete class implies access to the attributes defined in that class as well as to attributes inherited from a higher class.

3. An attribute defined for a subclass is not accessible by accessing any of its superclass.

In our model, we have considered similar access policies via subclass and membership relationship authorization rules [1]. These rules are specified by the theorems of subclass authorization, membership authorization, overriding subclass authorization and overriding membership authorization. They can capture the above three types of access policies. Our model is based on our previous work of formal specification for authorization policies and their transformations, and is discussed from authorization specification point of view. However, in practice, the access policies are organizational dependent. They can vary from organization to organization and can also vary depending on the type of applications.

The access policies that we have considered in this paper are based on subject authorization viewpoint. In practice, both subject and object can be in a hierarchy structure. ¿From object-oriented system viewpoint, access propagation is also data structure related. In our model, authorization propagation from object viewpoint can also be specified.

In our future work, we intend to consider attributes for other types of associations. In particular, we will consider in more detail the generalization and aggregation associations. In addition, the placement of the authorization policies also needs to be addressed. They may be placed in a special class or a class they refer to, or propagated through the hierarchy structure. Furthermore, we have not investigated the access to the authorization system itself yet. This will also be considered in our future work.

References

[1] Bai, Y. and Varadharajan, V. (1998). A Logical Formalization for Specifying Authorizations in Object-Oriented Databases. Manuscript.

[2] Bai, Y. and Varadharajan, V. (1997). A language for specifying sequences of authorization transformations and its applications. *Proceedings of the*

1997 International Conference on Information and Communication Security. Lecture Notes in Computer Science, Springer-Verlag, **1334**, pp. 39–49.

[3] Fernandez, E.B., Gudes, E. and Song, H. (1989). A security model for object-oriented databases. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 110–115.

[4] E.B. Fernandez, R.B. France and D. Wei (1995). A formal specification of an authorization model for object-oriented databases. *Database Security, IX: Status and Prospects* (eds. Spooner *et al.*), Elsevier Science Publishers B. V., pp. 95–109.

[5] Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9**, pp. 365–385.

[6] Gudes, E., Song, H. and Fernandez, E.B. (1991). Evaluation of negative, predicate, and instance-based authorization in object-oriented databases. *Database Security, IV: Status and Prospects* (eds. S. Jajodia and C.E. Landwehr), Elsevier Science Publishers B. V., pp. 85–98.

[7] Kifer, M., Lausen, G. and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, **42(4)**, pp. 741–843.

[8] Lunt, T.F. (1990). Discretionary Security for Object-Oriented Database Systems. Technical Report 7543, Computer Science Laboratory, SRI International.

[9] Millen, J.K. and Lunt, T.F. (1992). Security for object-oriented database systems. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 260–272.

[10] Su, S.Y.W. and Raschid, L. (1985). Incorporating knowledge rules in a semantic data model: An approach to integrated knowledge management. *Proceedings of the AI Applications Conference.*

[11] Su, S.Y.W., Krishnamurthy, V. and Lam, H. (1998). An objected-oriented semantic association model (OSAM*), *AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications* (eds. S. Kumara, R. Kashyap and A.L. Soyster), ALLE.