データベースシステム　102－2
（１９９５．　３．　１４）

論理体系へのデータ隠蔽機構の導入に関する考察

久米 出

東京工業大学大学院 理工学研究科 情報科学専攻

オブジェクト指向データベース上の諸概念にはデータ隠蔽のメカニズムが付随していることが多い。データ抽象、ビュー、アクセス管理の実現にはそれぞれ異なったデータ隠蔽のメカニズムが必要とされる。データベースプログラミングを行う場合、結果としてこうしたアドホックに用意された隠蔽のメカニズムを利用することができる。しかしながらプログラマが既存の隠蔽メカニズムに対して不満を覚えた場合はどうすれば良いのか？隠蔽のメカニズムを記述するための単純かつ統一的なフレームワークを備えたデータベースプログラミング言語がその答である。本稿ではその実現への第一歩としてデータ隠蔽のメカニズムを定義可能な論理体系を考察する。この論理体系にはデータ隠蔽を表現するためのな述語が用意されている。ここではオブジェクトのIDと属性はタームとして扱われ、ビューへのアクセスもタームを用いて表現される。その結果としてこの論理体系はデータ隠蔽のメカニズムを記述するための強力な表現力が備わっている。

A Study of Introduction of the Mechanism of Data Hiding to a Logical System

Kume Izuru

Department of Information Science, Graduate School of Science and Engineering, Tokyo Institute of Technology

Many concepts of object oriented database are accompanied with the mechanisms for data hiding. Different mechanisms of data hiding are needed for data abstraction, view mechanism, and access control. In database programming, we can use such mechanisms of data hiding which are predefined ad hoc But what can we do if the programmer is not satisfied with the predefined mechanisms of data hiding? A database programming language with simple and unified framework to describe new mechanisms of data hiding, is the answer. In this article in order to take the first step to that aim we consider a logical system with which we can describe the mechanism of data hiding. This logical system has special predicate to represent data hiding. In this system object identities and attributes are represented as terms and views are accessed by terms. As a result this logical system has strong expressive power to describe the mechanism of data hiding.

1 Introduction

Data hiding plays important role in database systems. In the area of object oriented data base (OODB), various kinds of mechanisms of data hiding appear. Views need data hiding mechanisms to hide object identities and attributes. Data abstraction also need another mechanisms for data hiding to hide the attributes of an object and permit indirect access by method. Data hiding is also needed for security. In [BJS93] we can see some examples of access control. The concept of access control includes data hiding, because we can realize a range of access control with data hiding. From the viewpoint of DB programmer so far as security is concerned, we can use "data hiding" instead of "access control". Because actually there is no difference between "You know its existence but can't touch it." and "You don't know its existence." So we can regard the problem of access control as that of data hiding.

Turn to the area of Deductive Object Oriented Database (DOOD), though there are many researches to realize data abstraction [YTY92, Nak84, BM92, McC93, HL92], but no study emphasizes on data hiding as well as we know.

Probably it is because DOOD use Logic Programming Language (LPL) for Database Programming Language (DBPL). Originally the declarative programming style of LPL matches description of the data structure or data definition in a broad sense. Examples are, say, recursively defined relations, integrity constraints [Llo87, Ull88] or heterogeneous database [LSS93], which are hard to express in procedural languages. In usual the semantic domain of LPL is single space, but the concept of data hiding requires multiple space of semantic domain. It is because if we introduce data hiding for security, we assume multiple space where we can see not all data. Whether we can recognize the existence of an object or an attribute of the object depends on which space we live in.

In this article we consider the logical basis for Data Definition Language (DDL) for DOOD, to describe various kind of data hiding mechanisms. We call this logical system **View-Logic**. The semantic domain of View-Logic is enriched to design the data structure with high level concept of data hiding. To do this we decide what are basic concepts commonly seen between enormous numbers of data models of OODBs [ZM90]. We argue the concept of object identity and complex object should be basic in our model [MH90, KW89, KL89, KLW90, HY90, AK89]. That is object identities are domain for interpretation of data, and attributes are functions [KL89, KLW90] and has object identity as its value.

To reflect the situation above View-Logic has multiple world semantics so it belongs to modal logic. In addition we introduce two concepts ... **belonging** and **data hiding**. Belonging is generalization of the relationship between class and instance. We have two types of data hiding, hiding of object identity and hiding of attribute of an object. They are realized as the relationships between worlds and object identities.

This article is organized as follows: In the section 2 we give the syntax of View-Logic. There we introduce the special notation to represent object's attribute, belonging, data hiding, and specification of a set of objects. In section 3 we discuss about the semantic domain of View-Logic. In section 4 we give the procedure in the form of tableau and discuss its soundness. In section 5 we show future works.

2 Syntax of View-Logic

In this section we explain the syntax of View-Logic. View-Logic is basically multi modal logic. We can use object identifier as action unlike ordinary multi modal logic. We can use constant names and variables as object identifiers. In View-Logic views correspond to world in ordinary modal logic. A sequence of object identifiers represents the access to a view. We use the term "view access" instead of action. So a formula $[\alpha]F$ at some view Γ represents that F is true at every view accessible from Γ with view access α.

As mentioned above we use an object identity for α and also use such term $f(o_1, \cdots, o_n)$ where f is function symbol called access function and $o_1 \ldots o_n$ are object identifiers.

To access a view with α we must recognize all of the object identifiers which appears in α. So from the view where some of the object identities in α is hidden we can't access any world by α.

We introduce four types of new atomic formulae for definition of object's attribute, for hiding of object identity, hiding of attribute and for belonging relationships. In addition, one formula is introduced to define the specification of all objects (instances) which belongs to the same object(class).

When an object whose object identity is *obj* has attribute *att* and it's value is *val*, we express it by $obj[att \Rightarrow val]$. We borrow this notation from [KW89, KL89, KLW90]. Note the *obj* must be object identifier and *att* be constant name or variables and *val* is either object identifier or basic value expression such as integer.

For hiding object identity from all views accessible with α we use formula such as $o \downarrow \alpha$ where o is object identifier. To hide attribute *att* of object o from the views accessible with α we use such formula $(o, att) \downarrow \alpha$.

To express o belongs to c we write $o : c$. In this case both of o and c must be object identifier. This formula is usually paired with the formula introduced below.

We introduce a new binary logical connective **is**. It is used as c **is** F where c must be a object identifier and F is any formula. Only in such F we permit to use two special object identifier **self** and **inst** where the former is assigned c and the latter is assigned any object identity which belongs to c and not hidden. Formula F defines the specification of all objects which belongs to c and are not hidden.

```
emp1:pub_reg_emp ∧ emp2:pub_reg_emp ∧
emp1[salary⇒1000]∧emp1[status⇒"part"]∧
emp1[salary⇒1000]∧emp1[status⇒"reg"]∧
pub_reg_emp is ((inst,salary)↓selt∧
     (inst↓selt↔inst[status⇒"part"]))
```

Figure 1: an example of data hiding

```
pub_reg_emp:hidespec∧
∀V∀O(V:hidespec ∧ O:V ∧ O↓V →
  ∀U((¬U[owns⇒V]∨U[auth⇒"low"])
     ↔O↓for_usr(V,U)))
```

Figure 2: an example of views for users

We explain informally the meaning of the formulae by two examples. Formal definition of the semantics of formulae is given in section 3. The first example in figure 1 gives an example of hiding of object identities and hiding of an attribute. Assume there is a view Γ where the formula in figure 1 is true. So the formula *pub_reg_emp*is $\cdots$ is also true. From this it follows that there is a view Δ which is accessible from Γ by *pub_reg_emp*. Notice that *emp*1 and *emp*2 belong to *pub_reg_emp*. So the attribute *salary* of *emp*1 and *emp*2 is hidden at the view Δ by the formula $(\mathbf{inst}, salary) \downarrow \mathbf{selt}$. *emp*1 is hidden at Δ by the formula $\mathbf{inst} \downarrow \mathbf{selt} \leftrightarrow \mathbf{inst}[status \Rightarrow "part"]$.

The second example in figure 2 defines views based on another view. Assume there is a view Γ where the formulae in figure 1 and in figure 2 are true. Then we have one world Γ_u for each user u of the database accessible from Γ by $R_{for\_usr(pub\_reg\_emp,u)}$. The formula in figure 2 states that if a user u has low authorization or is not the owner of the object *pub_reg_emp*, then the object identity whose status is part time is hidden at the view Γ_u.

Note that we can use object identities as a way to access views from a view. So we have a way of access control of views by hiding object identities. This is an example that View-Logic has strong expressive power of data hiding mechanism.

3 Semantic Domain of View-Log

In this section we explain the semantic domain of View-Logic and interpretation of terms and formulae. We define the logical consequence of a set of formulae. At first we define frame. A frame $\mathcal{FR}$ contains the entities of semantic domain and is represented as a tuple $(\mathcal{O}, \mathcal{G}, \mathcal{S}, \mathcal{R})$ where $\mathcal{O}$ is a non-empty set of object identities, $\mathcal{S}$ is a set of signatures which are the names of attributes, $\mathcal{G}$ a non-empty set of views (called world in ordinary modal logic). $\mathcal{R}$ is a set of binary relations on $\mathcal{G}$. Each element of $\mathcal{R}$ corresponds to view access. So an element in $\mathcal{R}$ is written as R_α where α is a view access. Then we must define the domain of view access $\mathcal{VA}$. $\mathcal{VA}$ includes $\mathcal{O}$. As mentioned in section 2 we permit a expressions like $f(o_1, \cdots, o_n)$ for view access. So the domain for interpretation of access function f is needed. We prepare the domain $\mathcal{AF}$ Then we can define $\mathcal{VA}$ as $\mathcal{O} \cup (\mathcal{AF} \times \mathcal{O}) \cup \cdots \cup (\mathcal{AF} \times \mathcal{O}^n \cdots)$

Then we define schema. A schema $\mathcal{SCH}$ defines the structure of a semantic domain based on $\mathcal{FR}$. It is represented as a tuple $(\mathcal{FR}, \mathit{Resident}, \mathit{Belong}, \mathcal{ATT})$. Then we explain the contents of $\mathcal{SCH}$. *Resident* is a function from $\mathcal{G}$ to $2^{\mathcal{O}}$. It determines for each view Γ in $\mathcal{G}$ the set of object identities not hidden in Γ. So $o \in \mathit{Resident}(\Gamma)$ means that o is accessible in Γ. *Belong* is a binary relation on $\mathcal{O} \times \mathcal{O}$. It determines for each object which object to belong. $\mathit{Belong}(obj, mobj)$ means obj belongs to $mboj$ and the former determines the specification of the latter if there is a formula in the form of $mobj\,\mathbf{is}\,F$. We can make a directed graph by regarding object identities as nodes and relation as edges from first argument to second argument. $\mathcal{ATT}$ assigns a pair of signature and view, a function from a set $Objs$ which is subset of O to the following set $V \cup (V \rightarrow V) \cup \cdots \cup (V^n \rightarrow V) \cup \cdots$ where V is the union of $Objs$ and the set of basic value ($\mathcal{BV}$) such as integers or strings.

Before we begin to explain the interpretation of a formula we restrict the set of schemata to consider. We require them to satisfy the following conditions.

1. For any $\Gamma \in \mathcal{G}$ and $sig \in \mathcal{S}$ both of the domain and range of $\mathcal{ATT}(sig, \Gamma)$ are the subset of $\mathit{Resident}(\Gamma)$.
2. for any view Γ, any view access α and object identity o, if there is a view Δ_0 accessible from Γ by R_α such that o belongs to $\mathit{Resident}(\Delta_0)$ then o belongs to $\mathit{Resident}(\Delta)$.
3. for any view Γ, any view access α, any signature sig and object identity o it is satisfied that there is a view Δ_0 accessible from Γ by R_α such that o belongs to the domain of $\mathcal{ATT}(sig, \Delta_0)$ then o belongs to $\mathcal{ATT}(sig, \Delta)$ and $\mathcal{ATT}(sig, \Delta_0)(o) = \mathcal{ATT}(sig, \Delta_0)(o)$.
4. for any view Γ and any object identity $mobj$, it is satisfied that if $mobj$ doesn't belong to $\mathit{Rersident}(\Gamma)$ then there is no view accessible by R_α where $mobj$ appears in α.
5. The directed graph made from *Belong* must be a collection of finite directed trees with direction from children to only one parent.
6. for any object identity o, any views Γ and Δ, if Δ is accessible from Γ and o belongs to $\mathit{Resident}(\Delta)$ then o belongs to Γ.

Now we define model of our logic. A model $\mathcal{M}$ gives interpretation of term and formula. It consists of $\mathcal{SCH}$ and functions to map syntactical expression to entity in $\mathcal{SCH}$. It is represented as a tuple $(\mathcal{SCH}, \mathcal{F}_{obj}, \mathcal{F}_{label}, \mathcal{F}_{att})$. $\mathcal{F}_{oid}$ is one to one mapping of constant object identity to $\mathcal{O}$. $\mathcal{F}_{acc}$ maps access function symbol to $\mathcal{AF}$. $\mathcal{F}_{att}$ maps constant attribute symbol to $\mathcal{S}$. Notice that different names of object identity are interpreted to different elements in $\mathcal{O}$. We may use a single notation $\mathcal{F}$ for $\mathcal{F}_{obj}$,$\mathcal{F}_{label}$ and $\mathcal{F}_{att}$ when no confusion occurs.

We pose constraint on assignments. An assignment σ must assign an element $\mathcal{S}$ to the variable which occurs it the place of attribute of the formula $obj[att \Rightarrow val]$.

Given model $\mathcal{M}$ and assignment σ we describe the interpretation of term t as $\mathcal{I}[\![\mathcal{M}, \sigma]\!](t)$. and define it as follows.

If t is a variable, then $\mathcal{I}[\![\mathcal{M}, \sigma]\!](t)$ is $\sigma(t)$. If t is a name of object identity, then $\mathcal{I}[\![\mathcal{M}, \sigma]\!](t)$ is $\mathcal{F}_{oid}(t)$. If t is a name of attribute, then $\mathcal{I}[\![\mathcal{M}, \sigma]\!](t)$ is $\mathcal{F}_{att}(t)$. If $t = f(o_1, \ldots, o_n)$ is view access, then $\mathcal{I}[\![\mathcal{M}, \sigma]\!](t)$ is $\mathcal{F}_{acc}(\mathcal{F}_{oid}(o_1), \ldots, \mathcal{F}_{oid}(o_n))$. In cases $\mathcal{M}$, Γ and σ are obvious, we often omit them and write as $\mathcal{I}(t)$ for interpretation.

In view-logic at a view each formula is assigned one of three values. $\{\mathbf{true}, \mathbf{false}, \mathbf{invalid}\}$. We introduce the third value to express the notion of illegal access. In view-logic a subset of the set of objects is assigned to each view (world) by the function *Resitents*. It defines the set of object identities which are not hidden so in a view we can access not necessarily all objects. Think about a formula F including an expression of object identity which cannot appear at a view. It is something like an "illegal formula". What value should we assign to it at the view ? Should we give it false? If so, we must assign true to the formula $\neg F$. But $\neg F$ is also illegal formula. So we cannot do without having the third value.

Given model M, view Γ and assignment σ, we can describe the interpretation of a formula F as $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma]\!](F)$. In cases $\mathcal{M}$, Γ and σ are obvious, we often omit them and write as $\mathcal{I}(F)$ for interpretation. It is defined as follows.

In the case F has the form of *dobj* : *mobj*, If either $\mathcal{I}(dboj)$ or $\mathcal{I}(mobj)$ doesn't belong to *Resident*(Γ) then $\mathcal{I}(F)$ is **invalid**. Otherwise if *Belong*$(\mathcal{I}(dboj),\mathcal{I}(mobj))$ then $\mathcal{I}(F)$ is **true** else **false**. Notice that *Belong* is global relation. So in the case both *dobj* and *mobj* are constant symbol, if $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma]\!](F)$ is **true** (resp. **false**) at a view, then $\mathcal{I}[\![\mathcal{M},\Delta,\sigma]\!](F)$ is **true**(resp. **false**) or **invalid** at every view Δ.

Consider the case F has the form of *obj*[*attr* $\Rightarrow$ *val*]. $\mathcal{I}(F)$ is **invalid** if either of following is satisfied.

1. $\mathcal{I}(obj)$ doesn't belongs to the domain of function of $\mathcal{ATT}(\mathcal{I}(attr),\Gamma)$.
2. $\mathcal{I}(val)$ doesn't belong to the range of $\mathcal{ATT}(\mathcal{I}(attr),\Gamma)$ nor $\mathcal{BV}$

Otherwise
$\mathcal{I}(F)$ is **true** if $\mathcal{ATT}(\mathcal{I}(attr),\Gamma)(\mathcal{I}(obj))$ is $\mathcal{I}(val)$. Otherwise $\mathcal{I}(F)$ is **false**.

Consider the case F is *obj* $\downarrow$ *mobj*. If either $\mathcal{I}(obj)$ or $\mathcal{I}(mobj)$ doesn't belong to *Resident*(Γ) then $\mathcal{I}(F)$ is **invalid**. Else if $\mathcal{I}(obj)$ doesn't belongs to *Resident*(R) for all Δ accessible by $R_{\mathcal{I}(mobj)}$ then $\mathcal{I}(F)$ is **true**. $\mathcal{I}(F)$ is **false** otherwise.

Consider the case F is (obj, att) $\downarrow$ *mobj*. If either $\mathcal{I}(obj)$ or $\mathcal{I}(mobj)$ doesn't belong to *Resident*(Γ) then $\mathcal{I}(F)$ is **invalid**. Else if $\mathcal{I}(obj)$ doesn't belongs to the domain of $\mathcal{ATT}(\mathcal{I}(attr),\Delta)$ for all Δ accessible by $R_{\mathcal{I}(mobj)}$ then $\mathcal{I}(F)$ is **true**. $\mathcal{I}(F)$ is **false** otherwise.

Consider the case F is *mobj* **is** G. $\mathcal{I}(F)$ is **invalid if either of the following conditions** is satisfied.

1. $\mathcal{I}(mobj)$ doesn't belong to *Resident*(Γ).
2. for some $d \in \mathcal{O}$, *Belong*$(d,\mathcal{I}(mobj))$ but $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma[\text{inst}/d][\text{self}/\mathcal{I}(mobj)]]\!](G)$ is **invalid**

Otherwise $\mathcal{I}(F)$ is **true** if both of the following conditions are satisfied. $\mathcal{I}(F)$ is **false** otherwise.

1. $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma[\text{inst}/d][\text{self}/\mathcal{I}(mobj)]]\!](G)$ is **true**
2. there is accessibility relation $R_{\mathcal{I}(mobj)}$ from Γ

In the case F has either form of $A \rightarrow B$, $A \vee B$, $A \wedge B$ and $\neg A$, $\mathcal{I}(F)$ is **invalid** if either $\mathcal{I}(A)$ or $\mathcal{I}(B)$ is **invalid**. As usual otherwise.

In the case F has the form of $\forall x A$, $\mathcal{I}(F)$ is **invalid** if for some v which belongs to *Residents*$(\Gamma) \cup \mathcal{BV}$ and $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma[x/v]]\!](A)$ is **invalid**. $\mathcal{I}(F)$ is **true** if for all $v \in$ *Residents*$(\Gamma) \cup \mathcal{BV}$, $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma[x/v]]\!](A)$ is **true**. $\mathcal{I}(F)$ is **false** otherwise.

In the case F has the form of $\exists x A$, $\mathcal{I}(F)$ is **invalid**. if for any v which belongs to *Residents*$(\Gamma) \cup \mathcal{BV}$, $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma[x/v]]\!](A)$ is **invalid**. $\mathcal{I}(F)$ is **true** if for some $v \in$ *Residents*$(\Gamma) \cup \mathcal{BV}$, $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma[x/v]]\!](A)$ is **true**. $\mathcal{I}(F)$ is **false** otherwise.

In the case F is $[\alpha]G$. If $\mathcal{I}(G)$ is **invalid**. Else if $\mathcal{I}[\![\mathcal{G},\Delta,\sigma]\!](G)$ is **true** for any Δ accessible by R_α from Γ or there is no accessible relation R_α from Γ then $\mathcal{I}(F)$ is **true**. $\mathcal{I}(F)$ is **false** otherwise.

In the case F is $<\alpha> G$. $\mathcal{I}(F)$ is **invalid** if $\mathcal{I}(G)$ is **invalid** or there is a o which occurs in α and doesn't belong to $\mathcal{I}(o)$. Else if $\mathcal{I}[\![\mathcal{G},\Delta,\sigma]\!](G)$ is **true** for some Δ accessible by R_α then $\mathcal{I}(F)$ is **true**. $\mathcal{I}(F)$ is **false** otherwise.

Notice that $[\alpha]F$ is not equivalent with $\neg <\alpha> F$. If the latter is **true** then the former is **true** . But the converse is not always true. Even if the former is **true** the latter may be **invalid**.

We define the logical consequence in the way of [Fit93]. Assume we are given a set of closed formulae P (as global assumption) and S (as local assumption) and a formula G. Assume for any model $\mathcal{M}$ such that $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma]\!](F)$ is **true** for any $F \in P$, any $\Gamma \in \mathcal{G}$ and any assignment σ, we have that $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma]\!](F)$ is **true** for any Γ where for any $F' \in S$, $\mathcal{I}[\![\mathcal{M},\Gamma,\sigma]\!](F')$ is **true**. In such case we say that P is logical consequence of global assumptions P and local assumptions of S and write as $P \models S \Rightarrow P$.

In the next section we give the procedure where given global assumption P and local assumption S, only deduce logical consequences of P and S. It means we have sound proof system.

4 Tableau for View-Logic

In this section we will show the proof procedure for view logic in the form of tableau. We borrow the notation from [Fit93].

Assume we are given sets of formulas P as axioms and S as local conditions for world. If we can make a finite tableau tree satisfying following conditions,

1. It begins with a prefixed formula $(\_, 1)F$.
2. It is extended by the rules described later.
3. Each branch in it ends with $\Box$.

we call this tableau a proof of a formula F.

We can deduce $\Box$ in a branch when we have both A and $\neg A$. where A is atomic formula.It is formalized as follows.

$$\frac{\begin{array}{c}\sigma\ A\\ \sigma\ \neg A\end{array}}{\Box}$$

for any atomic formula A

Figure 3: $\Box$

We omit tableau rules for usual logical connectives. They are same as described in [Fit93]. According to the notation for modal operator, prefixes are not simply sequences of numerals but of pairs of view access and numerals. In this article we adopt the same definition in [Fit93] for available and unrestricted prefix.Figure 4 shows the tableau rules for modal operator.

$$\frac{\sigma\ [\alpha]X}{\sigma(\alpha, n)\ X}$$

for $\sigma(\alpha, n)$ available

$$\frac{\sigma\ \neg[\alpha]X}{\sigma(\alpha, n)\ \neg X}$$

for $\sigma(\alpha, n)$ unrestricted

$$\frac{\sigma\ <\alpha> X}{\sigma(\alpha, n)\ X}$$

for $\sigma(\alpha, n)$ unrestricted

$$\frac{\sigma\ \neg <\alpha> X}{\sigma(\alpha, n)\ \neg X}$$

for $\sigma(\alpha, n)$ available

Figure 4: modal operator

Then we describe the rules for quantifiers in figure 5.

Notice that attribute is interpreted as function. This reflects the fact that each object has only one value for each attribute.

$$\frac{\sigma\ \forall x A(X)}{\sigma\ A(t)}$$

for any closed term t

$$\frac{\sigma\ \neg\forall x A(X)}{\sigma\ \neg A(c)}$$

for any parameter c

Figure 5: quantifiers

From this we can deduce contradiction and identification. The rule in figure 6 reflects such deduction. Finally we show two rules.

$$\frac{\begin{array}{c}\sigma\ \mathrm{o[att{\Rightarrow}t]}\\ \sigma\ \mathrm{o[att{\Rightarrow}c]}\\ \sigma\ P(c)\end{array}}{\sigma\ P(t)}$$

for any parameter c

$$\frac{\begin{array}{c}\sigma\ \mathrm{o[att{\Rightarrow}v]}\\ \sigma\ \mathrm{o[att{\Rightarrow}v']}\end{array}}{\Box}$$

for any v and v' which represent different value in basic value or object identity

Figure 6: tableau rules for object

They reflect the concept of information hiding which is the most characteristics of ViewLogic. Recall that we introduced the third value **invalid** for the formula which contains some expression of illegal access. We hope to have a guarantee that we deal with only valid formula. So we introduce a new notation to indicate which object identity is valid for each world. We use such a notation as σ (o) where o is a constant name of object identity. This notation means object identity represented by o is belongs to the **residents** of the world represented by σ.

Before describe the rules for valid object identity, we define the set of appearance of constant object identities of a formula F which are not hidden. We define two sets $val_{pos}(F)$ and $val_{neg}(F)$ that are defined mutually recursively as follows. If we write $val\_(F)$ it designates both of $val_{pos}(F)$ and $val_{neg}(F)$.

For atomic formula A $val\_(A)$ is the set of all constant symbol which represent object identity and appears in A. And $val\_(o\mathrm{is}F) = val(o) \cup val\_(F)$.

For any F whose form is either $X \vee Y$, $X \wedge Y$ or $X \rightarrow Y$ then $val\_(F)$ is $val\_(X) \cup val\_(Y)$. $val_{neg}(\neg X)$ is $val_{pos}(X)$ and $val_{pos}(negX)$ is $val_{neg}(X)$.

$val\_(\forall x F)$ is $val\_(F)$. $val\_(\exists x F)$ is $val\_(F)$.

We must pay attention on the treatment of modal operator. $val_{pos}([\alpha]X)$ is $val_{pos}(X)$. $val_{neg}([\alpha]X)$ is $val_{neg}(X) \cup val(\alpha)$. $val_{pos}(< \alpha > X)$ is $val_{pos}(X) \cup val(\alpha)$. $val_{neg}(< \alpha > X)$ is $val_{neg}(X)$.

In the definition above $val(t)$ designates the set of constant names for object identifiers which appear in the term t.

$$\frac{}{\sigma(\alpha,n)\ (o)} \quad \begin{array}{l} \text{for any } \sigma(\alpha,n) \text{ available} \\ \text{such that } \dfrac{\begin{array}{c}\sigma\ (o) \\ \sigma\ o \downarrow \alpha\end{array}}{\begin{array}{c}\vdots \\ \Box\end{array}} \end{array}$$

$$\frac{\sigma(\alpha,n)\ (o)}{\sigma\ (o)}$$

$$\frac{}{\begin{array}{l}\sigma\ F \\ \sigma\ (o_1) \\ \vdots \\ \sigma\ (o_n)\end{array}} \quad \begin{array}{l}\text{for any } \sigma \text{ available,} \\ \text{for any } F \in P \text{ and} \\ \text{for any } \{o_1 \ldots o_n\} \subseteq val_{pos}(F)\end{array}$$

$$\frac{}{\begin{array}{l}(\_,1)\ F \\ (\_,1)\ (o_1) \\ \vdots \\ (\_,1)\ (o_n)\end{array}} \quad \begin{array}{l}\text{for any } \sigma \text{ available,} \\ \text{for any } F \in S \text{ and} \\ \text{for any } \{o_1 \ldots o_n\} \subseteq val_{pos}(F)\end{array}$$

Figure 7: tableau rules for validity

theorem 1 *Assume we are given sets of formulas P as axioms and S as local conditions for world and we get a proof of $\neg F$. Assume also for any constant name of object identity which occurs (σ G) in the tableau, we have (σ (o)) in the same branch. Then $P \models S \Rightarrow P$ in the meaning of [Fit93].*

$$\frac{}{\sigma(\alpha,n)\ (o)} \quad \begin{array}{l} \text{for any } \sigma(\alpha,n) \text{ available} \\ \text{such that } \dfrac{\begin{array}{c}\sigma\ (o) \\ \sigma\ o \downarrow \alpha\end{array}}{\begin{array}{c}\vdots \\ \Box\end{array}} \end{array}$$

Figure 8: negation as failure

To prove above theorem, we must show at first that all formulae in the tableau are valid (that means all formula is **true** or **false**.) and the tableau rules preserve the soundness. The most difficulty lies in the step in figure 8. We use the technique of double induction which is similar in the proof of the soundness of SLDNF-resolution in [Llo87]. We call a tableau which don't use this step a tableau of rank 0. We define tableau of rank k+1 it use the deduction steps other than figure 8 or use tableau of rank k. In each step in the proof of induction about the rank, we use the induction of the length of tableau rules. Now we define the length of a tableau rule is the most numbers of steps of all branches.

5 Conclusion and Future Work

In this article we proposed a logical system View-Logic that has strong expressive power to describe the mechanism of data hiding. It is because we have a simple predicate for data hiding and use object identities as a way to access a view. Now we are considering to make use of View-Logic in the area of security.

From the theoretical point of view, two big problems remain as future work. The first one is the completeness of proof system. Another is the change of the state of an object. This problem is itself difficult in the area of LPL. As for this problem we are at the stage of considering the syntax to express a change of state. We hope not to extend the semantic domain any more. As for other future work, we are considering about the treatment of general predicates and set value. They are needed for the practical reason but it remains a problem how we relate them to the data hiding mechanism.

Acknowledgment

The author is much grateful to professor Shibayama Etsuya for helpful comments and thanks to all people who joined the discussions.

References

[AK89] Serge Abiteboul and Paris C. Kanellakis. Object identity as a query language primitive. In *Proceedings of ACM SIGMOD*, pages 159–173, 1989.

[BJS93] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. Access controls in OO database systems - some approaches and issues. In *Advanced Database Systems*, LNCS, 1993.

[BM92] Elisa Bertino and Danilo Montesi. Towards a logical-object oriented programming languages for databases. In *LNCS 580*, 1992.

[Fit93] Melvin Fitting. Basic modal logic. In Dov M. Gabbay, C.J.Hogger, and J.A.Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, pages 368–440. Oxford University Press, 1993.

[HL92] P. M. Hill and J. W. Lloyd. The Gödel Programming Language. Technical report, Department of Computer Science University of Bristol University Walk, 1992.

[HY90] Richard Hull and Masatoshi Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers (extended abstract). In *Very Large Data Bases*, pages 455–468, 1990.

[KL89] Michael Kifer and Georg Lausen. F-Logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *Proceedings of ACM SIGMOD*, pages 134–146, 1989.

[KLW90] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. Technical report, Department of Computer Science State University of New York at Stony Brook, 1990.

[KW89] Michael Kifer and James Wu. A logic for object-oriented logic programming (Maier's O-Logic revisited). In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 379–393, 1989.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second, extended edition edition, 1987.

[LSS93] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Deductive and Object-Oriented Databases*, volume 760, 1993.

[McC93] F. G. McCabe. An introduction to L&O. In K.R. Apt, J. W. de Bakker, and J. J. M. M. Rutten, editors, *Logic Programming Languages Constraints, Functions, and Objects*. The MIT Press, 1993.

[MH90] Yukihiro MORITA and Hiromi HANIUDA. Object Identity in Quixote. In *Proc. of SIGDBS and SIGAI of IPSJ*, pages 109–118, October 1990.

[Nak84] Hideyuki Nakashima. Knowledge representation in Prolog/KR. In *International Symposium on Logic Programming*, pages 126–130, 1984.

[Ull88] Jeffrey D. Ullman. *Principles of database and knowledge-base systems*. Computer Science Press, 1988.

[YTY92] Hideki Yasukawa, Hiroshi Tsuda, and Kazumasa Yokota. Objects, Properties, and Modules in Quixote. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 257–258, 1992.

[ZM90] Stanley B. Zdonik and David Maier. *Readings in object-oriented database systems*, chapter 3. Morgan Kaufmann, 1990.