# A Formal Approach to Detecting Security Flaws in Object-Oriented Database Schemas

Toshiyuki MORITA Yasunori ISHIHARA Minoru ITO

{toshi-m, ishihara, ito}@is.aist-nara.ac.jp

Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara, 630-0101 Japan

**Abstract** Detecting security flaws is important in order to keep the database secure. A security flaw in object-oriented databases means that a user can infer the result of an unpermitted method only from permitted methods. Although a database management system enforces access control by an authorization, security flaws can occur under the authorization. The main aim of this paper is to discuss the detection problem of security flaws for database schemas. This problem is to decide whether or not, when a schema $S$, an authorization $A$, and a term $\tau$ to be verified are given, there exists a database instance of $S$ such that a security flaw on $\tau$ exists with respect to $A$. This paper shows that the problem is undecidable and proposes a decidable sufficient condition for a given schema to have no security flaw. Furthermore, this paper shows that the sufficient condition is also a necessary one if the given schema is monadic (i.e., every method is unary), and evaluates the time complexity to decide the condition.

**Keywords:** object-oriented database, authorization, security flaw, term rewriting system

## 1 Introduction

In recent years, various authorization models for object-oriented databases (OODBs) have been proposed and studied (e.g., [4],[5],[7]). Among of them, the method-based authorization model [7],[15] is one of the most elegant models since it is in harmony with the concept that "an object can be accessed only via its methods" in the object-oriented paradigm. In the model, the authorization $A$ for a user $u$ is represented as a set of $(m, (c_1, c_2, \ldots, c_n))$, which means that $u$ can invoke method $m$ on any tuple $(o_1, o_2, \ldots, o_n)$ of objects such that $o_i$ is an object of class $c_i$ for each $i$ $(1 \leq i \leq n)$.

This paper adopts the method-based authorization model and assumes the following database management policies. Let $(m, (c_1, c_2, \ldots, c_n))$ be in the authorization for a user $u$.

1. When $u$ invokes $m(o_1, o_2, \ldots, o_n)$, where $o_i$ is an object of $c_i$ $(1 \leq i \leq n)$, and the method execution successfully terminates, the object identifier of the resultant object is open (i.e., unclassified) to $u$.

2. If $m$ is a primitive method (i.e., $m$ is a base method; see Sect. 2.1), then the type declaration of $m$ at $(c_1, c_2, \ldots, c_n)$ is open to $u$. If $m$ is not primitive (i.e., $m$ is a user method; see Sect. 2.1), then its external specification is open to $u$.

Detecting security flaws is important in order to keep the database secure. Intuitively, a security flaw means that $u$ can infer the result of an unpermitted method only from permitted methods under the authorization for $u$. Although a database management system enforces access control by an authorization, security flaws can occur under the authorization. The following example shows a security flaw.

**Example 1:** Let $m$, $m_1$, $m_2$ be unary methods, $c$ a class, and $o$ an object of class $c$. Suppose that the authorization $A$ for a user $u$ is $\{(m_1, c), (m_2, c)\}$, and the implementation bodies of $m_1$ and $m_2$, which $u$ knows by Policy 2, are $m_1(x) = m(m(m(x)))$ and $m_2(x) = m(m(m(m(m(x)))))$, respectively. Also, suppose that $m_1(o) = o$ and $m_2(o) = o$, which $u$ knows by Policy 1. Then, $u$ can infer that $m(o) = o$ although $m$ is not contained in $A$. □

More formally, for a given database schema $S$, an authorization $A$ for a user $u$ under $S$, and a term $\tau$ to be verified (to be kept secret from $u$), the detection problem of security flaws for database schemas is to decide whether or not there exists a database instance $I$ of $S$ such that $u$ can infer the value of $\tau$ under $S$, $I$, and $A$.

The main aim of this paper is to discuss the above-mentioned detection problem. We adopt method schemas proposed by [1],[2] as a formal model of OODB schemas since they support such basic features of OODBs as method overloading, dynamic binding, and complex objects. The semantics is simply defined based on term rewriting. In this formalization, an important point is that the above detection problem is also defined based on term rewriting. First, we show that the problem is undecidable and propose a decidable sufficient condition for a given method schema to have no security flaw. Next, we show that the sufficient condition is also a necessary one if the given schema is monadic (i.e., every method is unary). Lastly, we propose an algorithm to decide the sufficient condition. We then evaluate the time complexity of the algorithm, and show that, for a monadic method schema, the algorithm decides in polynomial time of the size of the schema whether a security flaw exists or not.

Various models of security flaws have been discussed (e.g., [3],[8]–[10],[16],[17]). Generally, user's attack is modeled by precise inference or imprecise inference. Precise inference means that a user can infer only the exact value of the result of an unpermitted method. Example 1 illustrates precise inference. Ref. [16] discusses precise inference for OODBs. On the other hand, imprecise inference means that a user can infer several candidates of the result of an unpermitted method. Ref. [9] discusses imprecise inference for relational databases, and [10] does for OODBs. This paper focuses on precise inference for OODBs.

In [16], security flaws are classified into inferability and controllability. Roughly speaking, inferability means that a user can infer the returned value of a method invocation, and controllability means that a user can control (alter arbitrarily) the attribute-value of an object in a database instance. Since our query language does not support update operations for database instances, only inferability is considered as security flaws in this paper. However, since our query language supports recursion while the one in [16] does not, detecting inferability in our formalization is not obvious. Ref. [16] also proposes, for a given database schema $S$ and an authorization $A$, a sound algorithm for deciding whether security flaws exist under a database instance $I$ of $S$. However, [16] does not evaluate the complexity of the algorithm.

# 2 Method Schemas

## 2.1 Syntax

We introduce some notations before defining the syntax of method schemas. Let $F$ be a family of disjoint finite sets $F_0, F_1, F_2, \ldots$, where $F_n$ $(n = 0, 1, 2, \ldots)$ is a set of function symbols of arity $n$. For a countable set $X$ of variables, let $T_F(X)$ denote the set of all terms freely generated by $F$ and $X$. For a term $t \in T_F(X)$ and variables $x_i$ $(1 \leq i \leq n)$ in $X$, let $t[t_1/x_1, t_2/x_2, \ldots, t_n/x_n]$ denote the term obtained by replacing every $x_i$ in $t$ with a term $t_i$ $(1 \leq i \leq n)$. For example, $f(x_1, g(x_1, x_2))[a/x_1, b/x_2] = f(a, g(a, b))$. For a term $t$, we define the set of *occurrences* $OC(t)$ as the smallest set of sequences of positive integers which satisfies the following (1) and (2):

(1) $\varepsilon \in OC(t)$, where $\varepsilon$ is the empty sequence.

(2) If $r \in OC(t_i)$, then $i \cdot r \in OC(f(t_1, t_2, \ldots, t_n))$ $(1 \leq i \leq n)$. (The center dot ('$\cdot$') represents the concatenation of sequences.)

The replacement in $t$ of $t'$ at $r$, denoted $t[r \leftarrow t']$, is defined as follows:

- $t[\varepsilon \leftarrow t'] = t'$;
- $f(t_1, t_2, \ldots, t_n)[i \cdot r \leftarrow t'] = f(t_1, \ldots, t_i[r \leftarrow t'], \ldots, t_n)$ $(1 \leq i \leq n)$.

For example, $f(f(x,g(x)),g(x))[1 \cdot 2 \leftarrow a] = f(f(x,a),g(x))$ and $f(f(x,g(x)),g(x))[2 \leftarrow a] = f(f(x,g(x)),a)$.

Now we go on to the definition of method schemas. Let $C$ be a finite set of *class names* (or simply classes) and $M$ a family of disjoint finite sets $M_0, M_1, M_2, \ldots$, where $M_n$ $(n = 0, 1, 2, \ldots)$ is a set of *method names* of arity $n$. Each $M_n$ is partitioned into $M_{b,n}$ and $M_{u,n}$: Each $m_b \in M_b$ $(= \bigcup_{n \geq 0} M_{b,n})$ (resp. $m_u \in M_u$ $(= \bigcup_{n \geq 0} M_{u,n})$) is called a *base method name* (resp. *user method name*). Furthermore, each $m \in M$ $(= M_b \cup M_u)$ is simply called a *method name*. We say that $M$ is a *method signature*.

**Definition 1:** A *base method definition of* $m_b$ *at* $(c_1, c_2, \ldots, c_n)$ is an expression

$$(m_b, (c_1, c_2, \ldots, c_n \rightarrow c)),$$

where $m_b \in M_{b,n}$, and $c$, $c_1, \ldots, c_n \in C$. □

Let $o_i$ be an object of class $c_i$ $(1 \leq i \leq n)$ (see Defs. 3 and 5 for formal definitions). Informally, the above base method definition declares that the application of $m_b$ to $o_1$, $o_2, \ldots$, $o_n$ results in an object of $c$ or its subclass.

**Definition 2:** A *user method definition of* $m_u$ *at* $(c_1, c_2, \ldots, c_n)$ is an expression

$$(m_u, (c_1, c_2, \ldots, c_n), t),$$

where $m_u \in M_{u,n}$, $c_1$, $c_2, \ldots$, $c_n \in C$, and $t \in T_M(\{x_1, x_2, \ldots, x_n\})$. □

Let $o_i$ be an object of $c_i$ $(1 \leq i \leq n)$. The above user method definition states that the application of $m_u$ to $o_1$, $o_2, \ldots$, $o_n$ results in term rewriting starting from $t[o_1/x_1, o_2/x_2, \ldots, o_n/x_n]$. The formal definition is presented in Sect. 2.3.

**Definition 3:** A *method schema*, which is originally introduced by Abiteboul et al. [1],[2], is a 5-tuple $(C, \leq, M, \Sigma_b, \Sigma_u)$ as follows:

1. $C$ is a finite set of class names.
2. $\leq$ is a partial order representing a class hierarchy. When $c' \leq c$, we say that $c'$ is a subclass of $c$ and $c$ is a superclass of $c'$.
3. $M$ is a method signature.
4. $\Sigma_b$ is a set of base method definitions.
5. $\Sigma_u$ is a set of user method definitions.

- $C = \{manager, employee, person, secretary\}$
- $manager \leq employee \leq person,\ \ secretary \leq employee$
- $M = \{\text{boss}, \text{partner}, \text{clerk}, \text{leader}, \text{leader\_of\_customer}\}$
- $\Sigma_b = \{(\text{boss}, (employee \rightarrow manager)),$
  $(\text{partner}, (person \rightarrow employee)),$
  $(\text{clerk}, (manager \rightarrow secretary)),$
  $(\text{leader}, (employee, employee \rightarrow manager))\}$
- $\Sigma_u = \{(\text{leader\_of\_customer}, (person), \text{leader}(\text{boss}(\text{partner}(x)), \text{partner}(x)))\}$

Fig. 1: Method schema $S_1$.

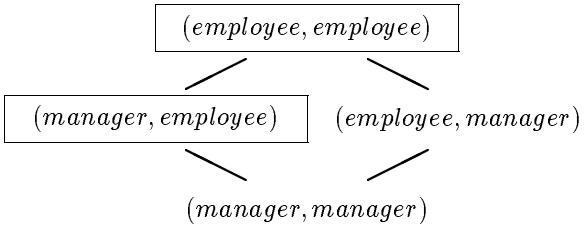


Fig. 2: Class hierarchy and method definitions.

For each possible combination $c_1, c_2, \ldots, c_n \in C$ and $m \in M_n$, there must exist at most one method definition of $m$ at $(c_1, c_2, \ldots, c_n)$. A method schema with only unary methods is called *monadic*. □

**Example 2:** Consider a method schema $S_1$ which represents relationships on personnel and customers. Fig. 1 shows the method schema. For example, a user method leader_of_customer is supposed to return the leader of the group to which the partner of a customer $x$ and his/her boss belong. □

## 2.2 Method Inheritance

Method definitions are inherited along the class hierarchy. For example, suppose that method definitions of $m$ are given at $(manager, employee)$ and $(employee, employee)$ (see Fig. 2), where class *manager* is a subclass of *employee*. Intuitively, the inherited method definition of $m$ at $(manager, manager)$ should be the definition at $(manager, employee)$, not the one at $(employee, employee)$, since $(manager, employee)$ is smaller than $(employee, employee)$ with respect to $\leq$. In the following, we formally define the inheritance of a method definition.

**Definition 4:** Let $S = (C, \leq, M, \Sigma_b, \Sigma_u)$ be a method schema, $m_b \in M_{b,n}$, and $c_1, c_2, \ldots, c_n \in C$. Suppose that $(m_b, (c'_1, c'_2, \ldots, c'_n \rightarrow c')) \in \Sigma_b$ is the base method definition of $m_b$ at the "componentwise smallest" $(c'_1, c'_2, \ldots, c'_n)$ above $(c_1, c_2, \ldots, c_n)$ in the sense that whenever $(m_b, (c''_1, c''_2, \ldots, c''_n \rightarrow c'')) \in \Sigma_b$ and $c_i \leq c''_i$ $(1 \leq i \leq n)$, it is the case that $c'_i \leq c''_i$ $(1 \leq i \leq n)$. We write $Res(m_b, (c_1, c_2, \ldots, c_n)) = (c'_1, c'_2, \ldots, c'_n \rightarrow c')$, which is called *the resolution of* $m_b$ *at* $(c_1, c_2, \ldots, c_n)$. We often write $Res(m_b, (c_1, c_2, \ldots, c_n)) = c'$ when $(c'_1, c'_2, \ldots, c'_n)$ is irrelevant. If such a unique base method definition does not exist, then the resolution of $m_b$ at $(c_1, c_2, \ldots, c_n)$ is *undefined*, and we write $Res(m_b, (c_1, c_2, \ldots, c_n)) = \perp$.

Similarly, for $m_u \in M_{u,n}$ and $c_1, c_2, \ldots, c_n \in C$, if $(m_u, (c'_1, c'_2, \ldots, c'_n), t)$ is the user method definition of $m_u$ at the "componentwise smallest" $(c'_1, c'_2, \ldots, c'_n)$ above $(c_1, c_2, \ldots, c_n)$, then we define *the resolution of* $m_u$ *at* $(c_1, c_2, \ldots, c_n)$ as $Res(m_u, (c_1, c_2, \ldots, c_n)) = ((c'_1, c'_2, \ldots, c'_n), t)$, or simply, $t$. If such $(m_u, (c'_1, c'_2, \ldots, c'_n), t)$ does not exist, then $Res(m_u, (c_1, c_2, \ldots, c_n)) = \perp$. □

In Fig. 2, $Res(m, (manager, manager))$ is the method definition of $m$ at $(manager, employee)$. However, if the method definition of $m$ also exists at $(employee, manager)$, then $Res(m, (manager, manager)) = \perp$, since $(manager, employee)$ and $(employee, manager)$ are incomparable with respect to $\leq$.

## 2.3 Semantics

The semantics of a method schema, which is introduced by Abiteboul et al. [1],[2], is defined as follows. To each class name, a set of objects is assigned. Also, to each base method name $m_b$, a mapping over appropriate sets of objects is assigned as its interpretation. The semantics of a user method is defined by the interpretation of base methods and term rewriting [6]. For a set $V$, let $V^m$ denote the Cartesian product $\underbrace{V \times V \times \cdots \times V}_{m}$.

**Definition 5:** An *interpretation*, also called a *database instance*, of a method schema $S = (C, \leq, M, \Sigma_b, \Sigma_u)$ is a pair $I = (\nu, \mu)$ as follows:

1. To each $c \in C$, $\nu$ assigns a finite disjoint set, denoted $\nu(c)$. Each $o \in \nu(c)$ is called an *object* of $c$. Let $O_{S,I} = \bigcup_{c \in C} \nu(c)$.

2. For each $m_b \in M_{b,n}$, $\mu(m_b)$ is a partial mapping from $O^n_{S,I}$ to $O_{S,I}$ which satisfies the following 2.1 and 2.2. Let $c_1, c_2, \ldots, c_n, c, c' \in C$.

   2.1 If $Res(m_b, (c_1, c_2, \ldots, c_n)) = c'$, then $\mu(m_b) \mid_{\nu(c_1) \times \nu(c_2) \times \cdots \times \nu(c_n)}$ is a total mapping to $\bigcup_{c \leq c'} \nu(c)$, where '$\mid$' denotes that the domain of $\mu$ is restricted to $\nu(c_1) \times \nu(c_2) \times \cdots \times \nu(c_n)$. That is, if $o_i$ belongs to $\nu(c_i)$ $(1 \leq i \leq n)$, then $\mu(m_b)(o_1, o_2, \ldots, o_n)$ is defined and must belong to $\nu(c)$ for some $c \leq c'$.

2.2 If $Res(m_b, (c_1, c_2, \ldots, c_n)) = \bot$, then $\mu(m_b)$ is undefined everywhere in $\nu(c_1) \times \nu(c_2) \times \cdots \times \nu(c_n)$. □

**Example 3:** Let $I_1 = (\nu, \mu)$ be an interpretation of $S_1$ in Example 2, where

$\nu(\mathit{manager}) = \{\texttt{Green}, \texttt{Black}\}$,
$\nu(\mathit{employee}) = \{\texttt{Silver}\}$,
$\nu(\mathit{person}) = \{\texttt{White}\}$,
$\nu(\mathit{secretary}) = \{\texttt{Brown}\}$,

and

$\mu(\text{partner})(\texttt{White}) = \texttt{Silver}$,
$\mu(\text{boss})(\texttt{Silver}) = \texttt{Green}$,
$\mu(\text{leader})(\texttt{Green}, \texttt{Silver}) = \texttt{Black}$,
$\mu(\text{clerk})(\texttt{Green}) = \texttt{Brown}$,
$\mu(\text{clerk})(\texttt{Black}) = \texttt{Brown}$.

For example, `Green` is an object of *manager*, `Brown` is an object of *secretary*, and the application of clerk to `Green` results in `Brown`. □

In what follows, we often write a tuple of classes and objects as $\vec{c}$ and $\vec{o}$, respectively. When we write $\vec{v}$, we implicitly assume that the $i$-th component of $\vec{v}$ is $v_i$, i.e., $\vec{v} = (v_1, v_2, \ldots, v_n)$. We also write $\nu(c_1) \times \nu(c_2) \times \cdots \times \nu(c_n)$ as $\nu(\vec{c})$ and $t[o_1/x_1, o_2/x_2 \ldots, o_n/x_n]$ as $t[\vec{o}/\vec{x}]$.

A term $t \in T_M(O_{S,I})$ is called an *instantiated term*. That is, an instantiated term consists of method names in $M$ and objects in $O_{S,I}$.

The *reduction relation* $\underset{S,I}{\rightarrow}$ on the instantiated terms, based on the leftmost innermost reduction strategy, is defined as follows.

**Definition 6:** For an instantiated term $t \in T_M(O_{S,I})$, let $r$ be the leftmost innermost occurrence such that the subterm of $t$ at $r$ is $m(\vec{o})$ for some $m \in M$ and $\vec{o} \in \nu(\vec{c})$.

1. If $m \in M_b$ and $Res(m, \vec{c}) \neq \bot$, then $t \underset{S,I}{\rightarrow} t[r \leftarrow \mu(m)(\vec{o})]$.
2. If $m \in M_u$ and $Res(m, \vec{c}) = t'$, then $t \underset{S,I}{\rightarrow} t[r \leftarrow t'[\vec{o}/\vec{x}]]$. □

Note that, by Def. 6, for any instantiated term $t$, there exists at most one term $t'$ such that $t \underset{S,I}{\rightarrow} t'$.

Let $\underset{S,I}{\overset{*}{\rightarrow}}$ be the reflexive and transitive closure of $\underset{S,I}{\rightarrow}$. If $t \underset{S,I}{\overset{*}{\rightarrow}} t'$ and there exists no $t''$ such that $t' \underset{S,I}{\rightarrow} t''$, then $t'$ is called the *normal form* of $t$, and we write $t \downarrow = t'$. If $t \downarrow \in O_{S,I}$, then the execution of $t$ is *successful*, and if $t \downarrow \notin O_{S,I}$ because of nonexistence of the resolution, then the execution of $t$ is *aborted*. In both cases (i.e., if $t \downarrow$ exists), the execution of $t$ is *terminating*. On the other hand, if $t \downarrow$ does not exist, then the execution of $t$ is *nonterminating*. We simply write $\rightarrow$ (resp. $\overset{*}{\rightarrow}$) instead of $\underset{S,I}{\rightarrow}$ (resp. $\underset{S,I}{\overset{*}{\rightarrow}}$) if $S$ and $I$ are understood from the context.

# 3 Security Flaws

## 3.1 Authorization

Various authorization models have been proposed for OODBs (e.g., [4],[5],[7],[12]). In this paper, however, discussing authorization models is not our main purpose, and therefore we adopt the following general and simple authorization model.

**Definition 7:** Let $S = (C, \leq, M, \Sigma_b, \Sigma_u)$ be a method schema. An *authorization* $A$ for a user $u$ under $S$ is a finite set of $(m, \vec{c})$, where $m \in M_n$ and $\vec{c} \in C^n$. Intuitively, $(m, \vec{c}) \in A$ means that $u$ is permitted to invoke method $m$ on any tuple $\vec{o}$ of objects such that $\vec{o} \in \nu(\vec{c})$. We simply write $(m, c)$ instead of $(m, (c))$ for unary methods. □

In many articles (e.g., [4],[12],[15]), an authorization is modeled by a base authorization and a set of inference rules. An example of an inference rule is "if $u$ is permitted to invoke $m$ on objects of $c$, then $u$ is also permitted to invoke $m$ on objects of the subclasses of $c$." By this rule, a base authorization $\{(m, c)\}$ is expanded into $\{(m, c), (m, c_1), (m, c_2)\}$ if $c_1 \leq c$ and $c_2 \leq c$. In this paper, we assume that a given authorization has already been expanded.

**Example 4:** Let $A_1$ be an authorization for a user $u$ under $S_1$ in Example 2 as follows:

$$\begin{aligned} A_1 = \{ & a_1 : (\text{boss}, \textit{manager}), \\ & a_2 : (\text{boss}, \textit{employee}), \\ & a_3 : (\text{partner}, \textit{person}), \\ & a_4 : (\text{clerk}, \textit{manager}), \\ & a_5 : (\text{leader\_of\_customer}, \textit{person})\}. \end{aligned}$$

Consider the interpretation $I_1$ in Example 3. Executing boss(`Silver`) by $u$ is permitted since $\texttt{Silver} \in \nu(\textit{employee})$ and $(\text{boss}, \textit{employee}) \in A_1$. On the other hand, leader(`Green`, `Silver`) is prohibited since $\texttt{Green} \in \nu(\textit{manager})$, $\texttt{Silver} \in \nu(\textit{employee})$, but $(\text{leader}, (\textit{manager}, \textit{employee})) \notin A_1$. □

## 3.2 Formal Definition of User's Inference

Let $S = (C, \leq, M, \Sigma_b, \Sigma_u)$ be a method schema and $A$ the authorization for a user $u$. In this paper, we are interested in deciding whether or not for a given term $\tau \in T_M(C)$, there exists an interpretation $I = (\nu, \mu)$ such that $u$ can infer $\tau[\vec{o}/\vec{c}] \downarrow = o$ for some $\vec{o} \in \nu(\vec{c})$ and $o \in O_{S,I}$.

In this section, we define the information which $u$ can obtain under a given interpretation $I = (\nu, \mu)$. First, we consider the information on the existence of objects. Let $\tilde{O}$ denote the set of objects such that $o \in \tilde{O}$ *iff* $u$ knows that $o$ exists in $O_{S,I}$. Formally, $\tilde{O}$ is defined as the smallest set satisfying the following $(*1)$ and $(*2)$:

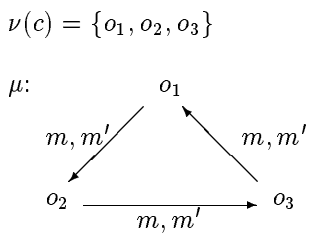


Fig. 3: An example of interpretation.

| $x$ | $m(x)\downarrow$ | $m'(x)\downarrow$ | $m_1(x)\downarrow$ |
|---|---|---|---|
| $o_1$ | $o_2$ | $o_2$ | $o_3$ |
| $o_2$ | $o_3$ | $o_3$ | $o_1$ |
| $o_3$ | $o_1$ | $o_1$ | $o_2$ |

Fig. 4: An example of method executions.

(∗1) $\tilde{O}$ contains a fixed set $\tilde{O}_0$. For each $o \in \tilde{O}_0$, the existence of $o$ in $O_{S,I}$ is regarded as a priori knowledge of $u$.

(∗2) Suppose that $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, and $(m, \vec{c}) \in A$. Also, suppose that there exists $o \in O_{S,I}$ such that $m(\vec{o})\downarrow = o$. Then, $\tilde{O}$ contains $o$.

By (∗1), we assume that $u$ has partial knowledge on $I$ in advance. (∗2) is derived from Policy 1 in Sect. 1, i.e., $u$ can obtain the result of $m(\vec{o})$. Secondly, we consider the information on classes.

(∗3) For each object $o \in \tilde{O}$, $u$ knows the class $c$ such that $o \in \nu(c)$. Thus, $u$ knows that a class $c$ exists in $C$ *iff* there exists $o \in \tilde{O}$ such that $o \in \nu(c)$.

(∗4) User $u$ knows whether $c \leq c'$ holds or not *iff* $u$ knows the existence of both $c$ and $c'$.

By (∗3), we mean that for a given object $o$, $u$ can obtain the class to which $o$ belongs. By (∗4), we mean that for $c$ and $c'$ whose existence $u$ knows, $u$ can obtain the (possible) superclass-subclass relationship between $c$ and $c'$. Lastly, we consider the information on equalities of terms which $u$ can obtain directly from the database. Let $m \in M_n$, $\vec{o} \in O^n_{S,I}$, and $t \in T_M(\{x_1, x_2, \ldots, x_n\})$.

(∗5) User $u$ knows that $m(\vec{o})\downarrow = o$ holds *iff* it is the case that $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, $(m, \vec{c}) \in A$, $o \in O_{S,I}$, and $m(\vec{o})\downarrow = o$.

(∗6) User $u$ knows that $Res(m, \vec{c}) = t$ holds *iff* it is the case that $(m, \vec{c}) \in A$ and $Res(m, \vec{c}) = t$.

(∗5) is derived from Policy 1 in Sect. 1. (∗6) corresponds to Policy 2 in Sect. 1, i.e., $u$ can obtain the type declaration of $m$ at $\vec{c}$ (if $m$ is a base method) and the external specification of $m$ at $\vec{c}$ (if $m$ is a user method).

Our goal is to model what the user can infer as a congruence closure of a finite set of ground equalities, which has many good properties. To achieve this, we suppose that the following two conditions hold:

1. The user does not know what $O_{S,I}$ is.
2. The user does not know what $C$ is.

In many cases, these conditions are satisfied by just hiding $O_{S,I}$ and $C$ from the user. Then, what can the user infer under the conditions?

First of all, we suppose that the user can use at least four kinds of inference rules: reflexivity, symmetry, transitivity, and substitutivity. The four inference rules yield equalities from equalities and their contrapositions yield inequalities from inequalities. However, Condition 1 implies that inequalities are useless to infer $o$ such that $\tau[\vec{o}/\vec{c}]\downarrow = o$, as in the following example.

**Example 5:** Consider the following schema:

$$\begin{aligned} C &= \{c\}, \\ M &= \{m, m', m_1\}, \\ \Sigma_b &= \{(m, (c \to c)), (m', (c \to c))\}, \\ \Sigma_u &= \{(m_1, (c), m'(m(x))\}. \end{aligned}$$

Also, consider the interpretation $I = (\nu, \mu)$ shown in Fig. 3. In the figure, the arrow labeled by $m$ from $o_1$ to $o_2$ shows that $\mu(m)(o_1) = o_2$. Fig. 4 shows the results of all the method executions. Suppose that $\tilde{O} = \{o_1, o_2, o_3\}$.

Assume that $(m, c)$ and $(m_1, c)$ are in the authorization for a user $u$. Then, $u$ can obtain an equality $m'(o_2)\downarrow = o_3$ in the following way:

| | | |
|---|---|---|
| (i) | $m(o_1)\downarrow = o_2$ | by (∗5), |
| (ii) | $m_1(o_1)\downarrow = o_3$ | by (∗5), |
| (iii) | $Res(m_1, c) = m'(m(x))$ | by (∗6), |
| (iv) | $o_1 \in \nu(c)$ | by (∗3), |
| (v) | $m_1(o_1)\downarrow = m'(m(o_1))\downarrow$ | by (iii) and (iv), |
| (vi) | $m'(m(o_1))\downarrow = o_3$ | by (ii), (v) and transitivity, |
| (vii) | $m'(o_2)\downarrow = o_3$ | by (i), (vi) and substitutivity. |

On the other hand, assume that $(m', c)$ and $(m_1, c)$ are in the authorization. Then, $u$ can obtain that $m'(o_1)\downarrow = o_2$ and $m'(o_3)\downarrow = o_1$ by (∗5) as well as the above (vi). Therefore, $u$ can obtain inequalities $m(o_1)\downarrow \neq o_1$ and $m(o_1)\downarrow \neq o_3$ by the contraposition of substitutivity.

In what follows, we examine effects on users' inference by Condition 1. For this authorization, if $u$ knows that $O_{S,I} = \{o_1, o_2, o_3\}$, then $u$ can obtain $m(o_1)\downarrow = o_2$. On the other hand, if $u$ does not know what $O_{S,I}$ is, then $u$ cannot infer $m(o_1)\downarrow = o_2$ since $u$ cannot neglect the case that there exists another object $o \neq o_2$ such that $m(o_1)\downarrow = o$. □

If the user knows what $O_{S,I}$ is, then a disjunction of equalities would be inferred from a conjunction of inequalities. However, by Condition 1, inequalities are useless to infer $o$ such that $\tau[\vec{o}/\vec{c}]\downarrow = o$, and hence we have only to consider equalities. Thus, what the user can infer is modeled as the congruence closure of the equalities which the user can obtain directly from the database by $(*5)$ and $(*6)$.

Equalities obtained by $(*6)$ are not ground (i.e., include variables). However, together with Condition 2, they are equivalent to a finite set of ground equalities.

**Example 6:** Consider the following schema:

$$
\begin{aligned}
C &= \{c, c'\},\\
M &= \{m, m', m_1\},\\
\Sigma_b &= \{(m, (c' \to c')), (m', (c' \to c'))\},\\
\Sigma_u &= \{(m_1, (c'), m'(x))\},
\end{aligned}
$$

and $c \leq c'$. Also, consider the following authorization for a user $u$:

$$\{(m_1, c), (m_1, c')\}$$

and an interpretation such that

$$o \in \nu(c').$$

Suppose that $u$ knows that $C = \{c, c'\}$ and $c \leq c'$. Then, $u$ can obtain that $m(o)\downarrow \in \nu(c) \cup \nu(c')$ if $m(o)\downarrow \in O_{S,I}$. Moreover, since $Res(m_1, c) = Res(m_1, c') = m'(x)$, $u$ can infer that $m_1(m(o))\downarrow = m'(m(o))\downarrow$ if both $m_1(m(o))\downarrow$ and $m'(m(o))\downarrow$ are in $O_{S,I}$. Note that, in this inference, $u$ does not need to know which class $m(o)\downarrow$ belongs to.

On the other hand, if Condition 2 is satisfied, then $u$ cannot conclude that $m_1(m(o))\downarrow = m'(m(o))\downarrow$ without exactly inferring the class to which $m(o)\downarrow$ belongs since $u$ cannot neglect the case that $m(o)\downarrow \in \nu(c'')$ for some $c''$ other than $c$ and $c'$ such that $Res(m_1, c'') \neq m'(x)$. By $(*3)$, to know the class to which $m(o)\downarrow$ belongs is to infer the exact value of $m(o)\downarrow$. Thus, the equalities obtained by $(*6)$ can be applied only to terms $t$ such that $t\downarrow$ is known. This means that the equalities obtained by $(*6)$ can be regarded as ground. More precisely, $Res(m, \vec{c}) = t$ is regarded as $\{m(\vec{o})\downarrow = t[\vec{o}/\vec{x}]\downarrow | \vec{o} \in \tilde{O}^n \text{ and } \vec{o} \in \nu(\vec{c})\}$. □

Thus, we can model what the user can infer as a congruence closure of a finite set of ground equalities induced by $(*5)$ and $(*6)$. Moreover, the congruence closure is identical to $\overset{*}{\underset{A,I}{\Rightarrow}}$ defined below because of the correctness of Knuth-Bendix completion [13].

**Definition 8:** Consider the minimum set $P_I$ of rewriting rules $\underset{A,I}{\triangleright}$ on $T_M(O_{S,I})$ satisfying the following three conditions:

(1) If $(m, \vec{c}) \in A$, $m \in M_n$, $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, and $m(\vec{o}) \xrightarrow{*} o \in O_{S,I}$, then $m(\vec{o}) \underset{A,I}{\triangleright} o \in P_I$.

(2) If $(m_u, \vec{c}) \in A$, $m_u \in M_{u,n}$, $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, $m_u(\vec{o}) \xrightarrow{*} o \in O_{S,I}$, and $Res(m_u, \vec{c}) = t \in T_M(\{\vec{x}\})$, then $t[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o \in P_I$.

(3) If $t''$ is a proper subterm of $t$ at $r''$ $(t, t'' \in T_M(O_{S,I}))$ and $t \underset{A,I}{\triangleright} o$, $t'' \underset{A,I}{\triangleright} o'' \in P_I$ $(o, o'' \in O_{S,I})$, then $t[r'' \leftarrow o''] \underset{A,I}{\triangleright} o \in P_I$.

Then, define $\underset{A,I}{\Rightarrow}$ as the one-step reduction relation by $\underset{A,I}{\triangleright}$. That is, $t \underset{A,I}{\Rightarrow} t'$ *iff* there exists a subterm $t''$ of $t$ at $r''$ such that $t'' \underset{A,I}{\triangleright} o'' \in P_I$ and $t' = t[r'' \leftarrow o'']$. Let $\overset{*}{\underset{A,I}{\Rightarrow}}$ denote the reflexive and transitive closure of $\underset{A,I}{\Rightarrow}$. Note that the existence of $t \underset{A,I}{\triangleright} o \in P_I$ implies $t \xrightarrow{*} o$. □

## 3.3 The Detection Problem

**Definition 9:** The *detection problem of security flaws for database schemas* is to decide whether or not, for given $S$, $A$, and $\tau \in T_M(C)$, there exist an interpretation $I = (\nu, \mu)$ and $\tilde{O}$ such that $\tau[\vec{o}/\vec{c}] \overset{*}{\underset{A,I}{\Rightarrow}} o$ for some $\vec{o} \in \nu(\vec{c})$ and $o \in O_{S,I}$. □

This problem is undecidable for general method schemas. However, it is decidable in polynomial time for monadic method schemas as will be stated in the next section.

**Theorem 1:** The detection problem of security flaws for method schemas with methods of arity two is undecidable.

**Sketch of Proof:** Ref. [11] shows that the type-consistency problem for method schemas with methods of arity two is undecidable by reducing the Post's Correspondence Problem (PCP) to the problem. In the reduction, each interpretation $I$ is regarded as a candidate for a solution to a PCP. If $I$ is actually a solution, then execution of a term, say $m(o)$, is aborted under $I$. Otherwise, $m(o)$ is nonterminating. By slightly modifying the reduction in [11], we can construct a schema as follows:

- If $I$ is a solution, then the execution of a term, say $m'(o)$, is successful under $I$.
- Otherwise, $m'(o)$ is nonterminating under $I$.

Let $c$ be the class to which $o$ belongs. Also, let $A = \{(m', c)\}$ and $\tau = m'(c)$. Then, we have that the PCP has a solution *iff* there exist $I = (\nu, \mu)$ and $\tilde{O}$ such that $\tau[o/c] \overset{*}{\underset{A,I}{\Rightarrow}} o'$ for some $o$ and $o'$. □

# 4 Security Analysis

## 4.1 A Sufficient Condition

In this section we propose a decidable sufficient condition for a given schema to have no security flaw. The main idea is to approximate $\underset{A,I}{\triangleright}$ using classes. To do this, we use a mapping $Z : T_M(C) \rightarrow 2^C$ which has the following property:

$$Z(t) \supseteq \{c | \text{ there exists an interpretation such that } t[\vec{o}/\vec{c}] \xrightarrow{*} o, \vec{o} \in \nu(\vec{c}), o \in \nu(c)\}.$$

Intuitively, $Z(t)$ contains all the classes $c$ such that the result of successful execution of $t[\vec{o}/\vec{c}]$ is an object $o$ for some $\vec{o} \in \nu(\vec{c})$ and $o \in \nu(c)$. The smaller $Z(t)$ is, the better approximation we have. In the following, we define rewriting rules $\underset{A,S}{\triangleright}$ on $T_M(C)$ which approximate $\underset{A,I}{\triangleright}$.

**Definition 10:** Consider the minimum set $P_S$ of rewriting rules $\underset{A,S}{\triangleright}$ on $T_M(C)$ satisfying the following three conditions:

(1) If $(m, \vec{c}) \in A$ and $c \in Z(m(\vec{c}))$, then $m(\vec{c}) \underset{A,S}{\triangleright} c \in P_S$.

(2) If $(m_u, \vec{c}) \in A$, $m_u \in M_{u,n}$, $Res(m_u, \vec{c}) = t \in T_M(\{\vec{x}\})$, and $c \in Z(t[\vec{c}/\vec{x}])$, then $t[\vec{c}/\vec{x}] \underset{A,S}{\triangleright} c \in P_S$.

(3) If $t''$ is a proper subterm of $t$ at $r''$ $(t, t'' \in T_M(C))$ and $t \underset{A,S}{\triangleright} c$, $t'' \underset{A,S}{\triangleright} c'' \in P_S$ $(c, c'' \in C)$, then $t[r'' \leftarrow c''] \underset{A,S}{\triangleright} c' \in P_S$ for each $c' \in Z(t[r'' \leftarrow c''])$.

Then, define $\underset{A,S}{\Rightarrow}$ as the one-step reduction relation by $\underset{A,S}{\triangleright}$. Let $\overset{*}{\underset{A,S}{\Rightarrow}}$ denote the reflexive and transitive closure of $\underset{A,S}{\Rightarrow}$. □

The following lemma states the relationship between rewriting rules $\underset{A,I}{\triangleright}$ and $\underset{A,S}{\triangleright}$.

**Lemma 1:** If there exists an interpretation $I = (\nu, \mu)$ such that $t[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o \in P_I$ for some $\vec{o} \in \nu(\vec{c})$ and $o \in \nu(c)$, then $t[\vec{c}/\vec{x}] \underset{A,S}{\triangleright} c \in P_S$.

**Proof:** We use induction on the number of the repetition of a procedure which computes the least fixed point satisfying the three conditions in Def. 8.

*Basis*: We consider the following two cases:

1. The case that $m(\vec{o}) \underset{A,I}{\triangleright} o$ is obtained from Def. 8 (1): It holds that $(m, \vec{c}) \in A$, $\vec{o} \in \nu(\vec{c})$, and $m(\vec{o}) \xrightarrow{*} o$. From the property of $Z$, there exists a class $c$ such that $c \in Z(m(\vec{c}))$ and $o \in \nu(c)$. From Def. 10 (1), we have $m(\vec{c}) \underset{A,S}{\triangleright} c$ since $(m, \vec{c}) \in A$ and $c \in Z(m(\vec{c}))$.

2. The case that $Res(m_u, \vec{c})[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o$ is obtained from Def. 8 (2): In the same way of the above 1, it holds that $(m_u, \vec{c}) \in A$, $Res(m_u, \vec{c}) = t$, and $c \in Z(t[\vec{o}/\vec{x}])$. From Def. 10 (2), we have $t[\vec{c}/\vec{x}] \underset{A,S}{\triangleright} c$.

*Induction*: Suppose that $t''[\vec{o''}/\vec{x''}]$ is a proper subterm of $t[\vec{o}/\vec{x}]$ at $r''$ and that $t[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o$ and $t''[\vec{o''}/\vec{x''}] \underset{A,I}{\triangleright} o''$ have been obtained. Let $t'[\vec{o'}/\vec{x'}] = t[\vec{o}/\vec{x}][r'' \leftarrow o'']$ ($\vec{o'} \in \nu(\vec{c'})$), and suppose that $t'[\vec{o'}/\vec{x'}] \underset{A,I}{\triangleright} o$ is obtained from Def. 8 (3). By the inductive hypothesis, it holds that

$$t[\vec{c}/\vec{x}] \underset{A,S}{\triangleright} c \in P_S,\ \vec{o} \in \nu(\vec{c}),\ o \in \nu(c),$$
$$t''[\vec{c''}/\vec{x''}] \underset{A,S}{\triangleright} c'' \in P_S,\ \vec{o''} \in \nu(\vec{c''}),\ o'' \in \nu(c'').$$

Since $t'[\vec{o'}/\vec{x'}] = t[\vec{o}/\vec{x}][r'' \leftarrow o'']$, we have $t'[\vec{c'}/\vec{x'}] = t[\vec{c}/\vec{x}][r'' \leftarrow c'']$. Since $t'[\vec{o'}/\vec{x'}] \underset{A,I}{\triangleright} o \in P_I$ implies $t'[\vec{o'}/\vec{x'}] \xrightarrow{*} o$, it holds that $c \in Z(t'[\vec{c'}/\vec{x'}])$. From the above inductive hypothesis and Def. 10 (3), we have $t'[\vec{c'}/\vec{x'}] \underset{A,S}{\triangleright} c \in P_S$. □

The following lemma states the relationship between reduction relations $\underset{A,I}{\overset{*}{\Rightarrow}}$ and $\underset{A,S}{\overset{*}{\Rightarrow}}$.

**Lemma 2:** If $t[\vec{o}/\vec{x}] \underset{A,I}{\overset{*}{\Rightarrow}} t'[\vec{o'}/\vec{x'}]$ for some $\vec{o} \in \nu(\vec{c})$ and $\vec{o'} \in \nu(\vec{c'})$, then $t[\vec{c}/\vec{x}] \underset{A,S}{\overset{*}{\Rightarrow}} t'[\vec{c'}/\vec{x'}]$.

**Proof:** Consider the $i$-th step $t_{i-1}[\vec{o}_{i-1}/\vec{x}_{i-1}] \underset{A,I}{\Rightarrow} t_i[\vec{o}_i/\vec{x}_i]$ of the reduction $t[\vec{o}/\vec{x}] \underset{A,I}{\overset{*}{\Rightarrow}} t'[\vec{o'}/\vec{x'}]$. By Lemma 1, it is easily shown that $t_{i-1}[\vec{c}_{i-1}/\vec{x}_{i-1}] \underset{A,S}{\Rightarrow} t_i[\vec{c}_i/\vec{x}_i]$, where $\vec{o}_{i-1} \in \nu(\vec{c}_{i-1})$ and $\vec{o}_i \in \nu(\vec{c}_i)$. Since this reduction holds at an arbitrary step, the lemma holds. □

By Lemma 2, we immediately have the following theorem.

**Theorem 2:** Let $S$ be a method schema and $\tau[\vec{c}/\vec{x}]$ a term to be verified. If there exists no class $c$ such that $\tau[\vec{c}/\vec{x}] \underset{A,S}{\overset{*}{\Rightarrow}} c$, then $S$ has no security flaw, i.e., there exist no interpretation $I$ and $\tilde{O}$ such that $\tau[\vec{o}/\vec{x}] \underset{A,I}{\overset{*}{\Rightarrow}} o$ for any $\vec{o} \in \nu(\vec{c})$ and $o \in O_{S,I}$. □

## 4.2 Monadic Case

In this section we show that the sufficient condition in Theorem 2 is also a necessary one if a given method schema is monadic and $Z$ satisfies that for each $t \in T_M(C)$,

$$Z(t) = \{c' \mid \text{there exists an interpretation such that } t[o/c] \xrightarrow{*} o', o \in \nu(c), o' \in \nu(c')\}. \quad (1)$$

**Definition 11:** Let $N$ be a positive integer. Define a *syntactic interpretation* $I_S = (\nu_{I_S}, \mu_{I_S})$ of $S$ as follows:

1. For each $c \in C$, $\nu_{I_S}(c) = \{c \cdot \alpha | \alpha \in C^*$ and the length of $c \cdot \alpha$ is at most $N\}$, where $C^*$ denotes the Kleene closure of $C$.

2. For each $m_b \in M_b$, define $\mu_{I_S}(m_b)$ as follows:

   2.1 If the resolution of $m_b$ at $c_0$ is $(m_b, (\ldots \rightarrow c'))$, then $\mu_{I_S}(m_b)(c_0) = c'$, and for $w \geq 1$,

   $$\mu_{I_S}(m_b)(c_0 \cdot c_1 \cdot c_2 \cdot \cdots \cdot c_w) = \begin{cases} c_1 \cdot c_2 \cdot \cdots \cdot c_w & \text{if } c_1 \leq c', \\ c' \cdot c_2 \cdot \cdots \cdot c_w & \text{otherwise.} \end{cases}$$

   2.2 If the resolution of $m_b$ at $c_0$ does not exist, then $\mu_{I_S}(m_b)(c_0 \cdot c_1 \cdot c_2 \cdot \cdots \cdot c_w)$ $(w \geq 0)$ is undefined. □

Hereafter we consider a syntactic interpretation with sufficiently large $N$. If $Z$ satisfies Eq. (1), then it also satisfies the following Lemmas 3 through 5.

**Lemma 3:** Let $t \in T_M(\{x\})$, and suppose that $c' \in Z(t[c/x])$. There exists $\beta \in C^*$ such that

1. the first symbol of $\beta \cdot c'$ is $c$, and

2. for each $\alpha \in C^*$ such that $\beta \cdot c' \cdot \alpha$ is an object of $I_S$ (i.e., the length of $\beta \cdot c' \cdot \alpha$ is at most $N$),

   $$t[\beta \cdot c' \cdot \alpha / x] \xrightarrow{*} c' \cdot \alpha.$$

We call $\beta$ a *reduction string* of $(t[c/x], c')$.

**Proof:** Suppose that $c' \in Z(t[c/x])$. By Eq. (1), there exists an interpretation $I = (\nu, \mu)$ such that $t[o/x] \xrightarrow{*} o'$ for some $o \in \nu(c)$ and $o' \in \nu(c')$. Consider the $i$-th step (counting from zero) $t_i[o_i/x] \rightarrow t_{i+1}[o_{i+1}/x]$ of the reduction $t[o/x] \xrightarrow{*} o'$, where $t_0[o_0/x] = t[o/x]$ and $t_n[o_n/x] = o'$. Let $c_i$ $(0 \leq i \leq n-1)$ be the class such that $o_i \in \nu(c_i)$ and $m_i(o_i)$ the innermost term of $t_i[o_i/x]$. Define $\beta_i$ $(0 \leq i \leq n-1)$ as follows:

$$\beta_i = \begin{cases} c_i & \text{if } m_i \in M_b, \\ \varepsilon \text{ (empty string)} & \text{otherwise.} \end{cases}$$

In what follows, we show that $\beta = \beta_0 \cdot \cdots \cdot \beta_{n-1}$ satisfies the conditions of this lemma.

It is easily verified that $\beta$ satisfies condition 1 since

$o_0 = o \in \nu(c)$, and

if $m_i \in M_u$, then $o_{i+1} = o_i$ by the definition of $\rightarrow$.

To see that $\beta$ also satisfies condition 2, consider the execution of $t_0[\beta \cdot c' \cdot \alpha/x]$ for an arbitrary $\alpha \in C^*$. If $m_0 \in M_b$, then $\beta_0 = c_0$, and thus $t_0[\beta \cdot c' \cdot \alpha/x] \rightarrow t_1[\beta_1 \cdot \cdots \cdot \beta_{n-1} \cdot c' \cdot \alpha/x]$. On the other hand, if $m_0 \in M_u$, then $\beta_0 = \varepsilon$, and thus $t_0[\beta \cdot c' \cdot \alpha/x] \rightarrow t_1[\beta_1 \cdot \cdots \cdot \beta_{n-1} \cdot c' \cdot \alpha/x]$. In either case,

$$t_0[\beta \cdot c' \cdot \alpha/x] = t_0[\beta_0 \cdot \beta_1 \cdot \cdots \cdot \beta_{n-1} \cdot c' \cdot \alpha/x] \rightarrow t_1[\beta_1 \cdot \cdots \cdot \beta_{n-1} \cdot c' \cdot \alpha/x].$$

By repeating this discussion, we have $t[\beta \cdot c' \cdot \alpha/x] \xrightarrow{*} c' \cdot \alpha$. □

**Lemma 4:** Let $t, t', t'' \in T_M(\{x\})$ such that $t''$ is a subterm of $t$ at $r''$ and $t' = t[r'' \leftarrow x]$. If both $c'' \in Z(t''[c/x])$ and $c' \in Z(t'[c''/x])$, then $c' \in Z(t[c/x])$.
**Proof:** By Lemma 3, there exist reduction strings $\beta''$ of $(t''[c/x], c'')$ and $\beta'$ of $(t'[c''/x], c')$, i.e., for any $\alpha'', \alpha' \in C^*$,

$$\begin{array}{ll} t''[\beta'' \cdot c'' \cdot \alpha''/x] \xrightarrow{*} c'' \cdot \alpha'', & \text{the first symbol of } \beta'' \cdot c'' \text{ is } c, \\ t'[\beta' \cdot c' \cdot \alpha'/x] \xrightarrow{*} c' \cdot \alpha', & \text{the first symbol of } \beta' \cdot c' \text{ is } c''. \end{array}$$

When we choose $\alpha''$ so that $c'' \cdot \alpha'' = \beta' \cdot c' \cdot \alpha'$, we have

$$t[\beta'' \cdot \beta' \cdot c' \cdot \alpha'/x] \xrightarrow{*} c' \cdot \alpha'.$$

Since $Z$ satisfies Eq. (1), $c'$ must be in $Z(t[c/x])$. □

**Lemma 5:** Let $t, t', t'' \in T_M(\{x\})$ such that $t''$ is a subterm of $t$ at $r''$ and $t' = t[r'' \leftarrow x]$. Suppose that $c'' \in Z(t''[c/x])$ and $c' \in Z(t'[c''/x])$. Let $\beta'$ be an arbitrary reduction string of $(t'[c''/x], c')$. Then, there exist reduction strings $\beta$ of $(t[c/x], c')$ and $\beta''$ of $(t''[c/x], c'')$ such that $\beta = \beta'' \cdot \beta'$.
**Proof:** Let $\beta''$ and $\beta'$ be arbitrary reduction strings of $(t''[c/x], c'')$ and $(t'[c''/x], c')$, respectively. By the proof of Lemma 4, $\beta'' \cdot \beta'$ is a reduction string of $(t[c/x], c')$. This fact implies the lemma. □

Suppose that $\tilde{O} = O_{S,I_S}$. We show that $t[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} t'[c'/x]$ implies $t[o/x] \underset{A,I_S}{\overset{*}{\Rightarrow}} t'[o'/x]$ for some $o \in \nu(c)$ and $o' \in \nu(c')$. The next lemma states the relationship between $\underset{A,I_S}{\triangleright}$ and $\underset{A,S}{\triangleright}$.

**Lemma 6:** Let $c, c' \in C$, $t \in T_M(\{x\})$, and $t' \in T_M(\{x'\})$. If $t[c/x] \underset{A,S}{\triangleright} c' \in P_S$, then for an arbitrary reduction string $\beta$ of $(t[c/x], c')$ and for any $\alpha \in C^*$,

$$t[\beta \cdot c' \cdot \alpha/x] \underset{A,I_S}{\triangleright} c' \cdot \alpha \in P_{I_S}.$$

**Proof:** We use induction on the number of the repetition of a procedure which computes the least fixed point satisfying the three conditions in Def. 10.
*Basis*: We consider the following two cases:

1. The case that $m(c) \underset{A,S}{\triangleright} c'$ is obtained from Def. 10 (1): Let $\beta$ be an arbitrary reduction string of $(m(c), c')$. Since $(m, c) \in A$ and $m(\beta \cdot c' \cdot \alpha) \xrightarrow{*} c' \cdot \alpha$, we obtain $m(\beta \cdot c' \cdot \alpha) \underset{A,I_S}{\triangleright} c' \cdot \alpha$ from Def. 8 (1).

2. The case that $Res(m_u, c)[c/x] \underset{A,S}{\triangleright} c'$ is obtained from Def. 10 (2): It can be proved similarly to the above case.

*Induction*: Suppose that there exist $c \in C$ and $t, t', t'' \in T_M(\{x\})$ such that

$t''$ is a subterm of $t$ at $r''$,
$t' = t[r'' \leftarrow x]$,
$t''[c/x] \underset{A,S}{\triangleright} c'' \in P_S$,
$t[c/x] \underset{A,S}{\triangleright} c_1 \in P_S$ for some $c_1$,
$c' \in Z(t'[c''/x])$.

Also, suppose that $t'[c''/x] \underset{A,S}{\triangleright} c'$ is obtained from Def. 10 (3). Since $t''[c/x] \underset{A,S}{\triangleright} c'' \in P_S$, it holds that $c'' \in Z(t''[c/x])$ by Def. 10. By Lemma 4, it holds that $c' \in Z(t[c/x])$. Then, by Def. 10 again, $t[c/x] \underset{A,S}{\triangleright} c'$ must be in $P_S$. Let $\beta'$ be an arbitrary reduction string of $(t'[c''/x], c')$. That is, the first symbol of $\beta' \cdot c'$ is $c''$ and $t'[\beta' \cdot c' \cdot \alpha'/x] \xrightarrow{*} c' \cdot \alpha'$ for any $\alpha' \in C^*$. By Lemma 5 and the inductive hypothesis for $t[c/x] \underset{A,S}{\triangleright} c'$ and $t''[c/x] \underset{A,S}{\triangleright} c''$, there exist $\beta$ and $\beta''$ such that, for any $\alpha$ and $\alpha''$,

$$t[\beta \cdot c' \cdot \alpha/x] \underset{A,I_S}{\triangleright} c' \cdot \alpha \in P_{I_S}, \qquad \text{the first symbol of } \beta \cdot c' \text{ is } c,$$
$$t''[\beta'' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\triangleright} c'' \cdot \alpha'' \in P_{I_S}, \quad \text{the first symbol of } \beta'' \cdot c'' \text{ is } c,$$

and $\beta = \beta'' \cdot \beta'$. When we choose $\alpha$ and $\alpha''$ so that $\beta' \cdot c' \cdot \alpha = c'' \cdot \alpha''$, it holds that $\beta \cdot c' \cdot \alpha = \beta'' \cdot c'' \cdot \alpha''$. By Def. 8 (3),

$$t[\beta \cdot c' \cdot \alpha/x][r'' \leftarrow c'' \cdot \alpha''] \underset{A,I_S}{\triangleright} c' \cdot \alpha \in P_{I_S}.$$

Since $t[\beta \cdot c' \cdot \alpha/x][r'' \leftarrow c'' \cdot \alpha''] = t'[c'' \cdot \alpha''/x] = t'[\beta' \cdot c' \cdot \alpha/x]$, it holds that $t'[\beta' \cdot c' \cdot \alpha/x] \underset{A,I_S}{\triangleright} c' \cdot \alpha \in P_{I_S}$ for any $\alpha$. □

**Lemma 7:** Let $c, c' \in C$ and $t, t' \in T_M(\{x\})$. If $t[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} t'[c''/x]$, then there exists a string $\beta$ such that the first symbol $\beta \cdot c''$ is $c$ and for any $\alpha'' \in C^*$,

$$t[\beta \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\overset{*}{\Rightarrow}} t'[c'' \cdot \alpha''/x].$$

**Proof:** We use induction on the length of the reduction $t[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} t'[c''/x]$.

*Basis*: Suppose that $t[c/x] \underset{A,S}{\Rightarrow} t'[c''/x]$. By Def. 10, there exists a subterm $t''$ of $t$ at $r''$ such that $t''[c/x] \underset{A,S}{\triangleright} c''$ and $t'[c''/x] = t[c/x][r'' \leftarrow c'']$. By Lemmas 3 and 6, there exists $\beta''$ such that the first symbol of $\beta'' \cdot c''$ is $c$ and for any $\alpha'' \in C^*$, rule $t''[\beta'' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\triangleright} c'' \cdot \alpha''$ exists in $P_{I_S}$. By Def. 8, it follows that $t''[\beta'' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\Rightarrow} c'' \cdot \alpha''$. Hence, $t[\beta'' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\Rightarrow} t''[c'' \cdot \alpha''/x]$.

*Induction*: Consider the following reduction that $t[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} t_i[c_i/x] \underset{A,S}{\Rightarrow} t'[c''/x]$. By the inductive hypothesis, there exists a string $\beta_i$ such that the first symbol of $\beta_i \cdot c_i$ is $c$ and for any $\alpha_i \in C^*$,

$$t[\beta_i \cdot c_i \cdot \alpha_i/x] \underset{A,I_S}{\overset{*}{\Rightarrow}} t_i[c_i \cdot \alpha_i/x].$$

In the same way as the basis case, there exists a string $\beta'$ such that the first symbol of $\beta' \cdot c''$ is $c_i$ and for any $\alpha'' \in C^*$,

$$t_i[\beta' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\Rightarrow} t'[c'' \cdot \alpha''/x].$$

We can choose $\alpha_i$ so that $\beta' \cdot c'' \cdot \alpha'' = c_i \cdot \alpha_i$. Therefore, it follows that

$$t[\beta_i \cdot \beta' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\overset{*}{\Rightarrow}} t'[c'' \cdot \alpha''/x].$$

Clearly, $\beta = \beta_i \cdot \beta'$ satisfies the condition of the lemma. □

**Theorem 3:** Let $S$ be a monadic method schema and $\tau[c/x]$ a term to be verified. Suppose that $Z$ satisfies Eq. (1). If there exists a class $c'$ such that $\tau[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} c'$, then $S$ has a security flaw, i.e., there exist an interpretation $I$ and $\tilde{O}$ such that $\tau[o/x] \underset{A,I}{\overset{*}{\Rightarrow}} o'$ for some $o \in \nu(c)$ and $o' \in O_{S,I}$. □

## 4.3 The Algorithm and its Complexity

The algorithm for deciding the sufficient condition stated in Theorem 2 consists of the following three steps.

(A1) Compute $Z$ from $S$.

(A2) Compute $P_S$ from $S$, $A$, and $Z$.

(A3) Determine whether there exists a class $c$ such that $\tau[\vec{c}/\vec{x}] \underset{A,S}{\overset{*}{\Rightarrow}} c$. If such $c$ exists, then output "a security flaw may exist." Otherwise, output "no security flaw exists."

Using the type-checking algorithm in [14], we can compute $Z$ which is fairly small, and for a monadic schema, we can compute $Z$ satisfying Eq. (1) in polynomial time of the size of $S$.

Now, we summarize the time complexity of the algorithm. Define the size $\|t\|$ of a term $t$ as the number of nodes in the tree representing $t$. For each $(m, \vec{c}, t) \in \Sigma_u$, define the description length of $(m, \vec{c}, t)$ as $\|t\|$. Define the description length of $\Sigma_u$, denoted $\|\Sigma_u\|$, as the sum of $\|t\|$ for all $(m, \vec{c}, t) \in \Sigma_u$. Also, define the size of $S$, denoted $N$, as follows:

$$N = |C| + | \leq | + |M| + |\Sigma_b| + \|\Sigma_u\|,$$

where $|X|$ denotes the number of elements of a set $X$. Let $k$ be the maximum arity of all methods, $H$ the maximum size of all $t$ such that $(m, \vec{c}, t) \in \Sigma_u$, and $l$ the size of a given term to be verified.

Before executing (A1), we define a finite set $T_A$ such that $t \underset{A,S}{\triangleright} c \in P_S$ *iff* $t \in T_A$ and $c \in Z(t)$. In what follows, we consider the size of $T_A$. Let $h$ be the height of a tree representing a term $t \in T_M(C)$. Also, let $X_h$ be a finite set of subtrees obtained by replacing subterms of $t$ with classes, where $X_h$ contains both $t$ itself and $c$ such that $c \in Z(t)$. For example, let $t = m_1(m_2(c_1, c_2), c_3)$, which is represented as a tree with height two. If $Z(m_2(c_1, c_2)) = \{c_1, c_2\}$, $Z(m_1(c_1, c_3)) = \{c_1\}$, and $Z(m_1(c_2, c_3)) = \{c_2\}$, then for such $t$,

$$X_2 = \{m_1(m_2(c_1, c_2), c_3), m_1(c_1, c_3), m_1(c_2, c_3), c_1, c_2\}.$$

By solving the following recurrence formula

$$\begin{cases} |X_0| \leq |C|, \\ |X_h| \leq |X_{h-1}|^k + |C|, \end{cases}$$

we can obtain

$$\begin{cases} |X_h| = \mathcal{O}(|C|^{k^h}) & \text{if } k \geq 2, \\ |X_h| = \mathcal{O}(h \cdot |C|) & \text{if } k = 1. \end{cases}$$

Since $T_A$ is contained by $X_h$ for all terms $t \in T_M(C)$ such that $m \in M_u$ and $t = Res(m, \vec{c})[\vec{c}/\vec{x}]$,

$$|T_A| = \begin{cases} \mathcal{O}(\|\Sigma_u\| \cdot |X_H|) = \mathcal{O}(N \cdot |C|^{k^H}) & \text{if } k \geq 2, \\ \mathcal{O}(\|\Sigma_u\| \cdot |C|) = \mathcal{O}(N \cdot |C|) & \text{if } k = 1. \end{cases}$$

After all, for $k \geq 1$,

$$|T_A| = \mathcal{O}(N \cdot |C|^{k^H}).$$

First, we consider (A1). Before computing $Z$, we define a finite subset $Z_0(m(\vec{c}))$ of $C$ such that for $m \in M_n$ and $\vec{c} \in C^n$,

$$Z_0(m(\vec{c})) = \{c | \text{ there exists an interpretation such that } m(\vec{o}) \xrightarrow{*} o, \vec{o} \in \nu(\vec{c}), o \in \nu(c)\}.$$

The time complexity to compute $Z_0(m(\vec{c}))$ for all $m \in M_n$ and $\vec{c} \in C^n$ is

$$\mathcal{O}(k \cdot N \cdot |C|^{2k+1}), \tag{2}$$

which is given in [14]. If $Z_0$ is implemented as a 2-3 tree, then the time complexity $t_{Z_0}$ to retrieve an element from $Z_0(m(\vec{c}))$ is

$$t_{Z_0} = \mathcal{O}(\log |Z_0|) = \mathcal{O}(\log(|M| \cdot |C|^k)) = \mathcal{O}(k \cdot \log N).$$

For $t \in T_M(C)$, we can compute $Z(t)$ as follows. If $m(\vec{c})$ is the innermost term of $t$, then we replace $m(\vec{c})$ with $c$ for each $c \in Z_0(m(\vec{c}))$. We repeat this replacement until $t$ is rewritten to a class. The time complexity to compute $Z(t)$ is

$$\mathcal{O}(t_{Z_0} \cdot |C|^k \cdot \|t\|) = \mathcal{O}(k \cdot H \cdot |C|^k \cdot \log N).$$

```
B1    for each (m, c⃗) in A
B2        add m(c⃗) to T_A
B3            if m ∈ M_u then
B4                let t be Res(m, c⃗)[c⃗/x⃗]
B5                add t[c⃗/x⃗] to T_A
B6    repeat
B7        for each t, t' in T_A
B8            if t' is a subterm of t at r' then
B9                for each c' in Z(t')
B10                   add t[r' ← c'] to T_A
B11   until T_A does not change
```

Fig. 5: Procedure to compute $T_A$.

Therefore, the time complexity to compute $Z(t)$ for all $t \in T_A$ is

$$\mathcal{O}(|T_A| \cdot k \cdot H \cdot |C|^k \cdot \log N) = \mathcal{O}(k \cdot H \cdot |C|^{k^H+k} \cdot N \cdot \log N). \tag{3}$$

If $Z$ is implemented as a 2-3 tree, then the time complexity $t_Z$ to retrieve an element from $Z(t)$ is

$$t_Z = \mathcal{O}(\log |Z|) = \mathcal{O}(\log |T_A|) = \mathcal{O}(k^H \cdot \log N).$$

Next, we consider (A2). In the following, we assume that $A$, $Res$, and $T_A$ are implemented as 2-3 trees. It is sufficient to compute $T_A$. Fig. 5 shows a procedure to compute $T_A$. Retrieving an element from $A$ and $Res$ takes $t_a = \mathcal{O}(k \cdot \log N)$ time and $t_r = \mathcal{O}(k \cdot \log N)$ time, respectively. Since $|T_A| = \mathcal{O}(N \cdot |C|^{k^H})$, retrieving an element from $T_A$ and inserting an element into $T_A$ takes $t_A = \mathcal{O}(\log |T_A|) = \mathcal{O}(k^H \cdot \log N)$ time. Therefore, executing (B1) through (B5) takes

$$\mathcal{O}(|A| \cdot (t_a + t_A + \log |M_u| + t_r + t_A)) = \mathcal{O}(k^H \cdot |C|^k \cdot N \cdot \log N)$$

time, and executing (B6) through (B11) takes

$$\mathcal{O}(|T_A| \cdot |T_A| \cdot (t_A + H^2 + t_Z + |C| \cdot t_A)) = \mathcal{O}(|C|^{2k^H} \cdot N^2 \cdot (H^2 + k^H \cdot |C| \cdot \log N))$$

time. The total time of (A2) is

$$\mathcal{O}(|C|^{2k^H} \cdot N^2 \cdot (H^2 + k^H \cdot |C| \cdot \log N)). \tag{4}$$

Lastly, we consider (A3). For a given term $\tau \in T_M(C)$, define a finite set $D_i$ as follows:

$$\begin{cases} D_1 = \{t | \tau \underset{A,S}{\Rightarrow} t\}, \\ D_i = \{t' | t \in D_{i-1}, t \underset{A,S}{\Rightarrow} t'\}. \end{cases}$$

```
B12   for each t'' in T_A
B13       if t'' is a subterm of t at r'' then
B14           for each c'' in Z(t'')
B15               let t' be t[r'' ← c'']
```

Fig. 6: Procedure to compute $t \underset{A,S}{\Rightarrow} t'$.

We first compute $D_1$, and then $D_2, D_3, \ldots$ until $D_j = D_{j+1}$. It holds that $j \leq l$, since $\|t\| > \|t'\|$ for each pair $t, t'$ such that $t \underset{A,S}{\triangleright} t' \in P_S$. The size of each $D_i$ is at most $|X_l|$. Fig. 6 shows a procedure to compute $t \underset{A,S}{\Rightarrow} t'$ for $t, t' \in T_M(C)$. We assume that it takes constant time to execute (B15). Executing (B12) through (B15) takes

$$\mathcal{O}(|T_A| \cdot (t_A + l \cdot H + t_Z + |C|)) = \mathcal{O}(N \cdot |C|^{k^H} \cdot (k^H \cdot \log N + l \cdot H + |C|))$$

time. Since the procedure in Fig. 6 is executed for each term in $D_i$ $(1 \leq i \leq j)$, the time complexity to execute (A3) is

$$\begin{aligned} &\mathcal{O}(j \cdot |X_l| \cdot N \cdot |C|^{k^H} \cdot (k^H \cdot \log N + l \cdot H + |C|)) \\ &= \begin{cases} \mathcal{O}(l \cdot |C|^{l+k^H} \cdot N \cdot (k^H \cdot \log N + l \cdot H + |C|)) & \text{if } k \geq 2, \\ \mathcal{O}(l^2 \cdot |C|^2 \cdot N \cdot (\log N + l \cdot H + |C|)) & \text{if } k = 1. \end{cases} \end{aligned} \tag{5}$$

After all, by letting $L = \max\{l, H\}$ in Eq. (2) through Eq. (5), the time complexity of the algorithm is

$$\begin{cases} \mathcal{O}(|C|^{2k^L} \cdot N \cdot (N \cdot L^2 + (|C| \cdot N + L) \cdot k^L \cdot \log N + L^3 + L \cdot |C|)) & \text{if } k \geq 2, \\ \mathcal{O}(|C|^2 \cdot N \cdot (N \cdot L^2 + |C| \cdot N \cdot \log N + L^4)) & \text{if } k = 1. \end{cases}$$

Moreover, by assuming $N \geq L$, the time complexity is

$$\begin{cases} \mathcal{O}(|C|^{2k^L} \cdot N^2 \cdot (L^2 + k^L \cdot |C| \cdot \log N)) & \text{if } k \geq 2, \\ \mathcal{O}(|C|^2 \cdot N^2 \cdot (L^3 + |C| \cdot \log N)) & \text{if } k = 1. \end{cases}$$

**Theorem 4:** The sufficient condition stated in Theorem 2 is decidable. □

**Corollary 1:** For a monadic method schema, the detection problem of security flaws is solvable in polynomial time of the size of the schema. □

# 5 Conclusions

This paper has discussed the detection problem of security flaws for database schemas, and has shown that the detection problem for general (i.e., non-monadic) method schemas is undecidable. We have proposed a decidable sufficient condition for a given general schema to have no

security flaw, and have shown that the sufficient condition is also a necessary one if a schema is monadic. Also, we have provided an algorithm to decide the sufficient condition, and have shown that the algorithm runs in polynomial time for a monadic method schema.

As future work, we intend to decrease the complexity to detect a security flaw by making the algorithm more efficient. Also, we intend to discuss the detection problem when, in the discussion of Sect. 3.2, a user knows either what $O_{S,I}$ is or what $C$ is, or the user knows both of them. Moreover, for a general schema, if $Z$ satisfies that for each $t \in T_M(C)$,

$$Z(t) = \{c | \text{ there exists an interpretation such that } t[\vec{o}/\vec{c}] \xrightarrow{*} o, \vec{o} \in \nu(\vec{c}), o \in \nu(c)\},$$

then whether or not the sufficient condition in Theorem 2 is also a necessary one is open.

# Acknowledgment

The authors would like to thank Professor Hiroyuki Seki of Nara Institute of Science and Technology for his invaluable suggestions and comments.

# References

[1] S. Abiteboul, R. Hull, and V. Vianu, "Foundations of databases," pp.563–571, Addison-Wesley Publishing Company, 1995.

[2] S. Abiteboul, P. Kanellakis, S. Ramaswamy, and E. Waller, "Method schemas," J. Computer and System Sci., vol.51, no.3, pp.433–455, 1995.

[3] L.J. Binns, "Implementation considerations for inference detection: intended vs. actual classification," Database Security, VII(A-47): Status and Prospects, Elsevier Science Publishers, pp.139–156, 1994.

[4] E. Bertino and H. Weigand, "An approach to authorization modeling in object-oriented database systems," Data and Knowledge Engineering, vol.12, no.1, pp.1–29, 1994.

[5] H.H. Brüggemann, "Object-oriented authorization," Advances in Database Systems – Implementations and Applications, CISM 347, Springer-Verlag, pp.139–160, 1994.

[6] N. Dershowitz and J. Jouannaud, "Rewrite systems," in Handbook of Theoretical Computer Science, ed. J. Leeuwen, vol.B, chap.6, pp.243–320, The MIT Press, 1990.

[7] E.B. Fernandez, M.M. Larrondo-Petrie, and E. Gudes, "A method-based authorization model for object-oriented databases," Proc. OOPSLA-93 Conf. Workshop on Security for Object-Oriented Systems, pp.135–150, 1993.

[8] T.D. Garvey and T.F. Lunt, "Cover stories for database security," Database Security, V: Status and Prospects, Elsevier Science Publishers, pp.363–380, 1992.

[9] J. Hale, J. Threet, and S. Shenoi, "A practical formalism for imprecise inference control," Database Security, VIII(A-60): Status and Prospects, Elsevier Science Publishers, pp.139–156, 1994.

[10] T.H. Hinke, H.S. Delugach, and R. Wolf, "A framework for inference-directed data mining," Database Security, X: Status and Prospects, Chapman & Hall, pp.229–239, 1996.

[11] Y. Ishihara, S. Shimizu, H. Seki, and M. Ito, "The type-consistency problem for queries in object-oriented databases," NAIST Technical Report 98004, http://isw3.aist-nara.ac.jp/IS/TechReport2/report/98004.ps, Apr. 1998.

[12] T. Morita, Y. Ishihara, H. Seki, and M. Ito, "An authorization model for object-oriented databases and its efficient access control," IEICE Transactions on Information and Systems (to appear).

[13] D.A. Plaisted, "Equational reasoning and term rewriting systems," in Handbook of Logic in Artificial Intelligence and Logic Programming, vol.1, pp.273–364, Oxford Science Publications, 1993.

[14] H. Seki, Y. Ishihara, and H. Dodo, "Testing type consistency of method schemas," IEICE Transactions on Information and Systems, vol.E81-D, no.3, March 1998.

[15] H. Seki, Y. Ishihara, and M. Ito, "Authorization analysis of queries in object-oriented databases," Proc. 4th Int'l Conf. on Deductive and Object-Oriented Databases, LNCS 1013, pp.521–538, 1995.

[16] K. Tajima, "Static detection of security flaws in object-oriented databases," Proc. 15th ACM PODS, pp.341–352, 1996.

[17] B. Thuraisingham, "The use of conceptual structures for handling the inference problem," Database Security, V: Status and Prospects, Elsevier Science Publishers, pp.333–362, 1992.