The Security Problem against Inference Attacks on Object-Oriented Databases

Yasunori ISHIHARA† Toshiyuki MORITA‡† Minoru ITO†

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

‡ Systems Development Laboratory
Hitachi, Ltd.
890 Kashimada, Saiwai, Kawasaki, Kanagawa, 211-8567 Japan

Abstract

Inference attacks mean that a user infers (or tries to infer) the result of an unauthorized method execution using only authorized methods to the user. We say that a method m is secure against inference attacks by a user u if there exists no database instance for which u can infer the result of m. It is important for database administrators to know which methods are secure and which ones are not. When an administrator finds that a method which retrieves top secret information is not secure against inference attacks by u, the administrator can prevent u from attacking the method by changing the authorization for u. This paper formalizes the security problem (i.e., to determine whether a given method is secure or not) for method schemas, and presents the following results. First, it is shown that the security problem is undecidable. Next, a decidable sufficient condition for a given method to be secure is proposed. Furthermore, it is shown that the sufficient condition is also a necessary one if a given schema is monadic (i.e., every method has exactly one parameter). The time complexity to decide the condition is also evaluated. For a monadic schema, the condition is decidable (and therefore, the security problem is solvable) in polynomial time of the size of the schema.

1 Introduction

In recent years, various authorization models for object-oriented databases (OODBs) have been proposed and studied. Among of them, the method-based authorization model [5, 13] 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, an authorization A for a user u can be represented as a set of expressions $m(c_1, \ldots, c_n)$, which means that u can *directly* invoke method m on any tuple $(o_1, \ldots, o_n)$ of objects such that o_i is an object of class c_i $(1 \leq i \leq n)$. On the other hand, even if $m(c_1, \ldots, c_n) \notin A$, u can invoke m *indirectly* through another method execution in several models (e.g., protection mode in [3]). Although such indirect invocations are useful for data hiding [3], they may also allow *inference attacks* in some situations.

Example 1: Let Employee, Host, and Room be classes representing employees, hosts, and rooms, respectively. Suppose that a method uses returns the host which a given employee uses, and a method located returns the room in which a given host is placed. Also suppose that a method office, which returns the room occupied by a given employee, is implemented as office(x) = located(uses(x)).

Now suppose that the physical computer network is top secret information. In this case, an authorization for a user u may be the one shown in Fig. 1, where a solid (resp. dotted) arrow denotes an authorized (resp. unauthorized) method to u. Suppose that u have obtained that uses(John) = mars and office(John) = A626 using the authorized methods. Also suppose that u knows the implementation body of office as its behavioral specification. Then, u can infer that located(mars) = A626.

On the other hand, suppose that method uses retrieves top secret information and therefore the authorization of u is set as shown in Fig. 2. Then, u knows that located(mars) = A626, office(John) = A626, and office(x) = located(uses(x)), similarly to the former case. However, u cannot conclude that uses(John) = mars only from the above information, since there may be another host h such that uses(John) = h and located(h) = A626. □

For a given database schema S and an authorization A for a user u, an n-ary method m is said to be *secure* at $(c_1, \ldots, c_n)$ (each c_i is a class in S) against inference attacks by u if u cannot infer the result of $m(o_1, \ldots, o_n)$ for any objects o_i of class c_i in any database instance I of S, using only authorized methods to u. Otherwise, m is *insecure*. For example, if uses(Employee) and

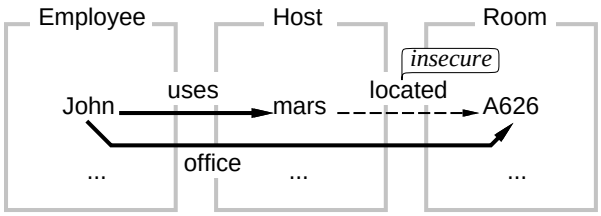

Fig. 1: An example of an insecure method.

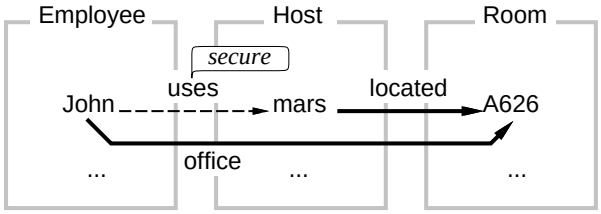

Fig. 2: An example of a secure method.

office(Employee) are authorized, then located is insecure since the user can infer located(mars) = A626 under the database instance shown in Fig. 1. On the other hand, it will be shown later that uses is secure when only located(Host) and office(Employee) are authorized. It is important for database administrators to know which methods are secure and which ones are not. When an administrator finds that a method which retrieves top secret information is insecure against inference attacks by u, the administrator can prevent u from attacking the method by changing the authorization for u.

In this paper, we formally define the security problem, i.e., to determine whether a given method is secure or not. We adopt method schemas proposed by [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. Under this formalization, we first show that the problem is undecidable. Next, we propose a decidable sufficient condition for a method to be secure. Furthermore, we show that the sufficient condition is also a necessary one if a given schema is monadic (i.e., every method has exactly one parameter). Finally, we evaluate the time complexity of deciding the condition. For a monadic method schema, the proposed condition is decidable (and therefore, the security problem is solvable) in polynomial time of the size of the schema.

The main idea of the proposed sufficient condition is to "conservatively" approximate the user's inference. The user's inference is object-level, while the approximation is class-level inference. To do the class-level inference, we use a static type checking technique for method schemas. Let m be an n-ary method and $c_1,\ldots, c_n$ be classes. The most difficult task in type checking is to solve the following question:

> Suppose that m is executed on arbitrary objects $o_1,\ldots, o_n$ such that o_i belongs to class c_i. What class does the object obtained by the execution belong to?

Unfortunately, this question is unsolvable in general [2]. However, the type checking algorithm proposed in [12] can compute a set of classes which contain all the correct classes, although the set may contain some wrong classes. Using this algorithm, we can conservatively approximate user's inference.

In this paper we discuss *precise inference* in OODBs. Precise inference means that a user can infer (or, is interested in) only the exact value of the result of an unauthorized method. On the other hand, most of the recent researches concentrate on *imprecise inference* in relational databases, not OODBs. Imprecise inference means that a user can infer (or, is interested in) possible values of the result of an unauthorized method (query) with a certain probability. In [6], FD-based imprecise inference involving abduction and partial deduction is discussed. In [15], a quantitative measure of inference risk is formally defined. Imprecise inference with external, common sense knowledge can be regarded as data mining [7, 11].

[14] focuses on both precise and imprecise inference in OODBs. Besides inferability of the result of a method execution, the article introduces the notion of controllability, which means that a user can control (alter arbitrarily) an attribute-value of an object in a database instance. We do not consider controllability since our query language does not support update operations for database instances. However, since our query language supports recursion while the one in [14] does not, detecting inferability in our formalization is not trivial. [14] also proposes, for a given database schema S and an authorization A, a sound algorithm for detecting inferability or controllability. However, [14] does not evaluate the complexity of the algorithm.

This paper is organized as follows. In Section 2, we give the definition of method schemas. In Section 3, we discuss inference attacks and formulate the security problem. We also show that the problem is undecidable. In Section 4, we propose a sufficient condition for a method to be secure. Finally, in Section 5, we conclude this paper.

2 Method Schemas

2.1 Syntax

We introduce some notations before defining the syntax of method schemas. Let F be a family of disjoint sets F_0, F_1, F_2,..., 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 the terms freely generated by F and X. For a set V, let V^n denote the Cartesian product $\underbrace{V \times \cdots \times V}_{n}$. For a term $t \in T_F(X)$, an n-tuple $\mathbf{t} = (t_1, \ldots, t_n) \in (T_F(X))^n$ of terms, and an n-tuple $\mathbf{x} = (x_1, \ldots, x_n) \in X^n$ of variables, let $t[\mathbf{t}/\mathbf{x}]$ denote the term obtained by simultaneously replacing every x_i in t with t_i ($1 \leq i \leq n$). For example, $f(x_1, g(x_1, x_2))[(f(a), x_1)/(x_1, x_2)] = f(f(a), g(f(a), x_1))$. For a term t, define the set of *occurrences* $R(t)$ as the smallest set of sequences of positive integers with the following two properties:

- $\varepsilon \in R(t)$, where ε is the empty sequence.
- If $r \in R(t_i)$, then $i \cdot r \in R(f(t_1, \ldots, t_n))$ ($1 \leq i \leq n$), where the center dot "$\cdot$" represents the concatenation of sequences.

An occurrence of t specifies (the position of) a subterm of t. For example, $1 \cdot 2$ of $f(f(x, g(x)), g(x))$ specifies the first $g(x)$. The replacement in t of t' at r, denoted $t[r \leftarrow t']$, is defined as follows:

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

For example,

$$f(f(x, g(x)), g(x))[1 \cdot 2 \leftarrow h(a, b)] = f(f(x, h(a, b)), g(x)).$$

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 mutually disjoint finite sets M_0, M_1, M_2,..., where M_n ($n = 0, 1, 2, \ldots$) is a set of *method names* of arity n. Each M_n is partitioned into $M_{\mathrm{b},n}$ and $M_{\mathrm{c},n}$: Each $m_\mathrm{b} \in M_\mathrm{b}$ ($= \bigcup_{n \geq 0} M_{\mathrm{b},n}$) (resp. $m_\mathrm{c} \in M_\mathrm{c}$ ($= \bigcup_{n \geq 0} M_{\mathrm{c},n}$)) is called a *base method name* (resp. *composite method name*). Furthermore, each $m \in M$ ($= M_\mathrm{b} \cup M_\mathrm{c}$) is simply called a *method name*. We say that M is a *method signature*.

Hereafter, we often use a bold letter $\mathbf{v}$ to mean $(v_1, \ldots, v_n)$ without explicitly defining it when n is irrelevant or obvious from the context.

Definition 1: Let $\mathbf{c} \in C^n$. A *base method definition* of $m_\mathrm{b} \in M_{\mathrm{b},n}$ at $\mathbf{c}$ is a pair $(m_\mathrm{b}(\mathbf{c}), c)$, where $c \in C$. A *composite method definition* of $m_\mathrm{c} \in M_{\mathrm{c},n}$ at $\mathbf{c}$ is a pair $(m_\mathrm{c}(\mathbf{c}), t)$, where $t \in T_M(\{x_1, \ldots, x_n\})$. □

Let o_i be an object of class c_i ($1 \leq i \leq n$) (see Defs. 2 and 4 for formal definitions). Informally, the above base method definition declares that the application of m_b to $\mathbf{o} = (o_1, \ldots, o_n)$ results in an object of c or its subclass, while the above composite method definition states that the application of m_c to $\mathbf{o}$ results in term rewriting starting from $t[\mathbf{o}/\mathbf{x}]$. The formal definition is presented in Section 2.2.

Definition 2: A *method schema* [1, 2] S is a 5-tuple $(C, \leq, M, \Sigma_\mathrm{b}, \Sigma_\mathrm{c})$, where:

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'. We naturally extend $\leq$ to n-tuples of classes as follows: For two tuples $\mathbf{c} = (c_1, \ldots, c_n)$ and $\mathbf{c}' = (c'_1, \ldots, c'_n)$, we write $\mathbf{c} \leq \mathbf{c}'$ *iff* $c_i \leq c'_i$ for all i.
3. M is a method signature.
4. Σ_b is a set of base method definitions.
5. Σ_c is a set of composite method definitions.

For each possible combination $\mathbf{c} \in C^n$ and $m \in M_n$, there must exist at most one method definition of m at $\mathbf{c}$. If every method of S has exactly one parameter, then S is *monadic*. □

Example 2: An example of a method schema S_1 is shown in Fig. 3. Manager is a subclass of Employee, and Server is a subclass of Host. Method boss(e) returns the direct boss of employee e, and method supervisor(e) returns the "second least manager" among the indirect bosses of e. Since every method has arity one, S_1 is monadic. □

2.2 Semantics

Method definitions are inherited along the class hierarchy.

Definition 3: Let $S = (C, \leq, M, \Sigma_\mathrm{b}, \Sigma_\mathrm{c})$, $m_\mathrm{b} \in M_{\mathrm{b},n}$, and $\mathbf{c} \in C^n$. Suppose that $(m_\mathrm{b}(\mathbf{c}'), c') \in \Sigma_\mathrm{b}$ is the base method definition of m_b at the smallest $\mathbf{c}'$ above $\mathbf{c}$, i.e., whenever $(m_\mathrm{b}(\mathbf{c}''), c'') \in \Sigma_\mathrm{b}$ and $\mathbf{c} \leq \mathbf{c}''$, it is the case that $\mathbf{c}' \leq \mathbf{c}''$. We define the *resolution* $Res(m_\mathrm{b}(\mathbf{c}))$ of m_b at $\mathbf{c}$ as c'. If such a unique base method definition does not exist, then $Res(m_\mathrm{b}(\mathbf{c}))$ is *undefined*, denoted $\perp$. The resolution of a composite method is defined in the same way. □

C = {Employee, Manager, Host, Server, Room}
Manager $\leq$ Employee, Server $\leq$ Host
M = {boss, uses, located, supervisor, office}
Σ_b = {(boss(Employee), Employee),
(boss(Manager), Manager),
(uses(Employee), Host),
(located(Host), Room)}
Σ_c = {(supervisor(Employee), supervisor(boss(x_1))),
(supervisor(Manager), boss(x_1)),
(office(Employee), located(uses(x_1)))}

Fig. 3: A method schema S_1.

Example 3: Consider schema S_1 shown in Fig. 3. By Def. 3, *Res*(located(Server)) = Room. In other words, class Server inherits method definition (located(Host), Room) $\in \Sigma_b$. On the other hand, *Res*(boss(Server)) = $\perp$ since no superclass of Server has a definition of boss. □

The semantics of a method schema 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 composite method is defined by the interpretation of base methods and term rewriting.

Definition 4: An *interpretation* (or, also called a *database instance*) of a method schema S is a pair $I = (\nu, \mu)$ with the following properties:

1. To each $c \in C$, ν assigns a finite disjoint set $\nu(c)$ of *object identifiers* (or simply, *objects*). Each $o \in \nu(c)$ is called an object of class c. Let $O_I = \bigcup_{c \in C} \nu(c)$. For $\mathbf{c} = (c_1, \ldots, c_n)$, let $\nu(\mathbf{c})$ denote $\nu(c_1) \times \cdots \times \nu(c_n)$.
2. For each $m_b \in M_{b,n}$, $\mu(m_b)$ is a partial mapping from O_I^n to O_I which satisfies the following two conditions. Let $\mathbf{c}, \mathbf{c}' \in C^n$.
   (a) If *Res*$(m_b(\mathbf{c})) = c'$, then $\mu(m_b)\,|_{\nu(\mathbf{c})}$ is a total mapping to $\bigcup_{c \leq c'} \nu(c)$, where "|" denotes that the domain of μ is restricted to $\nu(\mathbf{c})$.
   (b) If *Res*$(m_b(\mathbf{c})) = \perp$, then $\mu(m_b)$ is undefined everywhere in $\nu(\mathbf{c})$. □

A term in $T_M(O_I)$ is called an *instantiated term*. That is, an instantiated term consists of method names in M and objects in O_I. The *one-step execution relation* $\rightarrow_I$ on the instantiated terms, based on the leftmost innermost reduction strategy, is defined as follows:

Definition 5: Let $m(\mathbf{o})$ ($\mathbf{o} \in \nu(\mathbf{c})$) be the subterm of $t \in T_M(O_I)$ at the leftmost innermost occurrence r.

1. If $m \in M_b$ and *Res*$(m(\mathbf{c})) \neq \perp$, then
   $t \rightarrow_I t[r \leftarrow \mu(m)(\mathbf{o})]$.
2. If $m \in M_c$ and *Res*$(m(\mathbf{c})) = t' \neq \perp$, then
   $t \rightarrow_I t[r \leftarrow t'[\mathbf{o}/\mathbf{x}]]$. □

Note that, by Def. 5, for any instantiated term t, there exists at most one term t' such that $t \rightarrow_I t'$. That is, every execution is deterministic.

Let $\rightarrow_I^*$ be the reflexive and transitive closure of $\rightarrow_I$. If $t \rightarrow_I^* t'$ and there exists no t'' such that $t' \rightarrow_I t''$, then t' is called the *execution result* of t, and we write $t{\downarrow} = t'$. If $t{\downarrow} \in O_I$, then the execution of t is *successful*, and if $t{\downarrow} \notin O_I$ 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 omit the subscript I and simply write $\rightarrow$ or $\rightarrow^*$ if I is understood from the context.

Example 4: An example of an interpretation $I_1 = (\nu_1, \mu_1)$ of S_1 is shown in Fig. 4. ν_1 is represented by gray rectangles, e.g., ν_1(Employee) = {Alice, John}. μ_1 is represented by arrows, e.g., μ_1(boss)(John) = Alice, μ_1(uses)(John) = mars. By Def. 5, supervisor(Alice) is executed as follows:

supervisor(Alice) $\rightarrow_{I_1}$ supervisor(boss(Alice))
$\rightarrow_{I_1}$ supervisor(Sara)
$\rightarrow_{I_1}$ boss(Sara)
$\rightarrow_{I_1}$ Bob.

Thus supervisor(Alice)$\downarrow$ = Bob. □

3 Inference Attacks

3.1 Authorization

Various sophisticated method-based authorization models for OODBs have been proposed. In this paper, however, discussing authorization models is not our main purpose, and therefore we adopt the following simple but general authorization model.

Definition 6: Let $S = (C, \leq, M, \Sigma_b, \Sigma_c)$. An *authorization* A for a user u under S is a finite set of $m(\mathbf{c})$, where $m \in M_n$ and $\mathbf{c} \in C^n$. Intuitively, $m(\mathbf{c}) \in A$ means that u is authorized to directly invoke method m on any tuple $\mathbf{o}$ of objects such that $\mathbf{o} \in \nu(\mathbf{c})$. □

An authorization is often modeled as a pair of a base authorization and a set of inference rules. An example of an

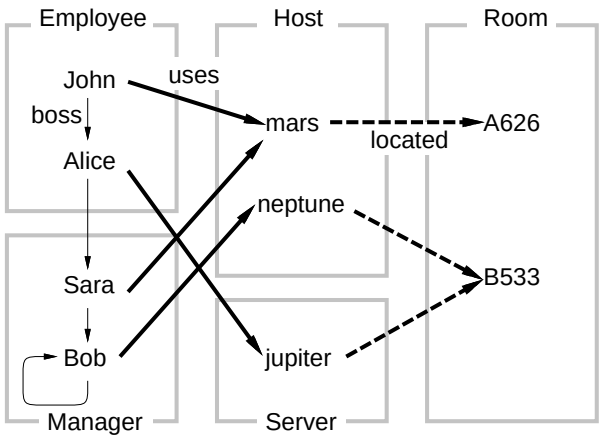

Fig. 4: An interpretation I_1 of S_1.

inference rule is "if u is authorized to invoke m on objects of c, then u is also authorized to invoke m on objects of the subclasses of c." When $c_1 \leq c$ and $c_2 \leq c$, the base authorization $\{m(c)\}$ is expanded into $\{m(c), m(c_1), m(c_2)\}$ by this rule. In this paper, we assume that a given authorization has already been expanded.

Example 5: Define an authorization A_1 for a user u under S_1 in Fig. 3 as follows:

$$A_1 = \{\mathsf{uses(Employee)}, \mathsf{supervisor(Employee)}, \mathsf{supervisor(Manager)}, \mathsf{office(Employee)}, \mathsf{office(Manager)}\}.$$

Consider the interpretation I_1 in Fig. 4. Executing office(John) by u is permitted since John $\in \nu_1$(Employee) and office(Employee) $\in A_1$. On the other hand, executing uses(Sara) is prohibited since Sara $\in \nu_1$(Manager) but uses(Manager) $\notin A_1$. □

3.2 Formal Definition of User's Inference

In this paper, information provided by a database is modeled as a set of (in)equalities. For example, suppose that a user u executes office(John) and obtains the result A626. In this case, the information that u obtains is office(John)$\downarrow$ = A626. In what follows, we demonstrate that user's inference can be formalized as the congruence closure of a finite set of ground equalities when the two reasonable conditions (Q1) and (Q2) stated below are satisfied.

First of all, we define the information which u can obtain directly from a database instance $I = (\nu, \mu)$.

($*1$) User u obtains $m(\mathbf{o})\downarrow = o$ *iff* it is the case that $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, and $m(\mathbf{o})\downarrow = o \in O_I$. That is, u knows what the result of $m(\mathbf{o})$ is if executing $m(\mathbf{o})$ is authorized and the execution is successful.

($*2$) User u obtains $Res(m(\mathbf{c})) = t$ *iff* it is the case that $m(\mathbf{c}) \in A$ and $Res(m(\mathbf{c})) = t$. That is, u knows the type declaration of m at $\mathbf{c}$ (when m is a base method) or the behavioral specification of m at $\mathbf{c}$ (when m is a composite method), if executing $m(\mathbf{o})$ ($\mathbf{o} \in \nu(\mathbf{c})$) is authorized.

In Example 1, ($*1$) and ($*2$) are stated informally.

Next, suppose that u can use at least four inference rules: reflexivity, symmetry, transitivity, and substitutivity (i.e., if $t_i = t_i'$ for all i, then $f(\mathbf{t}) = f(\mathbf{t}')$). Also suppose that user u knows that $o \neq o'$ for distinct objects o and o' (e.g., u knows John $\neq$ Alice, Sara $\neq$ A626, and so on).

As stated in Section 1, the goal of inference attacks is to obtain o such that $m(\mathbf{o})\downarrow = o$ for some m and $\mathbf{o}$. In other words, u wants to infer equalities. Therefore, if we find a reasonable condition under which inequalities are useless to infer equalities, we can formalize user's inference as the congruence closure of equalities induced by ($*1$) and ($*2$). Let us examine the following example:

Example 6: Recall the second case of Example 1, where u cannot infer uses(John)$\downarrow$ = mars since there may be another host h such that uses(John)$\downarrow = h$ and located(h)$\downarrow$ = A626. However, if u knows that located(o)$\downarrow \neq$ A626 for *any other object o in the database instance*, then u can conclude that uses(John) = mars. □

The above example suggests that inequalities are useless to infer equalities if the following condition is satisfied:

(Q1) User u does not know what O_I is.

In many cases, this condition is satisfied by just hiding O_I from the user.

Equalities obtained by ($*2$) are not ground (i.e., include variables). However, together with the following condition (Q2), they are equivalent to a finite set of ground equalities, which has many good properties:

(Q2) The user does not know what C is.

This condition is also satisfied by just hiding C.

Example 7: Consider a schema with a composite method m_c which has the same resolution t at every class $c \in C$. Let $A = \{m_c(c) \mid c \in C\}$ be an authorization for a user u.

Assume that u knows what C is. Then, u can infer that $m_c(t')\downarrow = t[t'/x]\downarrow$ for any term t' such that $t'\downarrow \in O_I$, since

m_c has the same resolution t at any class. Note that, in this inference, u does not need to know which class $t'{\downarrow}$ belongs to.

On the other hand, if (Q2) is satisfied, then u cannot conclude that $m_c(t'){\downarrow} = t[t'/x]{\downarrow}$ without exactly inferring the class to which $t'{\downarrow}$ belongs since there may be another class c in C such that $t'{\downarrow} \in \nu(c)$ and $Res(m_c(c)) \neq t$. □

Type checking [2, 12] is useless when (Q2) is satisfied. Therefore, to know the class to which $t'{\downarrow}$ belongs is to infer the exact value of $t'{\downarrow}$. Thus, the equalities obtained by ($*2$) can be applied only to terms t' such that $t'{\downarrow}$ is known. This means that each equality $Res(m_c(\mathbf{c})) = t$ obtained by ($*2$) can be regarded as $\{m_c(\mathbf{o}){\downarrow} = t[\mathbf{o}/\mathbf{x}]{\downarrow} \mid \mathbf{o} \in \nu(\mathbf{c})\}$, which is a finite set of ground equalities.

Consequently, by assuming (Q1) and (Q2), we can model user's inference as the congruence closure of a finite set of ground equalities induced by ($*1$) and ($*2$). For technical reasons, we define the congruence closure through rewriting rules $\rhd_{I,A}$ introduced below. From the correctness of Knuth-Bendix completion [10], $t{\downarrow} = o$ *iff* t is reducible to o by $\rhd_{I,A}$.

Definition 7: Define $P_{I,A}$ as the minimum set of rewriting rules $\rhd_{I,A}$ on $T_M(O_I)$ satisfying the following three conditions. Intuitively, $t \rhd_{I,A} o$ means that the user knows or can infer that $t{\downarrow} = o$.

(A) If $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, and $m(\mathbf{o}){\downarrow} = o \in O_I$, then $P_{I,A}$ contains

$$m(\mathbf{o}) \rhd_{I,A} o.$$

This corresponds to ($*1$).

(B) If $m_c(\mathbf{c}) \in A, m_c \in M_c, \mathbf{o} \in \nu(\mathbf{c}), m_c(\mathbf{o}){\downarrow} = o \in O_I$, and $Res(m_c(\mathbf{c})) = t \neq \bot$, then $P_{I,A}$ contains

$$t[\mathbf{o}/\mathbf{x}] \rhd_{I,A} o.$$

This essentially corresponds to ($*2$).

(C) If $P_{I,A}$ contains $t \rhd_{I,A} o$ and $t'' \rhd_{I,A} o''$ such that t'' is a proper subterm of t at r'', then $P_{I,A}$ contains

$$t[r'' \leftarrow o''] \rhd_{I,A} o.$$

See also Fig. 5. This simulates Knuth-Bendix completion procedure.

By definition, the right-hand side of each rule is an object. Note that the existence of $t \rhd_{I,A} o$ in $P_{I,A}$ implies $t \rightarrow_I^* o$.

Define $\Rightarrow_{I,A}$ as the one-step reduction relation by $\rhd_{I,A}$. That is, $t \Rightarrow_{I,A} t'$ *iff* there exists a subterm t'' of t at r'' such that $t'' \rhd_{I,A} o'' \in P_{I,A}$ and $t' = t[r'' \leftarrow o'']$. Let

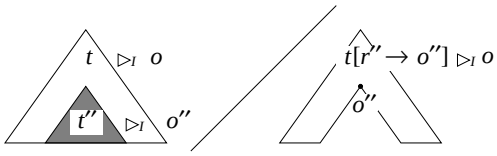

Fig. 5: Condition (C) of Definition 7.

$\Rightarrow_{I,A}^*$ denote the reflexive and transitive closure of $\Rightarrow_{I,A}$. For readability, we often write $\rhd_I$ and P_I instead of $\rhd_{I,A}$ and $P_{I,A}$, respectively. □

Example 8: For S_1 in Fig. 3, I_1 in Fig. 4, and A_1 in Example 5, P_{I_1} is computed as shown in Fig. 6. Rules (A1)–(A10) are obtained by Def. 7(A), and (B1)–(B8) by Def. 7(B) with composite methods supervisor and office. Rules (C1) and (C2) are obtained by Def. 7(C). For example, (C1) is derived from (A1) and (B5).

Rule (C1) indicates that the user can infer that located(mars)↓ = A626, as stated in the first case of Example 1. Moreover, rule (C2) indicates that located even for a server jupiter can be inferred. Let τ = office(boss(Sara)) and τ' = office(boss(John)). Then,

$$\tau \Rightarrow_{I_1} \text{office(Bob)} \Rightarrow_{I_1} \text{B533}.$$

Thus user u can infer that $\tau{\downarrow}$ = B533. On the other hand, u cannot infer the value of $\tau'{\downarrow}$ (although $\tau'{\downarrow}$ = B533) since no subterm of τ' can be rewritten by the rules in P_{I_1}. □

3.3 The Security Problem

The notion of security of methods discussed in Section 1 is naturally extended to terms in $T_M(C)$ as follows: A term $\tau \in T_M(C)$ is said to be *secure* if there exists no interpretation $I = (\nu, \mu)$ such that $\tau[\mathbf{o}/\mathbf{c}] \Rightarrow_{I,A}^* o$ for any $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in O_I$. Otherwise, τ is *insecure*. The *security problem* is to determine whether a given $\tau \in T_M(C)$ is secure or not.

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

Sketch of Proof: The *type-consistency problem* is to determine whether, for a given method schema S, there exists an interpretation of S which causes an aborted execution. [8] shows that the 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

$$
\begin{array}{rcll}
\mathsf{uses(John)} & \rhd_{I_1} & \mathsf{mars} & \text{(A1)} \\
\mathsf{uses(Alice)} & \rhd_{I_1} & \mathsf{jupiter} & \text{(A2)} \\
\mathsf{supervisor(John)} & \rhd_{I_1} & \mathsf{Bob} & \text{(A3)} \\
\mathsf{supervisor(Alice)} & \rhd_{I_1} & \mathsf{Bob} & \text{(A4)} \\
\mathsf{supervisor(Sara)} & \rhd_{I_1} & \mathsf{Bob} & \text{(A5)} \\
\mathsf{supervisor(Bob)} & \rhd_{I_1} & \mathsf{Bob} & \text{(A6)} \\
\mathsf{office(John)} & \rhd_{I_1} & \mathsf{A626} & \text{(A7)} \\
\mathsf{office(Alice)} & \rhd_{I_1} & \mathsf{B533} & \text{(A8)} \\
\mathsf{office(Sara)} & \rhd_{I_1} & \mathsf{A626} & \text{(A9)} \\
\mathsf{office(Bob)} & \rhd_{I_1} & \mathsf{B533} & \text{(A10)} \\
 & & & \\
\mathsf{supervisor(boss(John))} & \rhd_{I_1} & \mathsf{Bob} & \text{(B1)} \\
\mathsf{supervisor(boss(Alice))} & \rhd_{I_1} & \mathsf{Bob} & \text{(B2)} \\
\mathsf{boss(Sara)} & \rhd_{I_1} & \mathsf{Bob} & \text{(B3)} \\
\mathsf{boss(Bob)} & \rhd_{I_1} & \mathsf{Bob} & \text{(B4)} \\
\mathsf{located(uses(John))} & \rhd_{I_1} & \mathsf{A626} & \text{(B5)} \\
\mathsf{located(uses(Alice))} & \rhd_{I_1} & \mathsf{B533} & \text{(B6)} \\
\mathsf{located(uses(Sara))} & \rhd_{I_1} & \mathsf{A626} & \text{(B7)} \\
\mathsf{located(uses(Bob))} & \rhd_{I_1} & \mathsf{B533} & \text{(B8)} \\
 & & & \\
\mathsf{located(mars)} & \rhd_{I_1} & \mathsf{A626} & \text{(C1)} \\
\mathsf{located(jupiter)} & \rhd_{I_1} & \mathsf{B533} & \text{(C2)}
\end{array}
$$

Fig. 6: Contents of P_{I_1}.

of a term, say $m(o)$, is aborted under I. Otherwise, $m(o)$ is nonterminating. By slightly modifying the reduction in [8], we can construct a schema with the following properties:

- 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. Let $A = \{m'(c)\}$ and $\tau = m'(c)$. Then, the PCP has a solution *iff* there exists $I = (\nu, \mu)$ such that $\tau[o/c] \Rightarrow_I^* o'$ 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 term $\tau \in T_M(C)$ to be secure. The main idea is to introduce new rewriting rules on $T_M(C)$ which "conservatively" approximate $\rhd_{I,A}$, i.e., if τ is insecure, then τ is reducible to a class c by the new rewriting rules. Intuitively, each $t \in T_M(C)$ is considered as the set of instantiated terms $\{t[\mathbf{o}/\mathbf{c}] \mid \mathbf{o} \in \nu(\mathbf{c})\}$. The "execution result" $E(t)$ of t is defined as follows: $c \in E(t)$ *iff* there exists an interpretation $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{c}]{\downarrow} \in \nu(c)$ for some $\mathbf{o} \in \nu(\mathbf{c})$. Unfortunately, we cannot compute E exactly in general [2]. However, we can compute $Z : T_M(C) \to 2^C$ such that $Z(t) \supseteq E(t)$ for every $t \in T_M(C)$ [12]. We use such Z to approximate $\rhd_{I,A}$. The smaller $Z(t)$ is, the better approximation we have, although the approximation is still conservative even when $Z(t) = C$ for all t. The algorithm in [12] gives a fairly small Z.

$$
\begin{aligned}
Z(\mathsf{boss(Employee)}) &= \{\mathsf{Employee}, \mathsf{Manager}\} \\
Z(\mathsf{boss(Manager)}) &= \{\mathsf{Manager}\} \\
Z(\mathsf{uses(Employee)}) &= \{\mathsf{Host}, \mathsf{Server}\} \\
Z(\mathsf{uses(Manager)}) &= \{\mathsf{Host}, \mathsf{Server}\} \\
Z(\mathsf{located(Host)}) &= \{\mathsf{Room}\} \\
Z(\mathsf{located(Server)}) &= \{\mathsf{Room}\} \\
Z(\mathsf{supervisor(Employee)}) &= \{\mathsf{Manager}\} \\
Z(\mathsf{supervisor(Manager)}) &= \{\mathsf{Manager}\} \\
Z(\mathsf{office(Employee)}) &= \{\mathsf{Room}\} \\
Z(\mathsf{office(Manager)}) &= \{\mathsf{Room}\} \\
Z(m(c)) &= \emptyset \text{ for any other combinations of } m \text{ and } c, \\
Z(m(t)) &= \bigcup_{c \in Z(t)} Z(m(c))
\end{aligned}
$$

Fig. 7: Z for schema S_1.

Example 9: Using the algorithm in [12], we can compute Z for schema S_1 in Fig. 3. The result is presented in Fig. 7. For example, $Z(\mathsf{boss(Employee)}) = \{\mathsf{Employee}, \mathsf{Manager}\}$ means that for any object e of Employee, the result of $\mathsf{boss}(e)$ is an object of either Employee or Manager. Actually, the obtained Z is equal to E in this case. □

The next definition introduces the new rewriting rules $\rhd_{S,A,Z}$ on $T_M(C)$ which approximate $\rhd_{I,A}$.

Definition 8: Define $P_{S,A,Z}$ as the minimum set of rewriting rules $\rhd_{S,A,Z}$ on $T_M(C)$ satisfying the following three conditions:

(A) If $m(\mathbf{c}) \in A$, then $P_{S,A,Z}$ contains

$$m(\mathbf{c}) \rhd_{S,A,Z} c$$

for each $c \in Z(m(\mathbf{c}))$.

(B) If $m_c(\mathbf{c}) \in A$, $m_c \in M_{c,n}$, and $Res(m_c(\mathbf{c})) = t \neq \bot$, then $P_{S,A,Z}$ contains

$$t[\mathbf{c}/\mathbf{x}] \rhd_{S,A,Z} c$$

uses(Employee) $\rhd_{S_1}$ Host (Ai)
uses(Employee) $\rhd_{S_1}$ Server (Aii)
supervisor(Employee) $\rhd_{S_1}$ Manager (Aiii)
supervisor(Manager) $\rhd_{S_1}$ Manager (Aiv)
office(Employee) $\rhd_{S_1}$ Room (Av)
office(Manager) $\rhd_{S_1}$ Room (Avi)

supervisor(boss(Employee)) $\rhd_{S_1}$ Manager (Bi)
boss(Manager) $\rhd_{S_1}$ Manager (Bii)
located(uses(Employee)) $\rhd_{S_1}$ Room (Biii)
located(uses(Manager)) $\rhd_{S_1}$ Room (Biv)

located(Host) $\rhd_{S_1}$ Room (Ci)
located(Server) $\rhd_{S_1}$ Room (Cii)

Fig. 8: Contents of P_{S_1}.

for each $c \in Z(t[\mathbf{c}/\mathbf{x}])$.

(C) If P_S contains $t \rhd_{S,A,Z} c$ and $t'' \rhd_{S,A,Z} c''$ such that t'' is a proper subterm of t at r'', then $P_{S,A,Z}$ contains

$$t[r'' \leftarrow c''] \rhd_{S,A,Z} c'$$

for each $c' \in Z(t[r'' \leftarrow c''])$.

Define $\Rightarrow_{S,A,Z}$ as the one-step reduction relation by $\rhd_{S,A,Z}$. Let $\Rightarrow^*_{S,A,Z}$ denote the reflexive and transitive closure of $\Rightarrow_{S,A,Z}$. For readability, we often write $\rhd_S$ and P_S instead of $\rhd_{S,A,Z}$ and $P_{S,A,Z}$, respectively. □

Example 10: Fig. 8 presents the contents of P_{S_1} for schema S_1 in Fig. 3, A_1 in Example 5, and Z in Fig. 7. Rules (Ai)–(Avi) are obtained by Def. 8(A), and (Bi)–(Biv) by Def. 8(B) with composite methods supervisor and office. Rules (Ci) and (Cii) are obtained by Def. 8(C).

Rule (Cii) indicates that the user may be able to infer the location of a server. Moreover, rules (Avi) and (Bii) together indicate that the user may be able to infer the office of the boss of a manager. Compare this with the explanation in Example 8. □

The next lemma states that each rule in P_I is conservatively approximated by a rule in P_S.

Lemma 1: If there is an interpretation $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{x}] \rhd_I o \in P_I$ for some $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in \nu(c)$, then $t[\mathbf{c}/\mathbf{x}] \rhd_S c \in P_S$.

Proof: We use induction on the number of the iterations of a procedure which computes the least fixed point satisfying the three conditions in Def. 7.

Basis: Consider the case that $m(\mathbf{o}) \rhd_I o$ ($o \in \nu(c)$) is obtained from Def. 7(A). Then, $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, and $m(\mathbf{o})\downarrow = o$. Moreover, $c \in Z(m(\mathbf{c}))$ from the property of Z. From Def. 8(A), P_S contains $m(\mathbf{c}) \rhd_S c$ since $m(\mathbf{c}) \in A$ and $c \in Z(m(\mathbf{c}))$. The case that $Res(m_c(\mathbf{c}))[\mathbf{o}/\mathbf{x}] \rhd_I o$ is obtained from Def. 7(B) can be proved in the same way.
Induction: Suppose that $t''[\mathbf{o}''/\mathbf{x}'']$ ($\mathbf{o}'' \in \nu(\mathbf{c}'')$) is a proper subterm of $t[\mathbf{o}/\mathbf{x}]$ ($\mathbf{o} \in \nu(\mathbf{c})$) at r'' and that $t[\mathbf{o}/\mathbf{x}] \rhd_I o$ ($o \in \nu(c)$) and $t''[\mathbf{o}''/\mathbf{x}''] \rhd_I o''$ ($o'' \in \nu(c'')$) have been obtained. Let $t'[\mathbf{o}'/\mathbf{x}'] = t[\mathbf{o}/\mathbf{x}][r'' \leftarrow o'']$ ($\mathbf{o}' \in \nu(\mathbf{c}')$), and suppose that $t'[\mathbf{o}'/\mathbf{x}'] \rhd_I o$ is obtained from Def. 7(C). By the inductive hypothesis, P_S contains both $t[\mathbf{c}/\mathbf{x}] \rhd_S c$ and $t''[\mathbf{c}''/\mathbf{x}''] \rhd_S c''$. From the definition of $t'[\mathbf{o}'/\mathbf{x}']$, we obtain $t'[\mathbf{c}'/\mathbf{x}'] = t[\mathbf{c}/\mathbf{x}][r'' \leftarrow c'']$. Since $t'[\mathbf{o}'/\mathbf{x}'] \rhd_I o \in P_I$ implies $t'[\mathbf{o}'/\mathbf{x}']\downarrow = o$, it holds that $c \in Z(t'[\mathbf{c}'/\mathbf{x}'])$. From the above inductive hypothesis and Def. 8(C), we can conclude that $t'[\mathbf{c}'/\mathbf{x}'] \rhd_S c \in P_S$. □

We have the following theorem:

Theorem 2: Let $\tau \in T_M(C)$. If there exists no class c such that $\tau \Rightarrow^*_{S,A,Z} c$, then τ is secure, i.e., there exists no interpretation $I = (\nu, \mu)$ such that $\tau[\mathbf{o}/\mathbf{c}] \Rightarrow^*_{I,A} o$ for any $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in O_I$.
Proof: By Lemma 1, it can be easily shown that if there is $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{x}] \Rightarrow^*_I t'[\mathbf{o}'/\mathbf{x}']$ for some $\mathbf{o} \in \nu(\mathbf{c})$ and $\mathbf{o}' \in \nu(\mathbf{c}')$, then $t[\mathbf{c}/\mathbf{x}] \Rightarrow^*_S t'[\mathbf{c}'/\mathbf{x}']$. The theorem is implied by this fact. □

The proposed sufficient condition is obviously decidable, since the right-hand side of each rule $\rhd_{S,A,Z}$ is a class and therefore the "size" of the term decreases every time a rule is applied.

Example 11: Consider schema S_1 in Fig. 3, and let τ = office(boss(Employee)). We can conclude that τ is secure since no subterm of τ can be rewritten by any rule P_{S_1} in Fig. 8. □

Example 12: We said that method uses is secure in the second case of Example 1. Actually, it is not difficult to see that P_S has only located(Host) $\rhd_S$ Room and office(Employee) $\rhd_S$ Room. This implies that uses(Employee) is secure. □

It is open whether the undecidability of the security problem stems only from the uncomputability of E. In other words, it is open whether or not the sufficient condition in Theorem 2 is also a necessary one when we use E as Z.

4.2 Monadic Case

When a given schema is monadic, the algorithm in [12] computes E in time polynomial of the size of S. Moreover, when a given schema is monadic and we use E as Z, the sufficient condition in Theorem 2 is also a necessary one.

Theorem 3: Let S be a monadic schema and $\tau \in T_M(C)$. If there exists a class c' such that $\tau \Rightarrow^*_{S,A,E} c'$, then τ is insecure, i.e., there exists an interpretation I such that $\tau[o/c] \Rightarrow^*_{I,A} o'$ for some $o \in \nu(c)$ and $o' \in O_I$.
Proof: See Appendix B. □

Example 13: Consider schema S_1 in Fig. 3. Since S_1 is monadic and Z presented in Fig. 7 is equal to E, we can conclude that located(Server) is insecure by rule (Cii) in Fig. 8. Moreover,

$$\begin{aligned} \mathsf{office(boss(Manager))} &\Rightarrow_{S_1} \mathsf{office(Manager)} \\ &\Rightarrow_{S_1} \mathsf{Room}, \end{aligned}$$

and therefore office(boss(Manager)) is insecure. □

4.3 Complexity

We summarize the time complexity of deciding the sufficient condition stated in Theorem 2. Define the size of a term t as $|R(t)|$, i.e., the number of occurrences of t. Define the description length of Σ_c, denoted $\|\Sigma_c\|$, as the sum of the size of all t such that $(m(\mathbf{c}), t) \in \Sigma_c$. Also, define the size of S, denoted $\|S\|$, as follows:

$$\|S\| = |C| + |\leq| + |M| + |\Sigma_b| + \|\Sigma_c\|.$$

Let k be the maximum arity of all the methods. The height of t is defined as the maximum length of the occurrences in $R(t)$. Let L and H be the maximum size and height of all t in $\{t \mid (m(\mathbf{c}), t) \in \Sigma_c\} \cup \{\tau\}$, respectively.

By assuming $L \leq \|S\|$, the total time complexity (including computation of Z) is

$$\mathcal{O}(k^{H+1} L \|S\|^2 (|C| + 1)^{2k^{H+1}+1} \log \|S\|).$$

See Appendix C for details.

Theorem 4: For a monadic method schema, the security problem is solvable in polynomial time of the size of the schema. □

5 Conclusions

We have formalized the security problem against inference attacks on OODBs, and shown that the problem is undecidable. Then we have proposed a decidable sufficient condition for a given method to be secure, by introducing class-level inference ($\rhd_S$) which conservatively approximates object-level inference ($\rhd_I$). We believe that the approximation is fairly tight in spite of its simple definition, since the sufficient condition becomes a necessary one when the given schema is monadic.

There are several variants of the security problem. For example, [9] discusses the instance-level security. In [9], a method m is *secure* if a user cannot infer the result of m under a *given database instance*. The instance-level security problem is solvable in polynomial time in practical cases.

In several situations, imprecise inference becomes powerful enough to cause serious problems. Moreover, method schemas do not seem a perfect model of OODB schemas since they do not support multi-valued methods, update operations, and so on. Therefore, we intend to extend both inference and database models. Furthermore, the security of incomplete OODB schemas should be considered, so that the security can be checked in parallel with designing OODB schemas.

Acknowledgments

The authors are grateful to Professor Hiroyuki Seki of Nara Institute of Science and Technology for his invaluable discussions and comments. This research is supported in part by Japan Society for the Promotion of Science under Grant-in-Aid for Encouragement of Young Scientists No. 11780306.

References

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

[2] S. Abiteboul, P. Kanellakis, S. Ramaswamy and E. Waller, "Method schemas," *JCSS*, Vol. 51, No. 3, pp. 433–455, 1995.

[3] E. Bertino and P. Samarati, "Research issues in discretionary authorizations for object bases," *Proc. OOPSLA-93 Conf. Workshop on Security for Object-Oriented Systems*, pp. 183–199, 1994.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to algorithms*, The MIT Press, 1990.

[5] 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.

[6] J. Hale, J. Threet and S. Shenoi, "A practical formalism for imprecise inference control," *Database Security VIII: Proc. 8th DBSec*, pp. 139–156, 1994.

[7] T. H. Hinke, H. S. Delugach and R. Wolf, "A framework for inference-directed data mining," *Database Security X: Proc. 10th DBSec*, pp. 229–239, 1996.

[8] 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, 1998.

[9] T. Morita, Y. Ishihara, H. Seki and M. Ito, "A Formal Approach to Detecting Security Flaws in Object-Oriented Databases," *IEICE Transactions on Information and Systems*, Vol. E82-D, No. 1, pp. 89–98, 1999.

[10] 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.

[11] S. Rath, D. Jones, J. Hale and S. Shenoi, "A tool for inference detection and knowledge discovery in databases," *Database Security IX: Proc. 9th DBSec*, pp. 229–239, 1995.

[12] H. Seki, Y. Ishihara and H. Dodo, "Testing type consistency of method schemas," *IEICE Transactions on Information and Systems*, Vol. E81-D, No. 3, 1998.

[13] H. Seki, Y. Ishihara and M. Ito, "Authorization analysis of queries in object-oriented databases," *Proc. 4th DOOD, LNCS 1013*, pp. 521–538, 1995.

[14] K. Tajima, "Static detection of security flaws in object-oriented databases," *Proc. 1996 SIGMOD*, pp. 341–352, 1996.

[15] K. Zhang, "IRI: A quantitative approach to inference analysis in relational databases," *Database Security XI: Proc. 11th DBSec*, pp. 279–290, 1997.

APPENDIX

A Notation Table

$T_M(X)$: the set of all the terms freely generated by method signature M and variables X.

$t[\mathbf{t}/\mathbf{x}]$: the term obtained by simultaneously replacing every variable $x_i \in \mathbf{x}$ in term t with $t_i \in \mathbf{t}$.

$R(t)$: the set of occurrences of term t.

$t[r \leftarrow t']$: the replacement in term t of t' at occurrence r.

$Res(m(\mathbf{c}))$: the resolution of method m at tuple $\mathbf{c}$ of classes.

O_I: the set of all the objects in interpretation I.

$\rightarrow_I$: the one-step execution relation.

$\rightarrow_I^*$: the reflexive and transitive closure of $\rightarrow_I$.

$t{\downarrow}$: the execution result of $t \in T_M(O_I)$.

$\rhd_{I,A}$: the rewriting rules on $T_M(O_I)$ representing user's inference on interpretation I under authorization A.

$P_{I,A}$: the set of rewriting rules $\rhd_{I,A}$.

$\Rightarrow_{I,A}$: the one-step reduction relation by $\rhd_{I,A}$.

$\Rightarrow_{I,A}^*$: the reflexive and transitive closure of $\Rightarrow_{I,A}$.

$E(t)$: the "execution result" of $t \in T_M(C)$.

$\rhd_{S,A,Z}$: the rewriting rules on $T_M(C)$ representing user's inference on an interpretation of S under authorization A, using Z which approximates E.

$P_{S,A,Z}$: the set of rewriting rules $\rhd_{S,A,Z}$.

$\Rightarrow_{S,A,Z}$: the one-step reduction relation by $\rhd_{S,A,Z}$.

$\Rightarrow_{S,A,Z}^*$: the reflexive and transitive closure of $\Rightarrow_{S,A,Z}$.

B Complete Proof of Theorem 3

We introduce a *syntactic interpretation* $I_S = (\nu_S, \mu_S)$ of a monadic schema S, and show that I_S satisfies Theorem 3. Let N be a positive integer.

1. For each $c \in C$, $\nu_S(c) = \{c \cdot \alpha \mid \alpha \in C^*$ and the length of $c \cdot \alpha$ is at most $N\}$, where C^* denotes the Kleene closure of C.
2. For each $m_\mathrm{b} \in M_\mathrm{b}$, define $\mu_S(m_\mathrm{b})$ as follows:
   (a) If $Res(m_\mathrm{b}(c_0)) = c'$, then $\mu_S(m_\mathrm{b})(c_0) = c'$, and for $j \geq 1$,
   $$\mu_S(m_\mathrm{b})(c_0 \cdot c_1 \cdot c_2 \cdots c_j) = \begin{cases} c_1 \cdot c_2 \cdots c_j & \text{if } c_1 \leq c', \\ c' \cdot c_2 \cdots c_j & \text{otherwise.} \end{cases}$$
   (b) If $Res(m_\mathrm{b}(c_0)) = \bot$, then $\mu_S(m_\mathrm{b})(c_0 \cdot c_1 \cdot c_2 \cdots c_j)$ $(j \geq 0)$ is undefined.

We consider a syntactic interpretation $I_S = (\nu_S, \mu_S)$ with sufficiently large N.

In order to prove that I_S satisfies the theorem, we need several technical lemmas.

Lemma 2: Let $t \in T_M(\{x\})$, and suppose that $c' \in E(t[c/x])$. 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),
   $$t[\beta \cdot c' \cdot \alpha/x] \rightarrow_{I_S}^* c' \cdot \alpha.$$

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

Proof: Suppose that $c' \in E(t[c/x])$. By the definition of E, there exists an interpretation $I = (\nu, \mu)$ such that $t[o/x] \rightarrow_I^* o'$ for some $o \in \nu(c)$ and $o' \in \nu(c')$. Consider the i-th step (counting from zero) $t_i[o_i/x] \rightarrow_I t_{i+1}[o_{i+1}/x]$ of the reduction $t[o/x] \rightarrow_I^* o'$, 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 $m_i(o_i)$ be the innermost term of $t_i[o_i/x]$. Define β_i $(0 \leq i \leq n-1)$ as follows:

$$\beta_i = \begin{cases} c_i & \text{if } m_i \in M_{\mathrm{b}}, \\ \varepsilon & \text{otherwise.} \end{cases}$$

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

It is easily verified that β satisfies the first condition since

- $o_0 = o \in \nu(c)$, and
- if $m_i \in M_{\mathrm{c}}$, then $o_{i+1} = o_i$ by the definition of $\rightarrow_I$.

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

$$\begin{aligned} t_0[\beta \cdot c' \cdot \alpha/x] &= t_0[\beta_0 \cdot \beta_1 \cdots \beta_{n-1} \cdot c' \cdot \alpha/x] \\ &\rightarrow_{I_S} t_1[\beta_1 \cdots \beta_{n-1} \cdot c' \cdot \alpha/x]. \end{aligned}$$

By repeating this, $t[\beta \cdot c' \cdot \alpha/x] \rightarrow_{I_S}^* c' \cdot \alpha$. □

Lemma 3: 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 $c'' \in E(t''[c/x])$ and $c' \in E(t'[c''/x])$ hold, then $c' \in E(t[c/x])$.

Proof: By Lemma 2, there exist reduction strings β'' of $(t''[c/x], c'')$ and β' of $(t'[c''/x], c')$, i.e.,

- the first symbol of $\beta'' \cdot c''$ is c,
- $t''[\beta'' \cdot c'' \cdot \alpha''/x] \rightarrow_{I_S}^* c'' \cdot \alpha''$ for any $\alpha'' \in C^*$,
- the first symbol of $\beta' \cdot c'$ is c'', and
- $t'[\beta' \cdot c' \cdot \alpha'/x] \rightarrow_{I_S}^* c' \cdot \alpha'$ for any $\alpha' \in C^*$.

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

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

By the definition of E, c' must be in $E(t[c/x])$. □

Lemma 4: Let t, t', $t'' \in T_M(\{x\})$ such that t'' is a subterm of t at r'' and $t' = t[r'' \leftarrow x]$. Suppose that $c'' \in E(t''[c/x])$ and $c' \in E(t'[c''/x])$. Let β' be an arbitrary reduction string of $(t'[c''/x], c')$. Then, there exist reduction strings β of $(t[c/x], c')$ and β'' of $(t''[c/x], c'')$ such that $\beta = \beta'' \cdot \beta'$.

Proof: Let β'' and β' be arbitrary reduction strings of $(t''[c/x], c'')$ and $(t'[c''/x], c')$, respectively. By the proof of Lemma 3, $\beta'' \cdot \beta'$ is a reduction string of $(t[c/x], c')$. This fact implies the lemma. □

The next lemma states that if $t[c/x] \rhd_S c' \in P_S$, then $t[o/x] \rhd_{I_S} o' \in P_{I_S}$ for some $o \in \nu_S(c)$ and $o' \in \nu_S(c')$. In this sense, P_S contains no "wrong" rules.

Lemma 5: Let c, $c' \in C$, and $t \in T_M(\{x\})$. If $t[c/x] \rhd_S c' \in P_S$, then for an arbitrary reduction string β of $(t[c/x], c')$ and for any $\alpha \in C^*$,

$$t[\beta \cdot c' \cdot \alpha/x] \rhd_{I_S} c' \cdot \alpha \in P_{I_S}.$$

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

Basis: Suppose that $m(c) \rhd_S c'$ is obtained from Def. 8(A). Let β be an arbitrary reduction string of $(m(c), c')$. Since $m(c) \in A$ and $m(\beta \cdot c' \cdot \alpha) \rightarrow_{I_S}^* c' \cdot \alpha$, we obtain $m(\beta \cdot c' \cdot \alpha) \rhd_{I_S} c' \cdot \alpha$ from Def. 7(A). The case that $Res(m_{\mathrm{c}}(c))[c/x] \rhd_S c'$ is obtained from Def. 8(B) can be proved similarly.

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'',
- $t' = t[r'' \leftarrow x]$,
- $t''[c/x] \rhd_S c'' \in P_S$,
- $t[c/x] \rhd_S c_1 \in P_S$ for some c_1, and
- $c' \in E(t'[c''/x])$.

Also suppose that $t'[c''/x] \rhd_S c'$ is obtained from Def. 8(C). Since $t''[c/x] \rhd_S c'' \in P_S$, it holds that $c'' \in E(t''[c/x])$ by Def. 8. By Lemma 3, it holds that $c' \in E(t[c/x])$. Then, by Def. 8 again, $t[c/x] \rhd_S c'$ must be in P_S. Let β' be an arbitrary reduction string of $(t'[c''/x], c')$. That is, the first symbol of $\beta' \cdot c'$ is c'' and $t'[\beta' \cdot c' \cdot \alpha'/x] \rightarrow_{I_S}^* c' \cdot \alpha'$ for any $\alpha' \in C^*$. By Lemma 4 and the inductive hypothesis for $t[c/x] \rhd_S c'$ and $t''[c/x] \rhd_S c''$, there exist β and β'' such that

- the first symbol of $\beta \cdot c'$ is c,
- $t[\beta \cdot c' \cdot \alpha/x] \rhd_{I_S} c' \cdot \alpha \in P_{I_S}$ for any $\alpha \in C^*$,
- the first symbol of $\beta'' \cdot c''$ is c,

- $t''[\beta'' \cdot c'' \cdot \alpha''/x] \rhd_{I_S} c'' \cdot \alpha'' \in P_{I_S}$ for any $\alpha'' \in C^*$, and
- $\beta = \beta'' \cdot \beta'$.

When we choose α and α'' 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. 7(C),

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

Since $t[\beta \cdot c' \cdot \alpha/x][r'' \leftarrow c'' \cdot \alpha''] = t'[c'' \cdot \alpha''/x] = t'[\beta' \cdot c' \cdot \alpha/x]$, P_{I_S} contains $t'[\beta' \cdot c' \cdot \alpha/x] \rhd_{I_S} c' \cdot \alpha$ for any α. □

Finally, the next lemma states that if $t[c/x] \Rightarrow_S^* t'[c''/x]$, then $t[o/x] \Rightarrow_{I_S}^* t'[o''/x]$ for some $o \in \nu_S(c)$ and $o'' \in \nu_S(c'')$.

Lemma 6: Let c, $c' \in C$ and t, $t' \in T_M(\{x\})$. If $t[c/x] \Rightarrow_S^* t'[c''/x]$, then there exists a string β such that the first symbol $\beta \cdot c''$ is c and for any $\alpha'' \in C^*$,

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

Proof: We use induction on the length of the reduction $t[c/x] \Rightarrow_S^* t'[c''/x]$.
Basis: It is obvious when the length of the reduction is zero.
Induction: Consider the following reduction:

$$t[c/x] \Rightarrow_S^* t_i[c_i/x] \Rightarrow_S t'[c''/x].$$

By the inductive hypothesis, there exists a string β_i such that the first symbol of $\beta_i \cdot c_i$ is c and for any $\alpha_i \in C^*$,

$$t[\beta_i \cdot c_i \cdot \alpha_i/x] \Rightarrow_{I_S}^* t_i[c_i \cdot \alpha_i/x].$$

On the other hand, by Def. 8, there exists a subterm t'' of t_i at r'' such that $t''[c_i/x] \rhd_S c''$ and $t'[c''/x] = t_i[c_i/x][r'' \leftarrow c'']$. By Lemmas 2 and 5, there exists β' such that the first symbol of $\beta' \cdot c''$ is c_i and for any $\alpha'' \in C^*$, rule $t''[\beta' \cdot c'' \cdot \alpha''/x] \rhd_{I_S} c'' \cdot \alpha''$ exists in P_{I_S}. By Def. 7, it follows that $t''[\beta' \cdot c'' \cdot \alpha''/x] \Rightarrow_{I_S} c'' \cdot \alpha''$. Hence,

$$t_i[\beta' \cdot c'' \cdot \alpha''/x] \Rightarrow_{I_S} t'[c'' \cdot \alpha''/x].$$

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

$$t[\beta_i \cdot \beta' \cdot c'' \cdot \alpha''/x] \Rightarrow_{I_S}^* t'[c'' \cdot \alpha''/x].$$

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

Theorem 3 immediately follows from Lemma 6.

C Complexity Analysis

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

(S1) Compute Z_0 from S using the type checking algorithm in [12], where Z_0 is a mapping Z whose domain is restricted to $\{m(\mathbf{c}) \mid m \in M_n, \mathbf{c} \in C^n\}$.

(S2) Compute $P_{S,A,Z}$ from S, A, and Z_0.

(S3) Determine whether there exists a class c such that $\tau \Rightarrow_{S,A,Z}^* c$. If such c exists, then output "τ may be insecure." Otherwise, output "τ is secure."

We analyze the time complexity of the algorithm. For the reader's convenience, we explain the notation introduced in Section 4.3 again. Define the size l_t of a term t as $|R(t)|$, i.e., the number of occurrences of t. Define the description length of Σ_c, denoted $\|\Sigma_c\|$, as the sum of l_t for all $(m(\mathbf{c}), t) \in \Sigma_c$. Also, define the size of S, denoted $\|S\|$, as follows:

$$\|S\| = |C| + |\leq| + |M| + |\Sigma_b| + \|\Sigma_c\|.$$

For readability, we use N to mean $\|S\|$. Let k be the maximum arity of all the methods. The height of t, denoted h_t, is defined as the maximum length of the occurrences in $R(t)$. Let L and H be the maximum size and height of all t in $\{t \mid (m(\mathbf{c}), t) \in \Sigma_c\} \cup \{\tau\}$, respectively. We assume that $L \leq N$, i.e., the size of τ does not exceed the size of S.

First, consider (S1). The time complexity of computing $Z_0(m(\mathbf{c}))$ for all $m \in M_n$ and $\mathbf{c} \in C^n$ is

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

which is given in [12]. In this type-checking algorithm, Z_0 is implemented by a table, and retrieving an element from Z_0 takes $\mathcal{O}(kN|C|^k)$ time. In (S1), we also reconstruct Z_0 by a more efficient data structure such as binomial heap [4]. The time complexity ρ_{Z_0} of retrieving an element from or inserting an element into Z_0 becomes

$$\rho_{Z_0} = \mathcal{O}(k \log(|M| \cdot |C|^k)) = \mathcal{O}(k^2 \log N).$$

Note that $\rho_{Z_0} \neq \mathcal{O}(k \log N)$, since the keys are terms $m(\mathbf{c})$ and a key comparison takes $\mathcal{O}(k)$ time. This reconstruction takes

$$\begin{aligned} &\mathcal{O}(|M| \cdot |C|^k (kN|C|^k + \rho_{Z_0})) \\ &= \mathcal{O}(kN|C|^k (N|C|^k + k \log N)) \end{aligned}$$

time. Thus, the complexity of (S1) is

$$\mathcal{O}(kN|C|^k (N|C|^k + k \log N)). \quad (1)$$

T1 $Q_{\text{ans}} \leftarrow \emptyset$, $Q_\Delta \leftarrow \emptyset$
T2 compute $Res(m(\mathbf{c}))$ for all $m(\mathbf{c})$
T3 **for each** $m(\mathbf{c})$ in A
T4 add $m(\mathbf{c})$ to Q_Δ
T5 **if** $m \in M_c$ **then**
T6 let t be $Res(m(\mathbf{c}))$
T7 add $t[\mathbf{c}/\mathbf{x}]$ to Q_Δ
T8 **while** $Q_\Delta \neq \emptyset$
T9 $Q'_\Delta \leftarrow \emptyset$
T10 **for each** (t, t') in
$Q_{\text{ans}} \times Q_\Delta \cup Q_\Delta \times Q_{\text{ans}} \cup Q_\Delta \times Q_\Delta$
T11 **if** t' is a subterm of t at r' **then**
T12 **if** $Z(t')$ has not been computed **then**
T13 compute $Z(t')$ from Z_0
T14 **for each** c' in $Z(t')$
T15 add $t[r' \leftarrow c']$ to Q'_Δ
T16 $Q_{\text{ans}} \leftarrow Q_{\text{ans}} \cup Q_\Delta$, $Q_\Delta \leftarrow Q'_\Delta$
T17 **output** Q_{ans} as Q

Fig. 9: Procedure for computing Q.

Next, consider (S2). Define $Q = \{t \mid t \rhd_S c \in P_S\}$. In order to compute P_S, it suffices to compute Q, since the right-hand side of $\rhd_S$ can be computed from the left-hand side and Z.

Fig. 9 shows a procedure for computing Q. Suppose that variables Q_{ans}, Q_Δ, Q'_Δ, and Z are implemented by binomial heaps. Let $\rho_{Q_{\text{ans}}}$ denote the complexity of retrieving an element from or inserting an element into Q_{ans}. Define ρ_{Q_Δ}, $\rho_{Q'_\Delta}$, and ρ_Z in the same way. Then,

$$\rho_{Q_{\text{ans}}} = \rho_{Q_\Delta} = \rho_{Q'_\Delta} = \rho_Z = \mathcal{O}(L \log |Q|),$$

where L is for a key comparison.

Before analyzing the procedure in Fig. 9 in detail, we estimate $|Q|$. Since it is difficult to estimate $|Q|$ directly, we introduce a finite set Q_0 of terms which possibly appear in the left-hand side of $\rhd_S$. Formally,

$$Q_0 = \bigcup_{(m(\mathbf{c}),t)\in\Sigma_c} X_{t[\mathbf{c}/\mathbf{x}]},$$

where X_t ($t \in T_M(C)$) is defined as follows:

$$\begin{aligned} X_c &= C, \\ X_{m(\mathbf{t})} &= C \cup \{m(\mathbf{t}') \mid t'_i \in X_{t_i}\}. \end{aligned}$$

Intuitively, X_t is the set of all the terms obtained by replacing arbitrary subterms of t with arbitrary classes. Clearly $Q \subseteq Q_0$.

The size of X_t can be obtained by solving the following (in)equalities:

$$\begin{aligned} |X_c| &= |C|, \\ |X_{m(\mathbf{t})}| &\leq |C| + \prod_i |X_{t_i}|. \end{aligned}$$

If $k = 1$, then

$$|X_t| \leq (h_t + 1)|C|,$$

and therefore,

$$\begin{aligned} |Q_0| &\leq \sum_{(m(\mathbf{c}),t)\in\Sigma_c} |X_{t[\mathbf{c}/\mathbf{x}]}| \\ &\leq \sum_{(m(\mathbf{c}),t)\in\Sigma_c} (h_{t[\mathbf{c}/\mathbf{x}]} + 1)|C| \\ &= \sum_{(m(\mathbf{c}),t)\in\Sigma_c} l_{t[\mathbf{c}/\mathbf{x}]}|C| \\ &= \|\Sigma_c\| \cdot |C| \\ &\leq N|C|, \end{aligned} \tag{2}$$

since $l_t = h_t + 1$ if $k = 1$. Next, consider the case that $k \geq 2$. For any nonnegative integer h, let K_h denote $k^h + k^{h-1} + \ldots + k^0$. In what follows, we show that

$$|X_t| \leq (|C| + 1)^{K_{h_t}}. \tag{3}$$

If $h_t = 0$, then $|X_t| = |C| \leq (|C|+1)^{K_0} = |C|+1$. Suppose that Eq. (3) holds for any term t' such that $h_{t'} \leq h$ for some $h \geq 0$. Consider a term $t = m(\mathbf{t}')$ such that $h_t = h + 1$. Then,

$$\begin{aligned} |X_t| &\leq |C| + \prod_i |X_{t'_i}| \\ &\leq |C| + ((|C| + 1)^{K_h})^k \\ &= |C| + (|C| + 1)^{K_{h+1}-1} \\ &\leq (|C| + 1)^{K_{h+1}}. \end{aligned}$$

Therefore, Eq. (3) holds and

$$\begin{aligned} |Q_0| &\leq \sum_{(m(\mathbf{c}),t)\in\Sigma_c} |X_{t[\mathbf{c}/\mathbf{x}]}| \\ &\leq \sum_{(m(\mathbf{c}),t)\in\Sigma_c} (|C| + 1)^{K_{h_{t[\mathbf{c}/\mathbf{x}]}}} \\ &\leq \|\Sigma_c\|(|C| + 1)^{K_H} \\ &\leq N(|C| + 1)^{K_H} \\ &\leq N(|C| + 1)^{k^{H+1}}, \end{aligned} \tag{4}$$

using $K_H \leq k^{H+1}$ if $k \geq 2$. After all, from Eqs. (2) and (4), we obtain

$$|Q| \leq |Q_0| \leq N(|C| + 1)^{k^{H+1}}.$$

Let us analyze the procedure in Fig. 9 in detail. See (T2). A straightforward algorithm can compute *Res* in

$$\mathcal{O}(kN|C|^k) \tag{5}$$

time. Next, see (T3) through (T7). In (T3), $|A| \leq |M| \cdot |C|^k \leq N|C|^k$. If *Res* is implemented by an appropriate data structure, then retrieving an element from *Res* takes

$$\rho_{Res} = \mathcal{O}(k \log(|M| \cdot |C|^k)) = \mathcal{O}(k^2 \log N)$$

time in (T6). In (T7), computing $t[\mathbf{c}/\mathbf{x}]$ takes $\mathcal{O}(L)$ time. Therefore, executing (T3) through (T7) takes

$$\begin{aligned} &\mathcal{O}(|A|(\rho_{Q_\Delta} + \log|M_c| + \rho_{Res} + L + \rho_{Q_\Delta})) \\ &= \mathcal{O}(N|C|^k(L \log|Q| + k^2 \log N)) \\ &= \mathcal{O}(k^{H+1}LN|C|^k \log N), \end{aligned} \tag{6}$$

using $k \leq L$.

See (T8) through (T16). By Q_Δ and Q'_Δ, we avoid selecting a duplicated pair of t and t' in (T10). In other words, (T11) through (T15) are executed at most $|Q|^2$ times, and therefore, (T16) is also executed at most $|Q|^2$ times. Moreover, (T13) is executed at most $|Q|$ times, since the condition of (T12) holds at most $|Q|$ times.

In (T11), Knuth-Morris-Pratt string matching algorithm [4] can check in $\mathcal{O}(L)$ time whether t' is a subterm of t. Constructing $t[r' \leftarrow c]$ in (T15) takes $\mathcal{O}(L)$ time. Computing $Q_{\text{ans}} \cup Q_\Delta$ takes $\mathcal{O}(L \log|Q|)$ time [4]. Therefore, the complexity of (T11) through (T16) except (T13) is

$$\begin{aligned} &\mathcal{O}(|Q|^2(L + \rho_Z + \rho_Z + |C|(L + \rho_{Q'_\Delta})) \\ &\qquad + |Q|^2 \cdot L \log|Q|) \\ &= \mathcal{O}(|Q|^2 \cdot |C| \cdot L \log|Q|) \\ &= \mathcal{O}(k^{H+1}LN^2(|C|+1)^{2k^{H+1}+1} \log N). \end{aligned} \tag{7}$$

On the other hand, in (T13), $Z(t)$ is computed from Z_0 as follows:

$$\begin{aligned} Z(m(\mathbf{c})) &= Z_0(m(\mathbf{c})), \\ Z(m(\mathbf{t})) &= \bigcup_{\mathbf{c} \in Z(t_1) \times \cdots \times Z(t_n)} Z_0(m(\mathbf{c})). \end{aligned}$$

The time complexity of computing $Z(t)$ is

$$\mathcal{O}(\rho_{Z_0} l_t |C|^{k+1}) = \mathcal{O}(k^2 L|C|^{k+1} \log N).$$

The total complexity of (T13) is

$$\begin{aligned} &\mathcal{O}(|Q| \cdot k^2 L|C|^{k+1} \log N) \\ &= \mathcal{O}(k^2 LN(|C|+1)^{k^{H+1}+k+1} \log N). \end{aligned} \tag{8}$$

U1 $D_{\text{ans}} \leftarrow \emptyset$, $D_\Delta \leftarrow \{\tau\}$
U2 **while** $D_\Delta \neq \emptyset$
U3 $\quad D'_\Delta \leftarrow \emptyset$
U4 $\quad$ **for each** (t, t'') in $D_\Delta \times Q$
U5 $\quad\quad$ **if** t'' is a subterm of t at r'' **then**
U6 $\quad\quad\quad$ **for each** c'' in $Z(t'')$
U7 $\quad\quad\quad\quad$ add $t[r'' \leftarrow c'']$ to D'_Δ
U8 $\quad D_{\text{ans}} \leftarrow D_{\text{ans}} \cup D_\Delta$, $D_\Delta \leftarrow D'_\Delta$
U9 **if** D_{ans} contains a class **then**
U10 $\quad$ **output** "τ may be insecure"
U11 **else**
U12 $\quad$ **output** "τ is secure"

Fig. 10: Procedure for determining $\tau \Rightarrow_S^* c$.

Both of Eqs. (5) and (6) are bounded by Eq. (7). Furthermore, Eq. (8) is also bounded by Eq. (7) since $k \leq L \leq N$. Thus, the complexity of (S2) is given by Eq. (7).

Lastly, consider (S3). Let $D = \{t \mid \tau \Rightarrow_S^* t\}$. Then,

$$|D| \leq |X_\tau| \leq |Q_0| = \mathcal{O}(N(|C|+1)^{k^{H+1}}),$$

since $h_\tau \leq H$.

Fig. 10 shows a procedure for determining whether D contains a class. Suppose that D_{ans}, D_Δ, D'_Δ are implemented by binomial heaps. By D_Δ and D'_Δ, we avoid selecting $t \in D$ more than once. Therefore, (U5) through (U7) are executed at most $|D| \cdot |Q|$ times. (U8) is also executed at most $|D| \cdot |Q|$ times. Retrieving an element from or inserting an element into D'_Δ takes

$$\rho_{D'_\Delta} = \mathcal{O}(L \log|D|) = \mathcal{O}(k^{H+1}L \log N)$$

time. Computing $D \cup D_\Delta$ also takes $\mathcal{O}(L \log|D|)$ time. Thus, executing (U2) through (U8) takes

$$\begin{aligned} &\mathcal{O}(|D| \cdot |Q|(L + \rho_Z + |C|(L + \rho_{D'_\Delta})) \\ &\qquad + |D| \cdot |Q| \cdot L \log|D|)) \\ &= \mathcal{O}(|D| \cdot |Q| \cdot |C| \cdot L \log|D|) \\ &= \mathcal{O}(k^{H+1}LN^2(|C|+1)^{2k^{H+1}+1} \log N). \end{aligned} \tag{9}$$

(U9) can be checked in $\mathcal{O}(|D|)$ time. Therefore, the time complexity of executing (S3) is given by Eq. (9).

By Eqs. (1), (7), and (9), the time complexity of the algorithm is

$$\mathcal{O}(k^{H+1}LN^2(|C|+1)^{2k^{H+1}+1} \log N).$$