An Equational Logic Based Approach to the Security Problem against Inference Attacks on Object-Oriented Databases[⋆]

Yasunori ISHIHARA [a] Toshiyuki MORITA [b†] Hiroyuki SEKI [b]
Minoru ITO [b]

[a]*Graduate School of Information Science and Technology, Osaka University*
1–5, Yamadaoka, Suita, Osaka 565-0871 JAPAN

[b]*Graduate School of Information Science, Nara Institute of Science and Technology*
8916–5, Takayama, Ikoma, Nara 630-0192 JAPAN

⋆ Preliminary versions of this article appeared in the Proc. 13th IFIP WG11.3 Conf. on Database Security, 1999 [1], and in the Proc. 6th Int'l Conf. on Information and Communications Security, 2004 [2].

† Currently, he is with Systems Development Laboratory, Hitachi, Ltd.

Contact: Yasunori ISHIHARA
Graduate School of Information Science and Technology
Osaka University
1–5, Yamadaoka, Suita, Osaka 565-0871 JAPAN
Phone: +81-6-6879-4516 FAX: +81-6-6879-4519
E-mail: ishihara@ist.osaka-u.ac.jp

Abstract

A query is said to be secure against inference attacks by a user if there exists no database instance for which the user can infer the result of the query, using only authorized queries to the user. In this paper, first, the security problem against inference attacks on object-oriented databases is formalized. The definition of inference attacks is based on equational logic. Secondly, the security problem is shown to be undecidable, and a decidable sufficient condition for a given query to be secure under a given schema is proposed. The idea of the sufficient condition is to over-estimate inference attacks using over-estimated results of static type inference. The third contribution is to propose subclasses of schemas and queries for which the security problem becomes decidable. Lastly, the decidability of the security problem is shown to be incomparable with the static type inferability, although the tightness of the over-estimation of the inference attacks is affected in a large degree by that of the static type inference.

Key words: object-oriented database, database security, inference attack, computational complexity

1 Introduction

Nowadays, many people and organizations have a growing interest in data security. For a database system to be secure, secrecy, integrity, and availability of data must be achieved appropriately with respect to a given security policy. Since databases are often used as the core of the systems requiring high-level security (e.g., e-business, Web services, etc.), it is desirable that the verification of the security of databases be possible. Various authorization models for databases have been proposed and studied so far in order to both represent given security policies in a natural way and analyze the users' authorization in a rigorous way. In the context of object-oriented databases (OODBs), the method-based authorization model [3,4] 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 rights $m(c_1, \ldots, c_n)$, 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 with $1 \leq i \leq n$. 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 [5]. Although such indirect invocations are useful for data hiding [5], they may also allow a violation of secrecy by *inference attacks* in some situations.

Example 1 *Let* Employee, Host, *and* Room *be classes representing employees, hosts, and rooms, respectively. Suppose that a method* computer *returns the host which a given employee uses, and a method* location *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* $\text{office}(x) = \text{location}(\text{computer}(x))$.

Now suppose that the physical network topology 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 *has obtained that* $\text{computer}(\text{John}) = \text{mars}$ *and* $\text{office}(\text{John}) = \text{A626}$ *using the authorized methods. Also suppose that* u *knows the implementation body of* office *as its behavioral specification. Then,* u *knows that* $\text{location}(\text{computer}(\text{John})) = \text{A626}$, *and therefore,* u *can infer that* $\text{location}(\text{mars}) = \text{A626}$.

On the other hand, suppose that method computer *retrieves top secret information and therefore the authorization for* u *is set as shown in Fig. 2. Then,* u *knows that* $\text{location}(\text{mars}) = \text{A626}$, $\text{office}(\text{John}) = \text{A626}$, *and* $\text{office}(x) = \text{location}(\text{computer}(x))$, *similarly to the former case. However,* u *cannot conclude that* $\text{computer}(\text{John}) = \text{mars}$ *only from the above information, since there may be another host, say* neptune, *such that* $\text{computer}(\text{John}) = \text{neptune}$ *and* $\text{location}(\text{neptune}) = \text{A626}$.

Let S be a database schema and c_1,..., c_n be classes in S. An n-ary query (i.e., a composition of methods) $q(x_1, \ldots, x_n)$ is said to be *secure* at $(c_1, \ldots, c_n)$ against inference attacks by u if u cannot infer the result of $q(o_1, \ldots, o_n)$ for any objects o_i of class c_i in any database instance I of S, using only authorized methods to u. Otherwise, $q(x_1, \ldots, x_n)$ is *insecure*. For example, if $\mathsf{computer}(\mathsf{Employee})$ and $\mathsf{office}(\mathsf{Employee})$ are authorized, then $\mathsf{location}(x)$ is insecure at Host since the user can infer $\mathsf{location}(\mathsf{mars}) = \mathsf{A626}$ under the database instance shown in Fig. 1. On the other hand, it will be shown later that $\mathsf{computer}(x)$ is secure at $\mathsf{Employee}$ when only $\mathsf{location}(\mathsf{Host})$ and $\mathsf{office}(\mathsf{Employee})$ 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.

The contribution of the paper is fourfold. First, the security problem against inference attacks is formally defined. As a formal model of OODB schemas, method schemas [6] are adopted since they support such basic features of OODBs as method overloading, dynamic binding, and complex objects, although the returned value of a method execution is limited to a single object. The semantics is simply defined based on term rewriting. Then, user's inference based on equational logic is defined on the assumption that all the information available to the user is the execution results of authorized methods and the implementation bodies of authorized methods. Technically, user's inference is also treated in the framework of term rewriting. The execution result of a term can be computed as it is if the user has authorization to all the methods in the term. Otherwise, there may exist an indirect way to compute the term by equivalently rewriting it to another term which contains only authorized methods. Our definition of user's inference provides the rewriting rules representing such direct/indirect computation and enables us to treat the security against inference attacks in a simple and rigorous way.

Example 2 *Consider the case of Fig. 1 in Example 1 again. Let I denote the database instance shown in Fig. 1. In this paper, user's inference based on equational logic is treated in the framework of term rewriting as follows. First, executing* $\mathsf{computer}(\mathsf{John})$ *and* $\mathsf{office}(\mathsf{John})$ *is authorized to the user u, and u can obtain the execution results under I. This fact is represented by the following rewriting rules:*

$$\begin{aligned} \mathsf{computer}(\mathsf{John}) &\rhd_I \mathsf{mars}, \\ \mathsf{office}(\mathsf{John}) &\rhd_I \mathsf{A626}. \end{aligned}$$

Intuitively, the rule $\rhd_I$ represents a primitive step of inference by the user. Next, u knows that the implementation body of $\mathsf{office}(x)$ *is* $\mathsf{location}(\mathsf{computer}(x))$*. The rewriting rule below represents this fact combined with the fact that the execution result of* $\mathsf{office}(\mathsf{John})$ *is* $\mathsf{A626}$*:*

$\mathsf{location}(\mathsf{computer}(\mathsf{John})) \rhd_I \mathsf{A626}$.

Moreover, to simulate the inference based on equational logic, some more rewriting rules (e.g., $\mathsf{location}(\mathsf{mars}) \rhd_I \mathsf{A626}$*) are necessary. Then, whether the execution result of a term can be inferred corresponds to whether the term can be reduced to an object by the rewriting rules* $\rhd_I$*. In this example, the execution result of* $\mathsf{location}(\mathsf{mars})$ *can be inferred because of the rule* $\mathsf{location}(\mathsf{mars}) \rhd_I \mathsf{A626}$*. A little more complicated example will be given in Example 16 in Section 3.2.*

Secondly, the security problem is shown to be undecidable. Also, a decidable sufficient condition for a given query to be secure under a given schema is proposed. The main idea of the sufficient condition is to "conservatively" approximate the user's inference. The user's inference is object-level inference, while the approximation is class-level inference. To accomplish the class-level inference, the technique of type inference is used, where type inference means deriving the classes to which the possible results of the method execution belong. Unfortunately, exact type inference is impossible in general [6]. However, the type inference algorithm proposed in [7] 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.

The third contribution is to propose subclasses of schemas and queries for which the security problem becomes decidable. We focus on the linearity of schemas and/or queries, which is a popular notion of the field of term rewriting. A query (i.e., a term with variables) t is *linear* if no variable in t appears more than once. A schema S is *linear* if all the implementation bodies of the methods in S are linear. Then, in the linear case, the security problem is shown to be decidable. More precisely, the exact type inference of linear queries is possible under linear schemas, and the user's inference can be exactly simulated using the result of type inference (i.e., the proposed sufficient condition mentioned above is also a necessary condition).

The fourth contribution is to investigate the relationship between type inferability and decidability of the security problem (see also Table 1). The security of type-inferable but non-linear queries is undecidable under linear schemas. On the other hand, type inference is impossible for queries whose security is decidable under linear schemas. These results imply that type inferability and decidability of the security problem are incomparable (compare the second columns of Tables 1(a) and 1(b)).

In this paper we discuss "logical" inference in OODBs in the sense that the result of the inference is always true. The inference in statistical databases [9] is a kind of logical inference. Ref. [10] proposes a mechanism that accomplishes maximum data availability as long as given sensitive information is secure against logical inference. Ref. [11] focuses on logical 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 [11] does not, detecting inferability in our formalization is not trivial.

On the other hand, some of the recent researches concentrate on “statistical” inference, i.e., inference with some statistical assumptions. Ref. [12] discusses the inference based on Bayesian methods. In [13], a quantitative measure of inference risk is formally defined. In [14,15], the security against statistical inference is defined based on information theory.

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. In Section 4, we show that the problem is undecidable, and propose a sufficient condition for a query to be secure. In Section 5, we show that the problem is decidable in the linear case. In Section 6, we discuss the relationship between type inferability and security decidability. Finally, in Section 7, we conclude this paper.

2 The OODB Model

We adopt *method schemas* [6,16] as a formal model of OODBs. Method schemas have such basic features of OODBs as method overloading, dynamic binding, and complex objects, although the returned value of a method execution is limited to a single object. The semantics can simply be defined based on term rewriting [17]. In this section, we first introduce some notations and concepts for term rewriting. Then, by using those notations and concepts, we restate the original definition of method schemas.

2.1 Notations

Let F be a family of disjoint sets F_0, F_1, F_2,..., where, for a nonnegative integer n, F_n 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. A term $t \in T_M(X)$ is *linear* if every variable in X appears in t at most once.

For a set U, let U^n denote the Cartesian product $\underbrace{U \times \cdots \times U}_{n}$. Hereafter, we often use a bold letter $\mathbf{u}$ to mean $(u_1, \ldots, u_n)$ without explicitly mentioning it when n is irrelevant or obvious from the context. Also, we write $u \in \mathbf{u}$ if $u = u_i$ for some i.

Define the set $Pos(t)$ of *positions* of a term t as the smallest set of sequences of positive integers with the following two properties:

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

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

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

Let $V(t)$ denote the set of positions r of t such that $t/r \in X$. That is, $V(t)$ is the positions of variables of t, and hence, $V(t) \subseteq Pos(t)$. Let $\sigma : V(t) \to T_F(X)$ be a *substitution mapping*, i.e., a mapping such that if $t/r = t/r'$, where $r, r' \in V(t)$, then $\sigma(r) = \sigma(r')$. For a term $t \in T_F(X)$, let $t\sigma$ denote the term obtained from t by replacing each variable t/r with $\sigma(r)$. Also, let $t[\mathbf{t}/\mathbf{x}]$ denote $t\sigma$, where σ is a substitution mapping such that $\sigma(r_i) = t_i$ with $t/r_i = x_i$ for each $r_i \in V(t)$. For example, $f(x_1, g(x_1, x_2))[(f(a), x_1)/(x_1, x_2)] = f(f(a), g(f(a), x_1))$.

2.2 Database Schemas

Let C be a finite set of *class names* (or simply classes). Let M be a family of mutually disjoint finite sets M_0, M_1, M_2,..., where, for a nonnegative integer n, M_n is a set of function symbols (or often called *method names*) of arity n. Each M_n is partitioned into $M_{\mathrm{b},n}$ and $M_{\mathrm{c},n}$. Let $M_\mathrm{b} = \bigcup_{n \geq 0} M_{\mathrm{b},n}$ and $M_\mathrm{c} = \bigcup_{n \geq 0} M_{\mathrm{c},n}$. Each $m_\mathrm{b} \in M_\mathrm{b}$ (resp. $m_\mathrm{c} \in M_\mathrm{c}$) is called a *base method name* (resp. *composite method name*). We say that M is a *method signature*.

Definition 3 (Method definition) *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* $(m_\mathrm{b}(\mathbf{c}), c)$ *for some* $c \in C$. *A* composite method definition *of* $m_\mathrm{c} \in M_{\mathrm{c},n}$ *at* $\mathbf{c}$ *is a pair* $(m_\mathrm{c}(\mathbf{c}), t)$ *for some* $t \in T_M(\{x_1, \ldots, x_n\})$.

For $1 \leq i \leq n$, let o_i be an object of class c_i (see Def. 8 for the formal definition of objects). 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 $t[\mathbf{o}/\mathbf{x}]$.

Definition 4 (Method schema [6,16]) *A* method schema 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 on* C *representing a class hierarchy,*

(3) M *is a method signature,*
(4) Σ_{b} *is a set of base method definitions, and*
(5) Σ_{c} *is a set of composite method definitions.*

For every combination $\mathbf{c} \in C^n$ *and* $m \in M_n$*, there must exist at most one method definition of* m *at* $\mathbf{c}$*.*

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.

Example 5 *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*.*

2.3 Inheritance

Let $\mathbf{c} \in C^n$ and $m \in M_n$. By Def. 4, the method definition of m at $\mathbf{c}$ may not exist. In this case, the definition of m at the smallest superclass of $\mathbf{c}$ is "inherited" by $\mathbf{c}$. The inherited method definition is called *resolution* and defined as follows:

Definition 6 (Resolution) *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* $\mathbf{c}' \leq \mathbf{c}''$*. The* resolution $Res(m_{\mathrm{b}}(\mathbf{c}))$ *of* m_{b} *at* $\mathbf{c}$ *is defined 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. Suppose that $(m_{\mathrm{c}}(\mathbf{c}'), t') \in \Sigma_{\mathrm{c}}$ *is the composite method definition of* m_{c} *at the smallest* $\mathbf{c}'$ *above* $\mathbf{c}$*. Then,* $Res(m_{\mathrm{c}}(\mathbf{c}))$ *is defined as* t'*. If such a unique composite method definition does not exist, then* $Res(m_{\mathrm{c}}(\mathbf{c}))$ *is* undefined*, denoted* $\perp$*.*

Example 7 *Consider schema* S_1 *shown in Fig. 3. By Def. 6,* $Res($location$($Server$))$ $=$ Room*. In other words, class* Server *inherits method definition* $($location$($Host$),$ Room$) \in \Sigma_{\mathrm{b}}$*. On the other hand,* $Res($boss$($Server$)) = \perp$ *since no superclass of* Server *has a definition of* boss*.*

2.4 Database Instance

A database instance of a method schema assigns a set of objects to each class name. Also, it gives the semantics of base methods.

Definition 8 (Database instance) *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 $m_{\mathrm{b}} \in M_{\mathrm{b},n}$, $\mu(m_{\mathrm{b}})$ 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 $Res(m_{\mathrm{b}}(\mathbf{c})) = c'$, then $\mu(m_{\mathrm{b}}) \mid_{\nu(\mathbf{c})}$ is a total mapping to $\bigcup_{c \leq c'} \nu(c)$, where "$|$" denotes that the domain of $\mu(m_{\mathrm{b}})$ is restricted to $\nu(\mathbf{c})$.

   (b) If $Res(m_{\mathrm{b}}(\mathbf{c})) = \perp$, then $\mu(m_{\mathrm{b}})$ is undefined everywhere in $\nu(\mathbf{c})$.

   If $\mu(m)(\mathbf{o})$ is undefined, then we write $\mu(m)(\mathbf{o}) = \perp$.

In the above definition, $\nu(c)$'s are defined to be disjoint. This definition can be easily modified so that $\nu(c) \subseteq \nu(c')$ for any c and c' such that $c \leq c'$. However, as discussed later, we are often interested in the most specific (smallest) class of a given object. Hence, it is preferable that $\nu(c)$'s are defined to be disjoint.

2.5 Method Execution

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 innermost reduction strategy, is defined as follows:

Definition 9 (Method execution) *For a term $t \in T_M(O_I)$, let $m(\mathbf{o})$ ($\mathbf{o} \in \nu(\mathbf{c})$) be a subterm of t at position r.*

(1) If $m \in M_{\mathrm{b}}$ and $\mu(m)(\mathbf{o}) \neq \perp$, then $t \rightarrow_I t[r \leftarrow \mu(m)(\mathbf{o})]$.

(2) If $m \in M_{\mathrm{c}}$ and $Res(m(\mathbf{c})) = t' \neq \perp$, then $t \rightarrow_I t[r \leftarrow t'[\mathbf{o}/\mathbf{x}]]$.

Note that taking the innermost reduction strategy (i.e., rewriting only the term in the form of $m(\mathbf{o})$) is essential since the definition of m cannot be bound before knowing the classes of the arguments of m.

Let $\rightarrow_I^*$ be the reflexive and transitive closure of $\rightarrow_I$. The *execution result* of t, denoted $t{\downarrow_I}$, is a term t' such that $t \rightarrow_I^* t'$ and there exists no t'' such that $t' \rightarrow_I t''$. Since $\rightarrow_I$ has Church-Rosser property [17], the execution result is uniquely determined. If $t{\downarrow_I} \in O_I$, then the execution of t is *successful*, and if $t{\downarrow_I} \notin O_I$, then the execution of t is *aborted*. In both cases (i.e., if $t{\downarrow_I}$ exists), the execution of t is *terminating*. On the other hand, if $t{\downarrow_I}$ does not exist, then the execution of t is *nonterminating*. We omit the subscript I of $\rightarrow_I$, $\rightarrow_I^*$, and $\downarrow_I$ if I is irrelevant or obvious from the context.

Example 10 *An example of a database instance* $I_1 = (\nu_1, \mu_1)$ *of* S_1 *is shown in Fig. 4.* ν_1 *is represented by gray rectangles, e.g.,* $\nu_1(\mathsf{Employee}) = \{\mathsf{Alice}, \mathsf{John}\}$. μ_1 *is represented by arrows, e.g.,* $\mu_1(\mathsf{boss})(\mathsf{John}) = \mathsf{Alice}$, $\mu_1(\mathsf{computer})(\mathsf{John}) = \mathsf{mars}$. *By Def. 9,* $\mathsf{supervisor}(\mathsf{Alice})$ *is executed as follows:*

$$\begin{aligned}\mathsf{supervisor}(\mathsf{Alice}) &\to_{I_1} \mathsf{supervisor}(\mathsf{boss}(\mathsf{Alice}))\\ &\to_{I_1} \mathsf{supervisor}(\mathsf{Sara})\\ &\to_{I_1} \mathsf{boss}(\mathsf{Sara})\\ &\to_{I_1} \mathsf{Bob}.\end{aligned}$$

Thus $\mathsf{supervisor}(\mathsf{Alice})\downarrow_{I_1} = \mathsf{Bob}$.

3 The Security Problem

3.1 Authorization

Various sophisticated method-based authorization models for OODBs have been proposed [3,4]. In this paper, however, discussing authorization models is not our main purpose, and therefore we adopt the following simple but general method-based authorization model.

Definition 11 (Authorization) *Let* $S = (C, \leq, M, \Sigma_{\mathrm{b}}, \Sigma_{\mathrm{c}})$. *A* right *is a term in the form of* $m(\mathbf{c})$, *where* $m \in M_n$ *and* $\mathbf{c} \in C^n$. *An* authorization A *is a finite set of rights and is interpreted as follows. Suppose that a user requests to directly invoke a method* m *on a tuple* $\mathbf{o}$ *of objects. Let* $\mathbf{c}$ *be the tuple of the classes such that* $\mathbf{o} \in \nu(\mathbf{c})$. *If* $m(\mathbf{c}) \in A$, *then the invocation is permitted. Otherwise, it is prohibited.*

An authorization is often modeled as a pair of a base authorization and a set of inference rules. An example of an 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 12 *Define an authorization* A_1 *for a user* u *under* S_1 *in Fig. 3 as follows:*

$$\begin{aligned}A_1 = \{&\mathsf{computer}(\mathsf{Employee}),\\ &\mathsf{supervisor}(\mathsf{Employee}),\ \mathsf{supervisor}(\mathsf{Manager}),\\ &\mathsf{office}(\mathsf{Employee}),\ \mathsf{office}(\mathsf{Manager})\}.\end{aligned}$$

Consider the instance I_1 in Fig. 4. Executing $\mathsf{office}(\mathsf{John})$ *by u is permitted since* $\mathsf{John} \in \nu_1(\mathsf{Employee})$ *and* $\mathsf{office}(\mathsf{Employee}) \in A_1$*. On the other hand, executing* $\mathsf{computer}(\mathsf{Sara})$ *is prohibited since* $\mathsf{Sara} \in \nu_1(\mathsf{Manager})$ *but* $\mathsf{computer}(\mathsf{Manager}) \notin A_1$.

3.2 Inference Attacks and the Security Problem

In this section, we formalize inference attacks (Def. 15). We generally assume that user's knowledge is modeled as a set of (in)equalities (Section 3.2.1). For example, suppose that a user u executes $\mathsf{office}(\mathsf{John})$ and obtains the result $\mathsf{A626}$. The information that u obtains is $\mathsf{office}(\mathsf{John})\!\downarrow = \mathsf{A626}$. Then, we restrict the power of the user's inference in a reasonable way and demonstrate that user's inference is modeled as equational reasoning (Section 3.2.2). Section 3.2.3 states formal definitions. In Def. 15, how to perform equational reasoning is defined as term rewriting rules. Then, in Def. 17, we define the security problem. Let τ be a term representing the query to be attacked. This term can be computed as it is if the user has authorization to all its methods. Even if the user does not have authorization to some methods in τ, there may exist an indirect way to compute τ by equivalently rewriting τ to another term involving only authorized methods. The term rewriting rules defined in Def. 15 are the rules that represent how to compute terms in a direct and/or indirect way. Thus, if τ cannot be rewritten into an object by the rewriting rules, τ is said to be secure.

3.2.1 General Attacker Model

First of all, we define the equalities which u can obtain directly from the execution results of authorized methods and their resolutions (Def. 6) as follows:

($*$1) User u knows $m(\mathbf{o})\!\downarrow = o$ *iff* $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, 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 knows $Res(m(\mathbf{c})) = t$ *iff* $m(\mathbf{c}) \in A$ 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. Also suppose that user u knows that $o \neq o'$ for distinct objects o and o' (e.g., u knows $\mathsf{John} \neq \mathsf{Alice}$, $\mathsf{Sara} \neq \mathsf{A626}$, and so on). Then, to infer new knowledge, u can use at least four inference rules on equalities: reflexivity, symmetry, transitivity, and substitutivity (i.e., if $t_i = t'_i$ for all i, then $f(\mathbf{t}) = f(\mathbf{t}')$).

3.2.2 Assumptions on the Attacker Model

To make the attacker model theoretically simple, we would like to assume the following two conditions:

(Q1) User u can use no inference rules other than reflexivity, symmetry, transitivity, and substitutivity. In other words, the user's knowledge is the congruence closure of the direct knowledge ($*1$) and ($*2$).
(Q2) The knowledge of u is represented by a set of ground equalities (i.e., equalities without variables).

In what follows, we demonstrate that assuming these conditions is reasonable.

(Q1) mentions the inference power of the user. We must exclude at least trivial cases where the user can use other inference rules. Let us examine one trivial case illustrated in the following example:

Example 13 *Recall the second case of Example 1, where u cannot infer $\mathsf{computer}(\mathsf{John})\downarrow = \mathsf{mars}$ since there may be another host h such that $\mathsf{computer}(\mathsf{John})\downarrow = h$ and $\mathsf{location}(h)\downarrow = \mathsf{A626}$. However, if u knows that $\mathsf{location}(o)\downarrow \neq \mathsf{A626}$ for* any other object o in the database instance*, then u can conclude that* $\mathsf{computer}(\mathsf{John}) = \mathsf{mars}$.

In this example, u uses an inference rule such that an equality is inferred from the contents of O_I and a set of inequalities. However, such inference becomes impossible if u does not know what O_I is. Practically, just hiding O_I from the user is sufficient for making the inference using inequalities impossible.

(Q2) requires that non-ground equalities obtained by ($*2$) can be translated into equivalent ground equalities. The following example suggests when such translation is possible.

Example 14 *Consider a schema with a composite method m_{c} which has the same resolution t at every class $c \in C$. Let $A = \{m_{\mathrm{c}}(c) \mid c \in C\}$ be an authorization for a user u.*

Assume that u knows what C is. Then, u can infer that $m_{\mathrm{c}}(t')\downarrow = t[t'/x]\downarrow$ for any term t' (no matter whether $t'\downarrow \in O_I$ or not), 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. In other words, u can substitute any term to the variable x in $m_{\mathrm{c}}(x)\downarrow = t\downarrow$.

On the other hand, if u does not know what C is, then u cannot conclude that $m_{\mathrm{c}}(t')\downarrow = t[t'/x]\downarrow$ without exactly inferring the class to which $t'\downarrow$ belongs (or inferring that the execution of t' is aborted) since there may be another class c in C such that $t'\downarrow \in \nu(c)$ and $Res(m_{\mathrm{c}}(c)) \neq t$. However, since type inference [6,7] *is useless when u does not know what C is, to know the class to which $t'\downarrow$ belongs is to infer the exact value of $t'\downarrow$. Consequently, u can substitute only an object of class c to the variable x in $m_{\mathrm{c}}(x)\downarrow = t\downarrow$.*

Thus, when C is hidden from u, each equality $Res(m_{\mathrm{c}}(\mathbf{c})) = t$ obtained by ($*$2) can be translated into $\{m_{\mathrm{c}}(\mathbf{o})\downarrow = t[\mathbf{o}/\mathbf{x}]\downarrow \mid \mathbf{o} \in \nu(\mathbf{c})\}$, which is a finite set of ground equalities.

In summary, assuming (Q1) and (Q2) is practically reasonable. This means that user's inference can be defined as the congruence closure (by (Q1)) of a finite set of ground equalities (by (Q2)) 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 [17], $t\downarrow = o$ *iff* t is reducible to o by $\rhd_{I,A}$.

3.2.3 Formal Definitions

Now, we provide formal definitions of inference attacks and the security problem.

Definition 15 (Inference attacks) *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 $m_{\mathrm{c}}(\mathbf{c}) \in A$, $m_{\mathrm{c}} \in M_{\mathrm{c}}$, $\mathbf{o} \in \nu(\mathbf{c})$, $m_{\mathrm{c}}(\mathbf{o})\downarrow = o \in O_I$, and $Res(m_{\mathrm{c}}(\mathbf{c})) = t \neq \bot$, 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

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

This simulates Knuth-Bendix completion procedure and, roughly speaking, corresponds to symmetry.

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 $t \to_I^* o$.

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 $t' = t[r'' \leftarrow o'']$ (This* *corresponds to substitutivity). Let $\Rightarrow^*_{I,A}$ denote the reflexive and transitive closure of $\Rightarrow_{I,A}$ (This corresponds to reflexivity and transitivity). For readability, we often write $\rhd_I$ and P_I instead of $\rhd_{I,A}$ and $P_{I,A}$, respectively.*

Example 16 *For S_1 in Fig. 3, I_1 in Fig. 4, and A_1 in Example 12, P_{I_1,A_1} is computed as shown in Fig. 5. Rules (A1)–(A10) are obtained by Def. 15(A), and (B1)–(B8) by Def. 15(B) with composite methods* $\mathsf{supervisor}$ *and* office*. Rules (C1) and (C2) are obtained by Def. 15(C). For example, (C1) is derived from (A1) and (B5).*

Rule (C1) indicates that the user can infer that $\mathsf{location}(\mathsf{mars}){\downarrow} = \mathsf{A626}$*, as stated in the first case of Example 1. Moreover, rule (C2) indicates that* $\mathsf{location}$ *even for a server* $\mathsf{jupiter}$ *can be inferred. Let $\tau = \mathsf{office}(\mathsf{boss}(\mathsf{Sara}))$ and $\tau' = \mathsf{office}(\mathsf{boss}(\mathsf{John}))$. Then,*

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

Thus user u can infer that $\tau{\downarrow} = \mathsf{B533}$. On the other hand, u cannot infer the value of $\tau'{\downarrow}$ (although $\tau'{\downarrow} = \mathsf{B533}$) since no subterm of τ' can be rewritten by the rules in P_{I_1,A_1}.

Definition 17 (The security problem) *A term $\tau \in T_M(X)$ is said to be* secure *at a tuple $\mathbf{c}$ of classes under a schema S and an authorization A if there exists no instance $I = (\nu, \mu)$ of S such that $\tau[\mathbf{o}/\mathbf{x}] \Rightarrow^*_{I,A} o$ for any $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in O_I$. Otherwise, τ is* insecure *at $\mathbf{c}$ under S and A. The* security problem *is to determine whether a given τ is secure at a given $\mathbf{c}$ under given S and A.*

4 General Case

4.1 Undecidability of the Security Problem for General Schemas

We show that the security problem is undecidable by reducing the Modified Post's Correspondence Problem (MPCP) [18] to the security problem. The reduction strategy was obtained by modifying that of the proof of the undecidability of the type-consistency problem [8].

Let $(\mathbf{w}, \mathbf{u})$ be an instance of the MPCP over alphabet $\Sigma = \{0, 1\}$, where $\mathbf{w} = (w_1, \ldots, w_n)$, $\mathbf{u} = (u_1, \ldots, u_n)$, and $w_i, u_i \in \Sigma^*$. A solution of $(\mathbf{w}, \mathbf{u})$ is a finite sequence $(i_1, i_2, \ldots, i_k)$ of indices such that $w_{i_1} w_{i_2} \cdots w_{i_k} = u_{i_1} u_{i_2} \cdots u_{i_k}$ and $i_1 = 1$. In what follows, we construct a schema $S_{\mathbf{w},\mathbf{u}}$, a term τ, and an authorization A such that $(\mathbf{w}, \mathbf{u})$ has a solution *iff* there exists a database instance I of $S_{\mathbf{w},\mathbf{u}}$ under which the execution result of τ can be inferred.

$S_{\mathbf{w},\mathbf{u}}$ has classes c, c_1,..., c_n, c'_0, c'_1, c_{ok}, and c_{dummy}, where $c_i \leq c$ for each $1 \leq i \leq n$, $c'_0 \leq c_{\text{ok}}$, and $c'_1 \leq c_{\text{ok}}$. Each c_i ($1 \leq i \leq n$) represents the index i of $\mathbf{w}$ and $\mathbf{u}$. c'_0 and c'_1 represent symbols 0 and 1 in Σ, respectively. Classes c and c_{ok} represent the "separators" and "end markers" in database instances, respectively, as explained in the example below. Class c_{dummy} represents dummy execution results.

$S_{\mathbf{w},\mathbf{u}}$ has unary base methods next and dummy, unary composite methods isw, isw$_{i,j}$, isu, and isu$_{i,j}$, and a binary base method post. Their method definitions are constructed from $(\mathbf{w}, \mathbf{u})$ as follows. First, the definition of method next is:

$$\begin{aligned}
&(\mathsf{next}(c_i), c) \quad \text{for each } 1 \leq i \leq n,\\
&(\mathsf{next}(c), c_{\text{ok}}),\\
&(\mathsf{next}(c'_0), c_{\text{ok}}),\\
&(\mathsf{next}(c'_1), c_{\text{ok}}).
\end{aligned}$$

A pair of a database instance (more precisely, the semantics of next) and an object of class c_1 is regarded to represent two things, a candidate $(i_1, \ldots, i_k)$ for a solution of $(\mathbf{w}, \mathbf{u})$ and a string s over Σ, as illustrated in the following example.

Example 18 *Consider the following instance* $(\mathbf{w}, \mathbf{u})$ *of the MPCP:*

$$\begin{aligned}
w_1 &= 101, & u_1 &= 1,\\
w_2 &= 00, & u_2 &= 100,\\
w_3 &= 11, & u_3 &= 011.
\end{aligned}$$

A typical database instance I_2 of $S_{\mathbf{w},\mathbf{u}}$ is shown in Fig. 6. In the figure, method next *is represented by arrows. Pair (I_2, o_1) represents the following candidate and string. First, the candidate is represented by the sequence of objects from o_1 to the "separator" object o of class c (the upper half of the figure). In the figure, the objects are of classes c_1, c_3, and c_2, so the candidate is* $(1, 3, 2)$. *On the other hand, the string is represented by the sequence of objects between the "end marker" object o_{ok} of class c_{ok} and o (the lower half of the figure). In the figure, the objects are of classes c'_1, c'_0, c'_1, c'_1, c'_1, c'_0, and c'_0 (in the reverse order with respect to* next*), so the represented string is* 1011100.

On the other hand, pair (I'_2, o'_1) shown in Fig. 7 represents no candidates since no object of class c is "reachable" from o'_1 under I'_2.

Suppose that (I, o_1) represents a candidate $(i_1, \ldots, i_k)$ and a string s as is the case of Fig. 6 (s is regarded as an infinite string if no "end marker" object of class c_{ok} is reachable from the "separator" object o). To check whether the candidate is actually a solution, we examine whether both $w_{i_1} \cdots w_{i_k} = s$ and $u_{i_1} \cdots u_{i_k} = s$. Unary composite methods isw and isu are used for that purpose. Let $w_i = w_{i,1}w_{i,2} \cdots w_{i,d_i}$

for each i ($1 \leq i \leq n$), where $w_{i,j} \in \Sigma$. The definitions of method isw and its auxiliary composite methods $\mathsf{isw}_{i,j}$ are constructed as follows:

$$
\begin{aligned}
&(\mathsf{isw}(c_i), \mathsf{isw}_{i,1}(\cdots \mathsf{isw}_{i,d_i}(\mathsf{isw}(\mathsf{next}(x))))),\\
&(\mathsf{isw}(c), \mathsf{next}(x)),\\
&(\mathsf{isw}_{i,j}(c'_0), \mathsf{next}(x)) \quad \text{if } w_{i,j} = 0,\\
&(\mathsf{isw}_{i,j}(c'_1), \mathsf{next}(x)) \quad \text{if } w_{i,j} = 1,\\
&(\mathsf{isw}_{i,j}(c_{\mathrm{ok}}), \mathsf{dummy}(x)),\\
&(\mathsf{isw}_{i,j}(c_{\mathrm{dummy}}), x),
\end{aligned}
$$

where unary base method dummy is defined so that it always returns an object of class c_{dummy}. If $w_{i_1} \cdots w_{i_k} = s$, then $\mathsf{isw}(o_1)$ returns an object of class c_{ok}. Otherwise, $\mathsf{isw}(o_1)$ returns an object of a class other than c_{ok}. Note that if (I, o_1) represents no candidate as is the case of Fig. 7, then the execution of $\mathsf{isw}(o_1)$ is nonterminating under I. Method isu is defined in the same way.

Example 19 *For the instance* $(\mathbf{w}, \mathbf{u})$ *of the MPCP in Example 18, method* isw *and its auxiliary methods are defined as follows:*

$$
\begin{aligned}
&(\mathsf{isw}(c_1), \mathsf{isw}_{1,1}(\mathsf{isw}_{1,2}(\mathsf{isw}_{1,3}(\mathsf{isw}(\mathsf{next}(x)))))),\\
&(\mathsf{isw}(c_2), \mathsf{isw}_{2,1}(\mathsf{isw}_{2,2}(\mathsf{isw}(\mathsf{next}(x))))),\\
&(\mathsf{isw}(c_3), \mathsf{isw}_{3,1}(\mathsf{isw}_{3,2}(\mathsf{isw}(\mathsf{next}(x))))),\\
&(\mathsf{isw}(c), \mathsf{next}(x)),\\
&(\mathsf{isw}_{1,1}(c'_1), \mathsf{next}(x)), \quad (\mathsf{isw}_{1,2}(c'_0), \mathsf{next}(x)), \quad (\mathsf{isw}_{1,3}(c'_1), \mathsf{next}(x)),\\
&(\mathsf{isw}_{2,1}(c'_0), \mathsf{next}(x)), \quad (\mathsf{isw}_{2,2}(c'_0), \mathsf{next}(x)),\\
&(\mathsf{isw}_{3,1}(c'_1), \mathsf{next}(x)), \quad (\mathsf{isw}_{3,2}(c'_1), \mathsf{next}(x)),\\
&(\mathsf{isw}_{i,j}(c_{\mathrm{ok}}), \mathsf{dummy}(x)) \quad \textit{for any pair } (i,j),\\
&(\mathsf{isw}_{i,j}(c_{\mathrm{dummy}}), x) \quad \textit{for any pair } (i,j).
\end{aligned}
$$

The execution of $\mathsf{isw}(o_1)$ *under* I_2 *in Fig. 6 is as follows:*

$$
\begin{aligned}
\mathsf{isw}(o_1) &\to^*_{I_2} \mathsf{isw}_{1,1}(\mathsf{isw}_{1,2}(\mathsf{isw}_{1,3}(\mathsf{isw}_{3,1}(\mathsf{isw}_{3,2}(\mathsf{isw}_{2,1}(\mathsf{isw}_{2,2}(\mathsf{isw}(o))))))))\\
&\to^*_{I_2} o_{\mathrm{ok}}.
\end{aligned}
$$

Lastly, binary base method post is defined so that it always returns an object of class c_{dummy} regardless of its arguments.

Define authorization A as

$$A = \{\mathsf{post}(c_{\mathrm{ok}}, c_{\mathrm{ok}}), \mathsf{isw}(c_1), \mathsf{isu}(c_1)\}.$$

Let $\tau = \mathsf{post}(\mathsf{isw}(x), \mathsf{isu}(x))$.

Lemma 20 *τ is insecure at c_1* iff $(\mathbf{w}, \mathbf{u})$ *has a solution.*

PROOF. *Only if part.* τ involves method post. Since methods isw and isu never invokes post, in order for the user to know the execution result of τ, the user must invoke method post directly. By the definition of A, only $\mathsf{post}(c_{\text{ok}}, c_{\text{ok}})$ is authorized. Therefore, if τ is insecure at c_1, there must be a pair (I, o_1) such that the execution results of both $\mathsf{isw}(o_1)$ and $\mathsf{isu}(o_1)$ are objects of class c_{ok}. Such (I, o_1) represents a solution of $(\mathbf{w}, \mathbf{u})$.

If part. Suppose that $(\mathbf{w}, \mathbf{u})$ has a solution. Then, there must be a pair (I, o_1) such that the execution results of both $\mathsf{isw}(o_1)$ and $\mathsf{isu}(o_1)$ are objects of class c_{ok}. Therefore, the user can simply execute $\tau[o_1/x]$ and obtain the execution result. That is, τ is insecure at c_1. $\square$

Thus, we have the following theorem:

Theorem 21 *The security problem is undecidable.*

Note that the undecidability holds even if the "height" of the class hierarchy is one. On the other hand, in our conjecture, the security problem is decidable if the "height" of the class hierarchy is zero. The proof will be similar to the one of Theorem 1 in [8], which shows the decidability of the type-consistency problem for schemas with the height of the class hierarchy zero.

4.2 A Decidable Sufficient Condition for the Security

In this section we propose a decidable sufficient condition for a given term $\tau \in T_M(X)$ to be secure at $\mathbf{c}$. The main idea is to use classes instead of objects for analyzing the security. To do so, we introduce new rewriting rules on $T_M(C)$ which "conservatively" approximate $\rhd_{I,A}$, i.e., if τ is insecure at $\mathbf{c}$, then $\tau[\mathbf{c}/\mathbf{x}]$ is reducible to a class c by the new rewriting rules. Intuitively, each $t[\mathbf{c}/\mathbf{x}] \in T_M(C)$ is considered as the set of instantiated terms $t[\mathbf{o}/\mathbf{x}]$ such that $\mathbf{o} \in \nu(\mathbf{c})$. In order to compute the "execution result" of $t[\mathbf{c}/\mathbf{x}]$, the result E_S of *type inference* would be useful, where E_S is defined as follows: $c \in E_S(t, \mathbf{c})$ *iff* there is a database instance $I = (\nu, \mu)$ of S such that $t[\mathbf{o}/\mathbf{x}]{\downarrow_I} \in \nu(c)$ for some $\mathbf{o} \in \nu(\mathbf{c})$.

In what follows, we first summarize the known results on type inference for method schemas. Next, new rewriting rules for conservatively approximating inference attacks are introduced. Then, a sufficient condition for the security is proposed (Theorem 25) and its correctness and complexity are discussed.

4.2.1 Known Results on Type Inference

Unfortunately, E_S is uncomputable in general [6]. However, it is possible to compute an over-estimation $Z : T_M(C) \to 2^C$ of E_S, that is, $Z(t[\mathbf{c}/\mathbf{x}]) \supseteq E_S(t, \mathbf{c})$ for every pair of t and $\mathbf{c}$. The algorithm in [7] gives an over-estimation of E_S by computing the least fixpoint of $\hat{Z}$ satisfying the following four kinds of equations:

- For each $c \in C$, $\hat{Z}(c) = \{c\}$;
- For each pair $(m_{\mathrm{b}}(\mathbf{c}), c')$ such that $Res(m_{\mathrm{b}}(\mathbf{c})) = c'$, $\hat{Z}(m_{\mathrm{b}}(\mathbf{c})) = \{c \mid c \leq c'\}$;
- For each pair $(m_{\mathrm{c}}(\mathbf{c}), t)$ such that $Res(m_{\mathrm{c}}(\mathbf{c})) = t$, $\hat{Z}(m_{\mathrm{c}}(\mathbf{c})) = \hat{Z}(t[\mathbf{c}/\mathbf{x}])$;
- For every term $m(t_1, \ldots, t_n) \in T_M(X)$ and any tuples $\mathbf{c}_1, \ldots, \mathbf{c}_n$ of classes,

$$\hat{Z}(m(t_1[\mathbf{c}_1/\mathbf{x}_1], \ldots, t_n[\mathbf{c}_n/\mathbf{x}_n])) = \bigcup_{\mathbf{c}' \in \hat{Z}(t_1[\mathbf{c}_1/\mathbf{x}_1]) \times \cdots \times \hat{Z}(t_n[\mathbf{c}_n/\mathbf{x}_n])} \hat{Z}(m(\mathbf{c}')).$$

Let Z be the least fixpoint of $\hat{Z}$. Also let t be an arbitrary term in $T_M(X)$. It is guaranteed in [7] that $Z(t[\mathbf{c}/\mathbf{x}]) \supseteq E_S(t, \mathbf{c})$. Moreover, if S contains only unary methods, then Z is identical with E_S.

Example 22 *Using the algorithm in [7], we can compute Z for schema S_1 in Fig. 3. The result is presented in Fig. 8. For example, $Z(\mathsf{boss}(\mathsf{Employee})) = \{\mathsf{Employee}, \mathsf{Manager}\}$ 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_{S_1} since S_1 contains only unary methods.*

4.2.2 Conservative Approximation of Inference Attacks

In order to approximate inference attacks, we use an over-estimation Z of E_S since E_S is uncomputable. The smaller $Z(t[\mathbf{c}/\mathbf{x}])$ is, the better approximation we have, although the approximation is still conservative even when $Z(t[\mathbf{c}/\mathbf{x}]) = C$ for every pair of t and $\mathbf{c}$. In the next definition, the class-level inference, which approximates the object-level inference $\rhd_{I,A}$, is defined as inference rules $\rhd_{S,A,Z}$ on $T_M(C)$. The definition is similar to the definition of $\rhd_{I,A}$ (Def. 15) except that the objects are replaced with the possible classes indicated by Z. Since Z is an over-estimation of E_S, all the object-level inference is captured by the corresponding class-level inference (see Theorem 25 below for formal discussion), but the converse is not necessarily true.

Definition 23 (Approximation of inference attacks) *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 $m_c(\mathbf{c}) \in A$, $m_c \in M_c$, *and* $Res(m_c(\mathbf{c})) = t \neq \perp$, *then* $P_{S,A,Z}$ *contains*

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

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 24 *Fig. 9 presents the contents of* $P_{S_1,A_1,Z}$ *for schema* S_1 *in Fig. 3,* A_1 *in Example 12, and* Z *in Fig. 8. Rules (Ai)–(Avi) are obtained by Def. 23(A), and (Bi)–(Biv) by Def. 23(B) with composite methods* supervisor *and* office. *Rules (Ci) and (Cii) are obtained by Def. 23(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 16.

Next, consider a looser estimation Z', *which is identical to* Z *except that*

$$Z'(\mathsf{supervisor}(\mathsf{Employee})) = \{\mathsf{Employee}, \mathsf{Manager}\}.$$

Then, $P_{S_1,A_1,Z'}$ *contains the following two rules as well as all the rules in* $P_{S_1,A_1,Z}$:

$$\mathsf{supervisor}(\mathsf{Employee}) \rhd_{S_1} \mathsf{Employee}, \qquad \text{(Aiii}'\text{)}$$
$$\mathsf{supervisor}(\mathsf{boss}(\mathsf{Employee})) \rhd_{S_1} \mathsf{Employee}. \qquad \text{(Bi}'\text{)}$$

Rules (Aiii′) and (Ai) together indicate that the user may be able to infer the computer of the supervisor of an employee. However, it is impossible to do so because the supervisor of an employee is always a manager and computer(Manager) *is unauthorized by* A_1. *In this sence,* $P_{S_1,A_1,Z'}$ *is a worse approximation than* $P_{S_1,A_1,Z}$.

4.2.3 The Proposed Sufficient Condition

The proposed sufficient condition for the security is stated as the following theorem:

Theorem 25 *Let $\tau \in T_M(X)$. If there exists no class c such that $\tau[\mathbf{c}/\mathbf{x}] \Rightarrow^*_{S,A,Z} c$, then τ is secure at* $\mathbf{c}$*, i.e., there exists no instance $I = (\nu, \mu)$ such that $\tau[\mathbf{o}/\mathbf{x}] \Rightarrow^*_{I,A} o$ for any $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in O_I$.*

An overview of the correctness of the proposed sufficient condition is illustrated in Fig. 10. We first prove that each rule in P_I is conservatively approximated by a rule in P_S.

Lemma 26 *If there is an instance $I = (\nu, \mu)$ 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 structure of the definition of $\rhd_I$ (see Def. 15).

Basis. Consider the case that $m(\mathbf{o}) \rhd_I o$ $(o \in \nu(c))$ is obtained from Def. 15(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. 23(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. 15(B) can be treated 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. 15(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. 23(C), we can conclude that $t'[\mathbf{c}'/\mathbf{x}'] \rhd_S c \in P_S$. $\square$

By Lemma 26, 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 $t[\mathbf{c}/\mathbf{x}] \Rightarrow^*_S t'[\mathbf{c}'/\mathbf{x}']$. Theorem 25 is implied by this fact.

Example 27 *Consider schema S_1 in Fig. 3, and let $\tau = \mathsf{office}(\mathsf{boss}(x))$. We can conclude that τ is secure at* Employee *since no subterm of $\tau[\mathsf{Employee}/x]$ can be rewritten by any rule $P_{S_1,A_1,Z}$ in Fig. 9.*

Example 28 *We said that* $\mathsf{computer}(x)$ *is secure at* Employee *in the second case of Example 1. Actually, it is not difficult to see that P_S has only* $\mathsf{location}(\mathsf{Host}) \rhd_S \mathsf{Room}$ *and* $\mathsf{office}(\mathsf{Employee}) \rhd_S \mathsf{Room}$. *This implies that* $\mathsf{computer}(x)$ *is secure at* Employee.

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. In what follows, we summarize the time complexity of deciding the sufficient condition. Define the size of a term t as $|Pos(t)|$, i.e., the number of positions of t. Define the description length of Σ_{c}, denoted $\|\Sigma_{\mathrm{c}}\|$, as the sum of the size of all t such that $(m(\mathbf{c}), t) \in \Sigma_{\mathrm{c}}$. Also, define the size of S, denoted $\|S\|$, as follows:

$$\|S\| = |C| + |\leq| + |M| + |\Sigma_{\mathrm{b}}| + \|\Sigma_{\mathrm{c}}\|.$$

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

5 Linear Case

5.1 Type Inferability of Linear Schemas

A schema S is *linear* if for every composite method definition $(m_{\mathrm{c}}(\mathbf{c}), t)$ in S, t is linear. We show that linear terms are type inferable under linear schemas, which is an improvement of the known result in [7].

Theorem 29 *$E_S(t, \mathbf{c})$ is computable if both S and t are linear.*

PROOF. Let S be a linear schema. We introduce a *syntactic instance* $I_S = (\nu_S, \mu_S)$ of S as follows. Let N be a sufficiently large positive integer.

(1) For each $c \in C$, define

$$\nu_S(c) = \{c \cdot \alpha \mid \alpha \in C^* \text{ and the length of } c \cdot \alpha \text{ is at most } N\}.$$

Here, 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) Suppose that $Res(m_{\mathrm{b}}(c_1, c_2 \ldots c_n)) = c'$. Then, for any $o_i \in \nu_S(c_i)$ $(2 \leq i \leq n)$, define $\mu_S(m_{\mathrm{b}})(c_1, o_2, \ldots, o_n) = c'$. Moreover, for $l \geq 1$,

$$\mu_S(m_{\mathrm{b}})(c_1 \cdot c'_1 \cdot c'_2 \cdots c'_l, o_2, \ldots, o_n) = \begin{cases} c'_1 \cdot c'_2 \cdots c'_l \text{ if } c'_1 \leq c', \\ c' \cdot c'_2 \cdots c'_l \text{ otherwise.} \end{cases}$$

(b) Suppose that $Res(m_{\mathrm{b}}(c_1, c_2, \ldots c_n)) = \perp$. Then, for any $o_i \in \nu_S(c_i)$ $(1 \leq i \leq n)$, $\mu_S(m_{\mathrm{b}})(o_1, \ldots, o_n)$ is undefined.

The first (leftmost) symbol c of an object $c \cdot \alpha$ in a syntactic instance I_S represents the class of the object. For a base method m_{b} and objects $(o_1, \ldots, o_n) \in \nu_S(c_1, \ldots, c_n)$, $\mu_S(m_{\mathrm{b}})(o_1, \ldots, o_n)$ is the sequence obtained from the first argument o_1 by the following manipulation: remove the first symbol of $o_1 = c_1 \cdot c'_1 \cdot \alpha'$ and if the first symbol c'_1 of the remaining sequence $c'_1 \cdot \alpha'$ is not a subclass of the resolution c' of m_{b} at $(c_1, \ldots, c_n)$ then replace c'_1 with c' in order to meet the base method definition of m_{b}; the other arguments are simply discarded.

In what follows, we show that E_S can be computed by the algorithm in [7]. See the four kinds of equations in Section 4.2.1. Let Z be the least fixpoint of $\hat{Z}$. We already have $Z(t[\mathbf{c}/\mathbf{x}]) \supseteq E_S(t, \mathbf{c})$ by [7] for any linear term $t \in T_M(X)$, and therefore, it suffices to show the opposite containment. The set of objects of a syntactic instance contains all the sequences of classes of length less than or equal to N. Therefore, for any linear term $t \in T_M(X)$, there always exist objects of a syntactic instance (with a sufficiently large N) which exactly encode the fixpoint computation of $Z(t)$. In other words, for any linear term t with variables $\mathbf{x}$, there always exist objects $\mathbf{o}$ of classes $\mathbf{c}$ such that $c \in Z(t[\mathbf{c}/\mathbf{x}])$ implies $t[\mathbf{o}/\mathbf{x}] \rightarrow^*_{I_S} c \cdot \alpha$, which means that $t[\mathbf{o}/\mathbf{x}]{\downarrow_{I_S}}$ is an object of class c. This property can be proved by the induction on the structure of the definition of $\hat{Z}$.

- Consider $\hat{Z}(c)$. We have $\hat{Z}(c) = \{c\}$ and $c \cdot \alpha \rightarrow^*_{I_S} c \cdot \alpha$ for any $\alpha \in C^*$.
- Consider $\hat{Z}(m_{\mathrm{b}}(\mathbf{c}))$. Any $c \in \hat{Z}(m_{\mathrm{b}}(\mathbf{c}))$ must be a subclass of $Res(m_{\mathrm{b}}(\mathbf{c}))$. For any $\alpha \in C^*$, let $o_1 = c_1 \cdot c \cdot \alpha$ and $o_i = c_i$ $(2 \leq i \leq n)$. Then, $m_{\mathrm{b}}(\mathbf{o}) = m_{\mathrm{b}}(o_1, \ldots, o_n) \rightarrow_{I_S} c \cdot \alpha$ by the definition of μ_S.
- Suppose that $c \in Z(m_{\mathrm{c}}(\mathbf{c}))$. Then, there exists $t \in T_M(X)$ such that $Res(m_{\mathrm{c}}(\mathbf{c})) = t$ and $c \in Z(t[\mathbf{c}/\mathbf{x}])$. By the inductive hypothesis, for any $\alpha \in C^*$, there exists $\mathbf{o} \in \nu_S(\mathbf{c})$ such that $t[\mathbf{o}/\mathbf{x}] \rightarrow^*_{I_S} c \cdot \alpha$. Therefore, $m_{\mathrm{c}}(\mathbf{o}) \rightarrow_{I_S} t[\mathbf{o}/\mathbf{x}] \rightarrow^*_{I_S} c \cdot \alpha$.
- Suppose that $c \in Z(m(t_1[\mathbf{c}_1/\mathbf{x}_1], \ldots, t_n[\mathbf{c}_n/\mathbf{x}_n]))$. Then, there exists $\mathbf{c}' = (c'_1, \ldots, c'_n) \in Z(t_1[\mathbf{c}_1/\mathbf{x}_1]) \times \cdots \times Z(t_n[\mathbf{c}_n/\mathbf{x}_n])$ such that $c \in Z(m(\mathbf{c}'))$. By the inductive hypothesis, for any $\alpha \in C^*$, there exists $\mathbf{o}' = (o'_1, \ldots, o'_n) \in \nu_S(\mathbf{c}')$ such that $m(\mathbf{o}') \rightarrow^*_{I_S} c \cdot \alpha$. Note that by the definition of ν_S, $o'_i = c'_i \cdot \alpha'_i$ for some $\alpha'_i \in C^*$. By the linearity of $m(t_1, \ldots, t_n)$ and the inductive hypothesis again, for every $1 \leq i \leq n$ and for such α'_i, there exists $\mathbf{o}_i \in \nu_S(\mathbf{c}_i)$ such that $t_i[\mathbf{o}_i/\mathbf{x}_i] \rightarrow^*_{I_S} c'_i \cdot \alpha'_i$. Thus, for any α, there exist $\mathbf{o}_1, \ldots, \mathbf{o}_n$ such that $m(t_1[\mathbf{o}_1/\mathbf{x}_1], \ldots, t_n[\mathbf{o}_n/\mathbf{x}_n]) \rightarrow^*_{I_S} c \cdot \alpha$.

The theorem has been proved since $Z(t[\mathbf{c}/\mathbf{x}])$ is computable and $E_S(t, \mathbf{c}) = Z(t[\mathbf{c}/\mathbf{x}])$ if both S and t are linear. □

The decidability of the security of linear terms under linear schemas immediately follows the next theorem since E_S is computable under linear schemas. More concretely, the sufficient condition of the security proposed in the previous section becomes a necessary one if S and τ are linear and E_S is used as Z.

Theorem 30 *Let S be a linear schema and let τ be a linear term in $T_M(X)$. There is a class c' such that $\tau[\mathbf{c}/\mathbf{x}] \Rightarrow^*_{S,A,E_S} c'$* iff *$\tau$ is insecure at* $\mathbf{c}$.

It suffices to prove the only if part because the if part was shown by Theorem 25. In what follows, for a linear term $t \in T_M(X)$, we write $E_S(t, \sigma)$ to mean $E_S(t, \mathbf{c})$, where $t/r_i = x_i$ and $\sigma(r_i) = c_i$ for each i.

We show that a syntactic instance $I_S = (\nu_S, \mu_S)$ with sufficiently large N satisfies the theorem. The outline of the proof is as follows (see also Fig. 11). It suffices to show the soundness of the approximated inference attacks (i.e., that $\Rightarrow^*_{S,A,E_S}$ implies $\Rightarrow^*_{I_S,A}$) in linear case. This property can be proved by showing that $\rhd_{S,A,E_S}$ implies $\rhd_{I_S,A}$ (Lemma 34) and then by lifting it to rewrite sequences (Lemma 35). For Lemma 35, we need to show that the existence of an approximation rule in Def. 23(C) guarantees the existence of an inference rule in Def. 15(C) in a syntactic instance. This can be done with the help of Lemma 33, which states that type information E_S is closed under term composition.

The crucial point lies in constructing a substitution mapping into I_S which shows the soundness of the approximation in the proof of Lemma 34. Remember that only the first argument of a base method is used for constructing the returned value of the method in I_S. If a linear term t does not contain a composite method, it is easy to find the variable position which contributes the execution result of t (called the *principal position* as defined below). For example, for term $m_1(m_2(x_1, x_2), x_3)$ for base methods m_1 and m_2, the principal position is the position of x_1. For an arbitrary linear term t, the principal position can also be found although it needs a recursive search since composite methods may appear in t. Furthermore, we can choose objects that encode the fixpoint computation of E_S ($= Z$ in linear case) and substitute the objects to the variables in t to construct a term that simulates E_S in a syntactic instance. A mapping that specifies these objects is called a *consumed string mapping* (CSM for short), and is defined in parallel with the principal position. In the proof of Lemma 34, the substitution mapping into I_S will be constructed using CSMs and principal positions.

In what follows, CSMs and principal positions are defined. Their existence is shown in a constructive manner. Then, the soundness of the approximated inference attacks is shown using CSMs and principal positions.

5.2.1 Definitions of CSMs and principal positions

Definition 31 *Let t be a linear term in $T_M(X)$. Let $\sigma : V(t) \to C$ and $c' \in E_S(t, \sigma)$. A* consumed string mapping *(CSM for short) of (t, σ, c') is a mapping $\beta : V(t) \to C^+$ satisfying the following conditions:*

(1) The first (leftmost) symbol of $\beta(r)$ is $\sigma(r)$ for each $r \in V(t)$; and
(2) There is a position $\xi \in V(t)$, called the principal position *of (t, β), such that*
  (a) the last (rightmost) symbol of $\beta(\xi)$ is c', and
  (b) for any mapping $\theta_\beta : V(t) \to O_{I_S}$ such that $\theta_\beta(r) = \beta(r) \cdot \alpha_r$, where $\alpha_r \in C^*$, we have $t\theta_\beta \to^*_{I_S} c' \cdot \alpha_\xi$.

Here, C^+ denotes the positive Kleene closure of C. See also Fig. 12.

Example 32 *Consider a schema $S = (C, \leq, M, \Sigma_\mathrm{b}, \Sigma_\mathrm{c})$ with*

$$M_\mathrm{b} = \{(m_\mathrm{b}(c_1, c_2), c_3), (m'_\mathrm{b}(c_3), c_4)\},$$
$$M_\mathrm{c} = \{(m_\mathrm{c}(c_1, c_2), m_\mathrm{b}(x_2, x_1)))\}.$$

Let $t = m'_\mathrm{b}(m_\mathrm{b}(x_1, x_2))$. Let $\sigma : V(t) \to C$ be a substitution such that $\sigma(1 \cdot 1) = c_1$ and $\sigma(1 \cdot 2) = c_2$ (hence $t\sigma = m'_\mathrm{b}(m_\mathrm{b}(c_1, c_2))$). Then, $\beta : V(t) \to C^+$ with $\beta(1 \cdot 1) = c_1 \cdot c_3 \cdot c_4$ and $\beta(1 \cdot 2) = c_2$ is a CSM of (t, σ, c_4) and $1 \cdot 1$ is the principal position of (t, β) because for any $\alpha_{1\cdot 1}, \alpha_{1\cdot 2} \in C^*$,

$$m'_\mathrm{b}(m_\mathrm{b}(c_1 \cdot c_3 \cdot c_4 \cdot \alpha_{1\cdot 1}, c_2 \cdot \alpha_{1\cdot 2})) \to_{I_S} m'_\mathrm{b}(c_3 \cdot c_4 \cdot \alpha_{1\cdot 1}) \to_{I_S} c_4 \cdot \alpha_{1\cdot 1}.$$

Thus, β represents the prefixes of the object names that are "consumed" during the execution of t under I_S.

For another example, let $t' = m_\mathrm{c}(x_1, x_2)$. Let $\sigma' : V(t') \to C$ be a substitution such that $\sigma'(1) = c_2$ and $\sigma'(2) = c_1$. Then, $\beta' : V(t') \to C^+$ with $\beta'(1) = c_2$ and $\beta'(2) = c_1 \cdot c_3$ is a CSM of (t', σ', c_3) and 2 is the principal position of (t', β') because for any $\alpha_1, \alpha_2 \in C^*$,

$$m_\mathrm{c}(c_2 \cdot \alpha_1, c_1 \cdot c_3 \cdot \alpha_2) \to_{I_S} m_\mathrm{b}(c_1 \cdot c_3 \cdot \alpha_2, c_2 \cdot \alpha_1) \to_{I_S} c_3 \cdot \alpha_2.$$

Note that $c' \in E_S(t, \sigma) = E_S(t, \mathbf{c})$, one of the preconditions in Def. 31, is equivalent to $c' \in Z(t[\mathbf{c}/\mathbf{x}])$ by Theorem 29. Thus, $c' \in E_S(t, \sigma)$ means that fixpoint computation of $t[\mathbf{c}/\mathbf{x}]$ derives c'. By Theorem 29, this computation can be simulated by a syntactic instance I_S. For a CSM mapping β and a variable position r in t, $\beta(r)$ denotes the prefix of an object (i.e., a sequence of class names) substituted at r which is consumed during the execution of t to simulate the fixpoint computation.

5.2.2 Existence of CSMs and principal positions

In this section, we will effectively show the existence of a CSM β of (t, σ, c') and the principal position of (t, β). If t consists of only base methods, the story is easy to follow. By the definition of syntactic instances, each base method consumes the first symbol of its first argument. A CSM can be constructed by concatenating such consumed symbols. However, t may contain recursively-defined composite methods, and therefore, we cannot use the induction on the structure of t. Instead, we use the induction on the execution length. Since $c' \in E_S(t, \sigma)$, there must be a mapping $\theta : V(t) \to O_{I_S}$ such that $t\theta \to^*_{I_S} o' \in \nu_S(c')$ and $\theta(r) \in \nu_S(\sigma(r))$ for each $r \in V(t)$. The following proof is based on the induction on the execution length of $t\theta \to^*_{I_S} o'$.

Basis. Consider the zero-length reduction $x\theta = o' \in \nu_S(c')$. Then, a mapping β such that $\beta(\varepsilon) = c'$ is a CSM of (x, σ, c'). Also, ε is the principal position of (x, β).

Induction. Consider a reduction $t\theta \to_{I_S} t'\theta' \to^*_{I_S} o' \in \nu_S(c')$, where $\theta' : V(t') \to O_{I_S}$. Let $\sigma' : V(t') \to C$ be the mapping such that $\theta'(r') \in \nu_S(\sigma'(r'))$ for each $r' \in V(t')$. Also, let $q \in Pos(t)$ be the position contracted in the first step of this reduction, and let $m(\mathbf{x}'') = t/q$. Assume inductively that $\beta' : V(t') \to C^+$ is a CSM of (t', σ', c') and $\xi' \in V(t')$ is the principal position of (t', β').

(i) Suppose that m is a base method. Then, the following β is a CSM of (t, σ, c') (see also Fig. 13):

$$\beta(r) = \begin{cases} \sigma(r) \cdot \beta'(q) & \text{if } r = q \cdot 1, \\ \sigma(r) & \text{if } r = q \cdot i \text{ and } i \neq 1, \\ \beta'(r) & \text{otherwise.} \end{cases}$$

The principal position ξ of (t, β) is as follows:

$$\xi = \begin{cases} \xi' \cdot 1 & \text{if } \xi' = q, \\ \xi' & \text{otherwise.} \end{cases}$$

(ii) Suppose that m is a composite method. Let r''_i denote the position of $x''_i \in \mathbf{x}''$ in $Res(t\sigma/q)$ (note that $t\sigma/q$ is in the form of $m(\mathbf{c})$). Then, the following β is a CSM of (t, σ, c') (see also Fig. 14):

$$\beta(r) = \begin{cases} \beta'(q \cdot r''_i) & \text{if } r = q \cdot i, \\ \beta'(r) & \text{otherwise.} \end{cases}$$

The principal position ξ of (t, β) is as follows:

$$\xi = \begin{cases} q \cdot i \text{ if } \xi' = q \cdot r''_i, \\ \xi' \quad \text{otherwise}. \end{cases}$$

We must show that β and ξ constructed above are indeed a CSM and the principal position. The basis can be verified easily. For the induction case, consider a reduction $t\theta \rightarrow_{I_S} t'\theta' \rightarrow^*_{I_S} o' \in \nu_S(c')$.

(i) Suppose that m is a base method. By the construction, the first symbol of $\beta(q \cdot i)$ is $\sigma(q \cdot i)$. Moreover, for any $r \in V(t) - \{q \cdot i \mid 1 \leq i \leq n\}$, the first symbol of $\beta(r)$ is equal to that of $\beta'(r)$, and it is $\sigma'(r)$ by the inductive hypothesis. $t\theta/r$ must be equal to $t'\theta'/r$, and therefore, $\sigma'(r) = \sigma(r)$. Hence, the condition (1) in Def. 31 holds.

For the condition (2a), there are two cases to be considered. Suppose that $q = \xi'$. Since $\xi = \xi' \cdot 1 = q \cdot 1$, the last symbol of $\beta(\xi)$ is that of $\sigma(\xi) \cdot \beta'(\xi')$, and therefore, it is c' by the inductive hypothesis. On the other hand, suppose that $q \neq \xi'$. Since $\xi = \xi'$, the last symbol of $\beta(\xi)$ is that of $\beta'(\xi')$, and therefore, it is c' again by the inductive hypothesis. Thus the condition (2a) holds.

Let $\theta_\beta : V(t) \rightarrow O_{I_S}$ be an arbitrary mapping such that $\theta_\beta(r) = \beta(r) \cdot \alpha_r$, where $\alpha_r \in C^*$. Let $\theta_{\beta'} : V(t') \rightarrow O_{I_S}$ be a mapping such that

$$\theta_{\beta'}(r) = \begin{cases} \beta'(r) \cdot \alpha_\xi \text{ if } r = q, \\ \beta'(r) \cdot \alpha_r \text{ otherwise}. \end{cases}$$

Then, $t\theta_\beta \rightarrow_{I_S} t'\theta_{\beta'}$ by the definitions of method execution, and $t'\theta_{\beta'} \rightarrow^*_{I_S} c' \cdot \alpha_\xi$ by the inductive hypothesis. Thus the condition (2b) holds.

(ii) Suppose that m is a composite method. The first symbol of $\beta(q \cdot i)$ is equal to that of $\beta'(q \cdot r''_i)$, and it is $\sigma'(q \cdot r''_i)$ by the inductive hypothesis. By the definition of r''_i, $t\theta/q \cdot i$ is equal to $t'\theta'/q \cdot r''_i$, and therefore, $\sigma'(q \cdot r''_i) = \sigma(q \cdot i)$. Moreover, for any $r \in V(t) - \{q \cdot i \mid 1 \leq i \leq n\}$, the first symbol of $\beta(r)$ is equal to that of $\beta'(r)$, and it is $\sigma'(r)$ by the inductive hypothesis. $t\theta/r$ is equal to $t'\theta'/r$, and therefore, $\sigma'(r) = \sigma(r)$. Hence, the condition (1) in Def. 31 holds.

For the condition (2a), there are two cases to be considered. Suppose that q is a prefix of ξ'. Then, there must be j such that $q \cdot r''_j = \xi'$, and therefore, $\xi = q \cdot j$. Hence the last symbol of $\beta(\xi)$ is that of $\beta'(\xi')$, and it is c' by the inductive hypothesis. On the other hand, suppose that q is not a prefix of ξ'. Then, $\xi = \xi'$ by the construction. The last symbol of $\beta(\xi)$ is that of $\beta'(\xi')$, and it is c' by the inductive hypothesis. Thus the condition (2a) holds.

Let $\theta_\beta : V(t) \rightarrow O_{I_S}$ be an arbitrary mapping such that $\theta_\beta(r) = \beta(r) \cdot \alpha_r$, where $\alpha_r \in C^*$. Let $\theta_{\beta'} : V(t') \to O_{I_S}$ be a mapping such that

$$\theta_{\beta'}(r) = \begin{cases} \beta'(r) \cdot \alpha_{q \cdot i} \text{ if } r = q \cdot r''_i, \\ \beta'(r) \cdot \alpha_r \quad \text{otherwise.} \end{cases}$$

Then, $t\theta_\beta \to_{I_S} t'\theta_{\beta'}$ by the definitions of method execution, and $t'\theta_{\beta'} \to^*_{I_S} c' \cdot \alpha_\xi$ by the inductive hypothesis. Thus the condition (2b) holds.

5.2.3 Soundness of the Approximated Inference Attacks

The following lemma states that E_S is closed under term composition. This property is needed for showing that existence of an approximation rule in Def. 23(C) implies the existence of an attacker's inference rule in Def. 15(C), in the proof of Lemma 34.

Lemma 33 *Let t, t', and t'' be linear terms in $T_M(X)$ such that $t = t'[q \leftarrow t'']$ for some $q \in V(t')$. Let $\sigma : V(t) \to C$, $\sigma' : V(t') \to C$ and $\sigma'' : V(t'') \to C$ be mappings such that for some $c'' \in C$,*

$$\sigma'(r) = \begin{cases} c'' & \textit{if } r = q, \\ \sigma(r) & \textit{otherwise}, \end{cases}$$
$$\sigma''(r'') = \sigma(q \cdot r'').$$

Now, suppose that $c' \in E_S(t', \sigma')$ and $c'' \in E_S(t'', \sigma'')$. Then, $c' \in E_S(t, \sigma)$ (see also Fig. 15).

PROOF. Let $\beta' : V(t') \to C^+$ and $\beta'' : V(t'') \to C^+$ be arbitrary CSMs of (t', σ', c') and (t'', σ'', c''), respectively. Define $\theta : V(t) \to O_{I_S}$ as

$$\theta(r) = \begin{cases} \beta''(r'') \cdot \gamma'_q \text{ if } r = q \cdot r'', \\ \beta'(r) \qquad \text{otherwise,} \end{cases}$$

where γ'_q is the string obtained from $\beta'(q)$ by removing its first symbol. Also, let $\theta'' : V(t'') \to O_{I_S}$ be a mapping such that $\theta''(r'') = \beta''(r'') \cdot \gamma'_q$ for all $r'' \in V(t'')$. Then, $t''\theta'' \to^*_{I_S} c'' \cdot \gamma'_q = \beta'(q)$, and therefore, $t\theta \to^*_{I_S} t'\theta' \to^*_{I_S} c'$. Thus, we have $c' \in E_S(t, \sigma)$. $\square$

The next lemma states that if $t\sigma \rhd_{S,A,E_S} c'$ exists, then under I_S the corresponding inference rule *for an attacker* exists. The lemma is shown by constructing a substitution mapping which shows the soundness of the approximation using CSMs and principal positions.

Lemma 34 *Let t be a linear term in $T_M(X)$ and let σ be a mapping from $V(t)$ to C. If $t\sigma \rhd_S c' \in P_S$, then for any mapping $\theta : V(t) \to O_{I_S}$ such that $\theta(r) \in \nu_S(\sigma(r))$ for each $r \in V(t)$ and $t\theta \to^*_{I_S} o' \in \nu_S(c')$, we have $t\theta \rhd_{I_S} o' \in P_{I_S}$.*

PROOF. The lemma is shown by induction on the structure of the definition of $\rhd_S$ (see Def. 23).

Basis. Suppose that $m(\mathbf{c}) \rhd_S c'$ is obtained by Def. 23(A). Let $\theta : V(m(\mathbf{x})) \to O_{I_S}$ be a mapping which assigns an object $o_i \in \nu_S(c_i)$ to each x_i. Suppose that $m(\mathbf{x})\theta \to^*_{I_S} o' \in \nu_S(c')$. Since $m(\mathbf{c}) \in A$, we have $m(\mathbf{x})\theta \rhd_{I_S} o'$ by Def. 15(A). The case of Def. 23(B) is similarly proved.

Induction. Let t, t', and t'' be linear terms in $T_M(X)$ such that $t = t'[q \leftarrow t'']$ for some $q \in V(t')$. Let $\sigma : V(t) \to C$, $\sigma' : V(t') \to C$ and $\sigma'' : V(t'') \to C$ be mappings such that for some $c'' \in C$,

$$\sigma'(r) = \begin{cases} c'' & \text{if } r = q, \\ \sigma(r) & \text{otherwise,} \end{cases}$$
$$\sigma''(r'') = \sigma(q \cdot r'').$$

Note that this is the same setting of Lemma 33 (see also Fig. 15). Now suppose that $t'\sigma' \rhd_S c'$ is obtained by Def. 23(C) with

- $t\sigma \rhd_S c \in P_S$ for some $c \in C$;
- $t''\sigma'' \rhd_S c'' \in P_S$; and
- $c' \in E_S(t', \sigma')$.

Since $t''\sigma'' \rhd_S c'' \in P_S$, c'' must be in $E_S(t'', \sigma'')$ by Def. 23. By Lemma 33, $c' \in E_S(t, \sigma)$. Then, by Def. 23 again, $t\sigma \rhd_S c'$ must be in P_S before $t'\sigma' \rhd_S c'$ is obtained.

Let $\theta' : V(t') \to O_{I_S}$ be an arbitrary mapping such that $\theta'(r) \in \nu_S(\sigma'(r))$ for each $r \in V(t')$ and $t'\theta' \to^*_{I_S} o' \in \nu_S(c')$. We will prove that $t'\theta' \rhd_{I_S} o'$ is in P_{I_S}, using the inductive hypothesis on $t''\sigma'' \rhd_S c''$ and $t\sigma \rhd_S c'$. Let $\beta'' : V(t'') \to C^+$ be a CSM of (t'', σ'', c''). Define $\theta : V(t) \to O_{I_S}$ as

$$\theta(r) = \begin{cases} \beta''(r'') \cdot \gamma'_q & \text{if } r = q \cdot r'', \\ \theta'(r) & \text{otherwise,} \end{cases}$$

where γ'_q is the string obtained from $\theta'(q)$ by removing its first symbol. Also, let $\theta'' : V(t'') \to O_{I_S}$ be a mapping such that $\theta''(r'') = \beta''(r'') \cdot \gamma'_q$ for all $r'' \in V(t'')$. Note that $\theta(r) \in \nu_S(\sigma(r))$ for each $r \in V(t)$ and $\theta''(r'') \in \nu_S(\sigma''(r''))$ for each $r'' \in V(t'')$. Then, $t''\theta'' \to^*_{I_S} c'' \cdot \gamma'_q = \theta'(q) \in \nu_S(c'')$, and therefore, $t\theta \to^*_{I_S} t'\theta' \to^*_{I_S} o' \in \nu_S(c')$. Thus, by the inductive hypothesis, $t\theta \rhd_{I_S} o'$ and $t''\theta'' \rhd_{I_S} \theta'(q)$ are in P_{I_S}. Hence, by Def. 15(C), $t'\theta' \rhd_{I_S} o'$ is in P_{I_S}. $\square$

Lastly, as stated in the lemma below, Lemma 34 can be lifted to rewrite sequences. Then, the only if part of Theorem 30 can be derived immediately.

Lemma 35 *Let t be a linear term in $T_M(X)$ and let σ be a mapping from $V(t)$ to C. If $t\sigma \Rightarrow^*_S c'$, then there is a mapping $\theta : V(t) \to O_{I_S}$ such that $t\theta \Rightarrow^*_{I_S} c'$.*

PROOF. The lemma is shown by induction on the length of $t\sigma \Rightarrow^*_S c'$.

Basis. The lemma obviously holds if $t\sigma = c'$.

Induction. Consider a reduction $t\sigma \Rightarrow_S t'\sigma' \Rightarrow^*_S c'$, where $\sigma' : V(t') \to C$. Let q be the position at which the subterm of $t\sigma$ is rewritten in the first step of the reduction (i.e., t' can be written as $t[q \leftarrow x']$ for some variable x'). σ' must satisfy

$$\sigma'(r) = \begin{cases} c'' & \text{if } r = q, \\ \sigma(r) & \text{otherwise,} \end{cases}$$

for some $c'' \in C$. Let $t'' = t/q$ and define $\sigma'' : V(t'') \to C$ as

$$\sigma''(r'') = \sigma(q \cdot r'').$$

Then, $t''\sigma'' \rhd_S c''$ must be in P_S. By Def. 23, c'' must be in $E_S(t'', \sigma'')$, and therefore, there is a CSM $\beta'' : V(t'') \to C^+$ of (t'', σ'', c''). By applying Lemma 34 to β'', we have $t''\beta'' \rhd_{I_S} c'' \in P_{I_S}$. On the other hand, by the inductive hypothesis, there is a mapping $\theta' : V(t') \to O_{I_S}$ such that $t'\theta' \Rightarrow^*_{I_S} c'$ since $t'\sigma' \Rightarrow^*_S c'$.

Now, define $\theta : V(t) \to O_{I_S}$ as follows:

$$\theta(r) = \begin{cases} \beta''(r'') \cdot \gamma'_q & \text{if } r = q \cdot r'', \\ \theta'(r) & \text{otherwise,} \end{cases}$$

where γ'_q is the string obtained from $\theta'(q)$ by removing its first symbol. Also, let $\theta'' : V(t'') \to O_{I_S}$ be a mapping such that $\theta''(r'') = \beta''(r'') \cdot \gamma'_q$ for all $r'' \in V(t'')$. Then, $t\theta \Rightarrow_{I_S} t'\theta'$ using $t''\theta'' \rhd_{I_S} c'' \cdot \gamma'_q$ in P_{I_S}. Thus, we have $t\theta \Rightarrow^*_{I_S} c'$. $\square$

6 Incomparability of Type Inferability and Security Decidability

For general terms and schemas, type inference is impossible and the security is undecidable. On the other hand, for linear terms and schemas, type inference is possible and the security is decidable. A natural question is whether the undecidability of the security stems only from the impossibility of type inference. In this section, we provide a negative answer to this question.

Theorem 36 *The security of a non-linear term τ at $\mathbf{c}$ under a schema S is undecidable even if S is linear and $E_S(\tau, \mathbf{c})$ is computable.*

PROOF. Consider the reduction from the MPCP to the security problem stated in Section 4.1. Let $\tau = \mathsf{post}(\mathsf{isw}(x), \mathsf{isu}(x))$. Since $S_{\mathbf{w},\mathbf{u}}$ is linear, it suffices to show that $E_{S_{\mathbf{w},\mathbf{u}}}(\tau, c_1) = \{c_{\text{dummy}}\}$ for any $(\mathbf{w}, \mathbf{u})$.

Consider a pair (I, o_1) such that the string represented by (I, o_1) is empty. For any $(\mathbf{w}, \mathbf{u})$, such (I, o_1) exists and under such I, both $\mathsf{isw}(o_1)$ and $\mathsf{isu}(o_1)$ are terminating. Since post always returns an object of class c_{dummy}, we have $E_{S_{\mathbf{w},\mathbf{u}}}(\tau, c_1) = \{c_{\text{dummy}}\}$. $\square$

Note that Theorems 30 and 36 show a tight bound of the decidability of the security problem. That is, the non-linearity of only τ makes the problem undecidable.

In order for the security to be decidable, type inference of *tuples* of terms seems necessary. The essence of the reduction stated in Section 4.1 is whether for some pair (I, o_1), both $\mathsf{isw}(o_1)$ and $\mathsf{isu}(o_1)$ return objects of the same class c_{ok} under I. Thus, the results of "separated" type inference, i.e., $E_{S_{\mathbf{w},\mathbf{u}}}(\mathsf{isw}(x), c_1)$ and $E_{S_{\mathbf{w},\mathbf{u}}}(\mathsf{isu}(x), c_1)$, are insufficient. However, it is open whether type inference of tuples of terms is sufficient for the security to be decidable.

A natural next question may be whether the security problem is more difficult than type inference. We provide a negative answer again.

Theorem 37 *$E_S(\tau, \mathbf{c})$ is uncomputable even if S is linear and the security of τ at $\mathbf{c}$ is decidable.*

PROOF. Consider again $S_{\mathbf{w},\mathbf{u}}$ and A defined in Section 4.1. Modify the definition of post so that $\mathsf{post}(o, o')$ returns an object of class c_{ok} if both o and o' are objects of c_{ok}, and it returns an object of class c_{dummy} otherwise. Also, add to A the rights of post on any class. Then, the security of $\tau = \mathsf{post}(\mathsf{isw}(x), \mathsf{isu}(x))$ at c_1 is trivially decidable (i.e., τ is always insecure at c_1) since the user can invoke post on any objects. However, $E_S(\tau, c_1)$ is uncomputable since $c_{\text{ok}} \in E_S(\tau, c_1)$ if and only if $(\mathbf{w}, \mathbf{u})$ has a solution. $\Box$

The above theorem states that the security of τ may be easily decided using only A. In that case, whether τ is secure or not is no help for type inference of τ. Thus, security decidability does not imply type inferability.

7 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 query 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 linear.

It is impossible to formalize the whole "inference engine" of the attacker. We have focused on inference based on equational logic because it is one of the most fundamental and powerful kind of inference. It is practically significant that we can verify the security against such fundamental and powerful inference, although the linearity condition is necessary.

Although type inferability and decidability of the security problem are incomparable, they still seem to be closely related. Especially, as stated in Section 6, type inference of tuples of terms may be helpful for deciding the security problem. One of the future works is to examine the relationship between the type inference of tuples of terms and the decidability of the security problem.

We have assumed that a user knows the definitions of composite methods only if the methods are authorized to the user. However, in some situations, the definitions of unauthorized methods may be open to the public or can be guessed from the method names, etc. Weakening this assumption makes the definition of inference technically complicated, and therefore left as a future work.

Acknowledgements

The authors are thankful to Prof. Toru Fujiwara of Osaka University and Ms. Yumi Shimakawa for their valuable discussions on the results on linear schemas.

References

[1] Y. Ishihara, T. Morita, M. Ito, The security problem against inference attacks on object-oriented databases, in: Research Advances in Database and Information Systems Security, Kluwer, 2000, pp. 303–316.

[2] Y. Ishihara, Y. Shimakawa, T. Fujiwara, Type inferability and decidability of the security problem against inference attacks on object-oriented databases, in: Proceedings of the Sixth International Conference on Information and Communications Security, LNCS 3269, 2004, pp. 145–157.

[3] E. B. Fernandez, M. M. Larronodo-Peritrie, E. Gudes, A method-based authorization model for object-oriented databases, in: Proceedings of OOPSLA-93 Conference Workshop on Security for Object-Oriented Systems, 1993, pp. 135–150.

[4] H. Seki, Y. Ishihara, M. Ito, Authorization analysis of queries in object-oriented databases, in: Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases, LNCS 1013, 1995, pp. 521–538.

[5] E. Bertino, P. Samarati, Research issues in discretionary authorizations for object bases, in: Proceedings of OOPSLA-93 Conference Workshop on Security for Object-Oriented Systems, 1994, pp. 183–199.

[6] S. Abiteboul, P. Kanellakis, S. Ramaswamy, E. Waller, Method schemas, Journal of Computer and System Sciences 51 (3) (1995) 433–455.

[7] H. Seki, Y. Ishihara, H. Dodo, Testing type consistency of method schemas, IEICE Transactions on Information and Systems E81-D (3) (1998) 278–287.

[8] Y. Ishihara, S. Shimizu, H. Seki, M. Ito, Refinements of complexity results on type consistency for object-oriented databases, Journal of Computer and System Sciences 62 (4) (2001) 537–564.

[9] D. E. R. Denning, Cryptography and Data Security, Addison-Wesley, 1982.

[10] C. Farkas, T. Toland, C. Eastman, The inference problem and updates in relational databases, in: Databases and Application Security XV, Kluwer, 2002, pp. 181–194.

[11] K. Tajima, Static detection of security flaws in object-oriented databases, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 1996, pp. 341–352.

[12] L. Chang, I. S. Moskowitz, Bayesian methods applied to the database inference problem, in: Database Security XII, Kluwer, 1999, pp. 237–251.

[13] K. Zhang, IRI: A quantitative approach to inference analysis in relational databases, in: Database Security XI, 1998, pp. 279–290.

[14] M. Morgenstern, Security and inference in multilevel database and knowledge-base systems, in: Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, 1987, pp. 357–373.

[15] I. Moskowitz, L. Chang, An entropy-based framework for database inference, in: Proceedings of the Third International Workshop on Information Hiding, LNCS 1768, 1999, pp. 405–418.

[16] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[17] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge, 1998.

[18] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to algorithms, 1990.

Appendix: Complexity Analysis

The algorithm for deciding the sufficient condition stated in Theorem 25 consists of the following three steps:

(S1) Compute Z_0 from S using the type inference algorithm in [7], where Z_0 is the least fixpoint 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 $\tau[\mathbf{c}/\mathbf{x}] \Rightarrow^*_{S,A,Z} c$. If such c exists, then output "τ may be insecure at **c**." Otherwise, output "τ is secure at **c**."

We analyze the time complexity of the algorithm. Define the size l_t of a term t as $|Pos(t)|$, i.e., the number of positions of t. Define the description length of Σ_c, denoted $\|\Sigma_\mathrm{c}\|$, as the sum of l_t for all $(m(\mathbf{c}), t) \in \Sigma_\mathrm{c}$. Also, define the size of S, denoted $\|S\|$, as follows:

$$\|S\| = |C| + |\leq| + |M| + |\Sigma_\mathrm{b}| + \|\Sigma_\mathrm{c}\|.$$

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 positions in $Pos(t)$. Let L and H be the maximum size and height of all t in $\{t \mid (m(\mathbf{c}), t) \in \Sigma_\mathrm{c}\} \cup \{\tau\}$, respectively. Note that $k \leq N$ and $k \leq L$ by definition.

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 [7]. In this 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 [19]. 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

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

time. Thus, the complexity of (S1) is

$$\mathcal{O}(kN|C|^k(N|C|^k + k \log N)). \tag{3}$$

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. 16 shows a procedure for computing Q. Suppose that variables Q_{ans}, Q_Δ, Q'_Δ, and Z are implemented by binomial heaps. Let $\rho_{Q_{\mathrm{ans}}}$ 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,

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

where L is for a key comparison.

Before analyzing the procedure in Fig. 16 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,

$$\begin{aligned}|Q_0| &\leq \sum_{(m(\mathbf{c}),t)\in\Sigma_\mathrm{c}} |X_{t[\mathbf{c}/\mathbf{x}]}|\\ &\leq \sum_{(m(\mathbf{c}),t)\in\Sigma_\mathrm{c}} (h_{t[\mathbf{c}/\mathbf{x}]} + 1)|C|\\ &= \sum_{(m(\mathbf{c}),t)\in\Sigma_\mathrm{c}} l_{t[\mathbf{c}/\mathbf{x}]}|C|\\ &= \|\Sigma_\mathrm{c}\| \cdot |C|\\ &\leq N|C|, \end{aligned} \tag{4}$$

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 $k^h + k^{h-1} + \cdots + k^0$. In what follows, we show that

$$|X_t| \leq (|C| + 1)^{K_{h_t}}. \tag{5}$$

If $h_t = 0$, then $|X_t| = |C| \leq (|C| + 1)^{K_0} = |C| + 1$. Suppose that Eq. 5 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. 5 holds and

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

$$\leq N(|C|+1)^{k^{H+1}}, \tag{6}$$

using $K_H \leq k^{H+1}$ if $k \geq 2$. After all, from Eqs. 4 and 6, we obtain

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

Let us analyze the procedure in Fig. 16 in detail. See (T2). A straightforward algorithm can compute Res in

$$\mathcal{O}(kN|C|^k) \tag{7}$$

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

$$\begin{aligned}
&\mathcal{O}(|A|(\rho_{Q_\Delta} + \log|M_{\mathrm{c}}| + \rho_{Res} + L + \rho_{Q_\Delta}))\\
&= \mathcal{O}(N|C|^k(L\log|Q| + k^2\log N))\\
&= \mathcal{O}(k^{H+1}LN|C|^k \log N),
\end{aligned} \tag{8}$$

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 [19] 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 $Q_{\mathrm{ans}} \cup Q_\Delta$ takes $\mathcal{O}(L\log|Q|)$ time [19]. Therefore, the complexity of (T11) through (T16) except (T13) is

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

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

$$\mathcal{O}(|Q|\cdot k^2 L|C|^{k+1}\log N) = \mathcal{O}(k^2 LN(|C|+1)^{k^{H+1}+k+1}\log N). \tag{10}$$

Both of Eqs. 7 and 8 are bounded by Eq. 9. Furthermore, Eq. 10 is also bounded by Eq. 9 since $k \le N$. Thus, the complexity of (S2) is given by Eq. 9.

Lastly, consider (S3). Let $D = \{t \mid \tau[\mathbf{c}/\mathbf{x}] \Rightarrow_S^* t\}$. Then,

$$|D| \le |X_{\tau[\mathbf{c}/\mathbf{x}]}| \le |Q_0| = \mathcal{O}(N(|C|+1)^{k^{H+1}}),$$

since $h_\tau \le H$.

Fig. 17 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

$$
\begin{aligned}
&\mathcal{O}(|D|\cdot|Q|(L+\rho_Z+|C|(L+\rho_{D'_\Delta}))+|D|\cdot|Q|\cdot L\log|D|))\\
&= \mathcal{O}(|D|\cdot|Q|\cdot|C|\cdot L\log|D|)\\
&= \mathcal{O}(k^{H+1}LN^2(|C|+1)^{2k^{H+1}+1}\log N).
\end{aligned} \tag{11}
$$

(U9) can be checked in $\mathcal{O}(|D|)$ time. Therefore, the time complexity of executing (S3) is given by Eq. 11.

By Eqs. 3, 9, and 11, the time complexity of the algorithm is

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

List of Figures

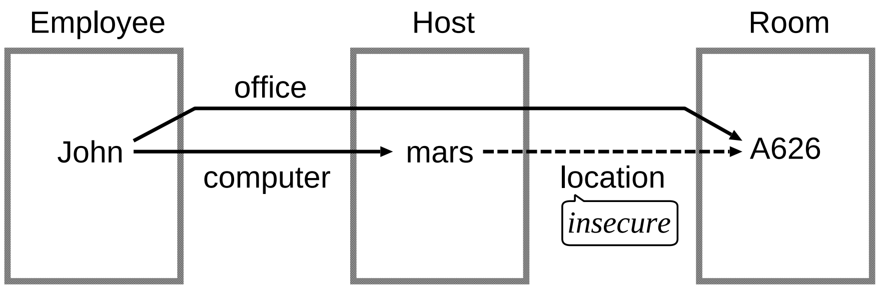

Fig. 1. An example of an insecure method.

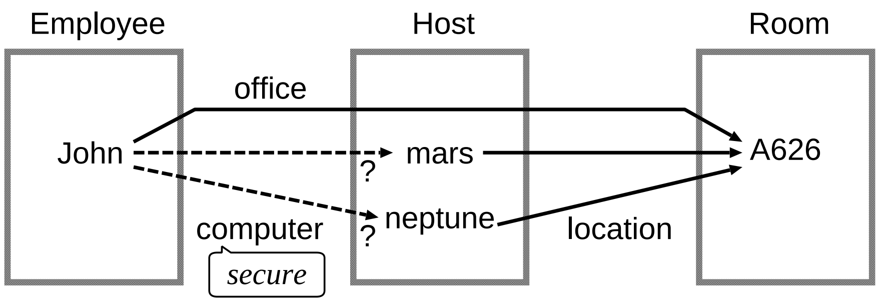

Fig. 2. An example of a secure method.

$$\begin{aligned}
C &= \{\textsf{Employee}, \textsf{Manager}, \textsf{Host}, \textsf{Server}, \textsf{Room}\} \\
&\textsf{Manager} \leq \textsf{Employee},\ \textsf{Server} \leq \textsf{Host} \\
M &= \{\textsf{boss}, \textsf{computer}, \textsf{location}, \textsf{supervisor}, \textsf{office}\} \\
\Sigma_{\mathrm{b}} &= \{(\textsf{boss}(\textsf{Employee}), \textsf{Employee}), \\
&\qquad (\textsf{boss}(\textsf{Manager}), \textsf{Manager}), \\
&\qquad (\textsf{computer}(\textsf{Employee}), \textsf{Host}), \\
&\qquad (\textsf{location}(\textsf{Host}), \textsf{Room})\} \\
\Sigma_{\mathrm{c}} &= \{(\textsf{supervisor}(\textsf{Employee}), \textsf{supervisor}(\textsf{boss}(x_1))), \\
&\qquad (\textsf{supervisor}(\textsf{Manager}), \textsf{boss}(x_1)), \\
&\qquad (\textsf{office}(\textsf{Employee}), \textsf{location}(\textsf{computer}(x_1)))\}
\end{aligned}$$

Fig. 3. A method schema S_1.

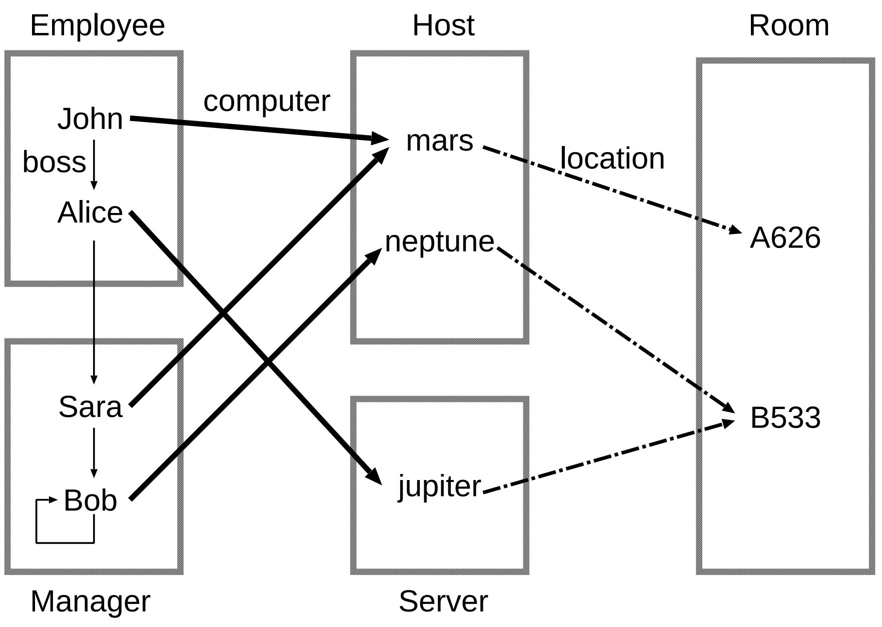

Fig. 4. A database instance I_1 of S_1.

$$\mathsf{computer}(\mathsf{John}) \rhd_{I_1} \mathsf{mars} \tag{A1}$$
$$\mathsf{computer}(\mathsf{Alice}) \rhd_{I_1} \mathsf{jupiter} \tag{A2}$$
$$\mathsf{supervisor}(\mathsf{John}) \rhd_{I_1} \mathsf{Bob} \tag{A3}$$
$$\mathsf{supervisor}(\mathsf{Alice}) \rhd_{I_1} \mathsf{Bob} \tag{A4}$$
$$\mathsf{supervisor}(\mathsf{Sara}) \rhd_{I_1} \mathsf{Bob} \tag{A5}$$
$$\mathsf{supervisor}(\mathsf{Bob}) \rhd_{I_1} \mathsf{Bob} \tag{A6}$$
$$\mathsf{office}(\mathsf{John}) \rhd_{I_1} \mathsf{A626} \tag{A7}$$
$$\mathsf{office}(\mathsf{Alice}) \rhd_{I_1} \mathsf{B533} \tag{A8}$$
$$\mathsf{office}(\mathsf{Sara}) \rhd_{I_1} \mathsf{A626} \tag{A9}$$
$$\mathsf{office}(\mathsf{Bob}) \rhd_{I_1} \mathsf{B533} \tag{A10}$$

$$\mathsf{supervisor}(\mathsf{boss}(\mathsf{John})) \rhd_{I_1} \mathsf{Bob} \tag{B1}$$
$$\mathsf{supervisor}(\mathsf{boss}(\mathsf{Alice})) \rhd_{I_1} \mathsf{Bob} \tag{B2}$$
$$\mathsf{boss}(\mathsf{Sara}) \rhd_{I_1} \mathsf{Bob} \tag{B3}$$
$$\mathsf{boss}(\mathsf{Bob}) \rhd_{I_1} \mathsf{Bob} \tag{B4}$$
$$\mathsf{location}(\mathsf{computer}(\mathsf{John})) \rhd_{I_1} \mathsf{A626} \tag{B5}$$
$$\mathsf{location}(\mathsf{computer}(\mathsf{Alice})) \rhd_{I_1} \mathsf{B533} \tag{B6}$$
$$\mathsf{location}(\mathsf{computer}(\mathsf{Sara})) \rhd_{I_1} \mathsf{A626} \tag{B7}$$
$$\mathsf{location}(\mathsf{computer}(\mathsf{Bob})) \rhd_{I_1} \mathsf{B533} \tag{B8}$$

$$\mathsf{location}(\mathsf{mars}) \rhd_{I_1} \mathsf{A626} \tag{C1}$$
$$\mathsf{location}(\mathsf{jupiter}) \rhd_{I_1} \mathsf{B533} \tag{C2}$$

Fig. 5. Contents of P_{I_1,A_1}.

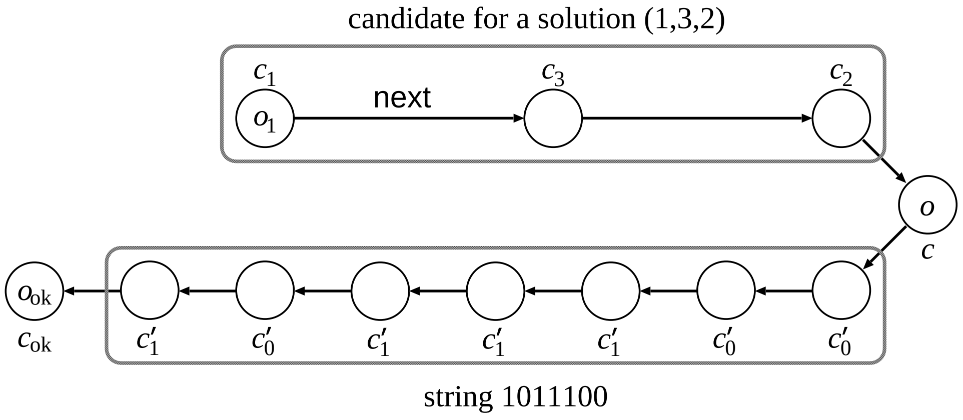
candidate for a solution (1,3,2)
c_1
o_1
next
c_3
c_2
o
c
o_{ok}
c_{ok}
c'_1 c'_0 c'_1 c'_1 c'_1 c'_0 c'_0
string 1011100

Fig. 6. An example of a database instance I_2 of $S_{\mathbf{w},\mathbf{u}}$.

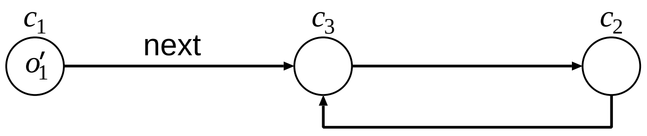

Fig. 7. Another database instance I_2' of $S_{\mathbf{w},\mathbf{u}}$.

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

Fig. 8. Z for schema S_1.

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

$$\mathsf{supervisor}(\mathsf{boss}(\mathsf{Employee})) \rhd_{S_1} \mathsf{Manager} \tag{Bi}$$
$$\mathsf{boss}(\mathsf{Manager}) \rhd_{S_1} \mathsf{Manager} \tag{Bii}$$
$$\mathsf{location}(\mathsf{computer}(\mathsf{Employee})) \rhd_{S_1} \mathsf{Room} \tag{Biii}$$
$$\mathsf{location}(\mathsf{computer}(\mathsf{Manager})) \rhd_{S_1} \mathsf{Room} \tag{Biv}$$

$$\mathsf{location}(\mathsf{Host}) \rhd_{S_1} \mathsf{Room} \tag{Ci}$$
$$\mathsf{location}(\mathsf{Server}) \rhd_{S_1} \mathsf{Room} \tag{Cii}$$

Fig. 9. Contents of $P_{S_1,A_1,Z}$.

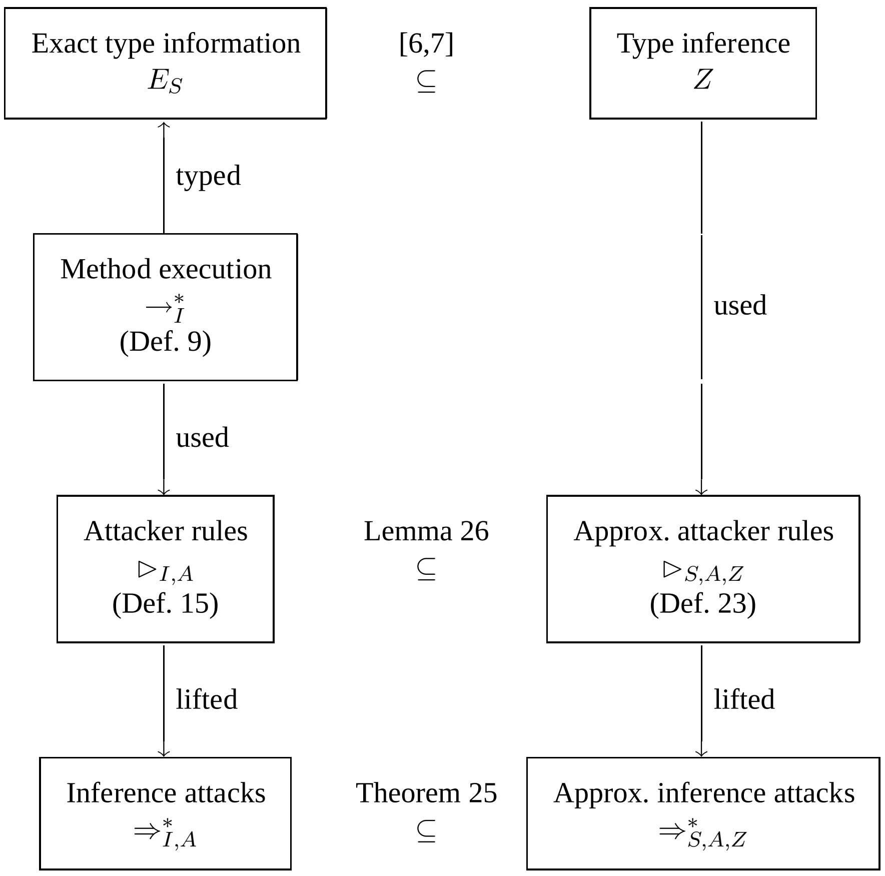
Exact type information E_S
[6,7] $\subseteq$
Type inference Z
typed
Method execution $\rightarrow_I^*$ (Def. 9)
used
used
Attacker rules $\rhd_{I,A}$ (Def. 15)
Lemma 26 $\subseteq$
Approx. attacker rules $\rhd_{S,A,Z}$ (Def. 23)
lifted
lifted
Inference attacks $\Rightarrow_{I,A}^*$
Theorem 25 $\subseteq$
Approx. inference attacks $\Rightarrow_{S,A,Z}^*$

Fig. 10. An overview of the conservativeness of the approximated inference attacks.

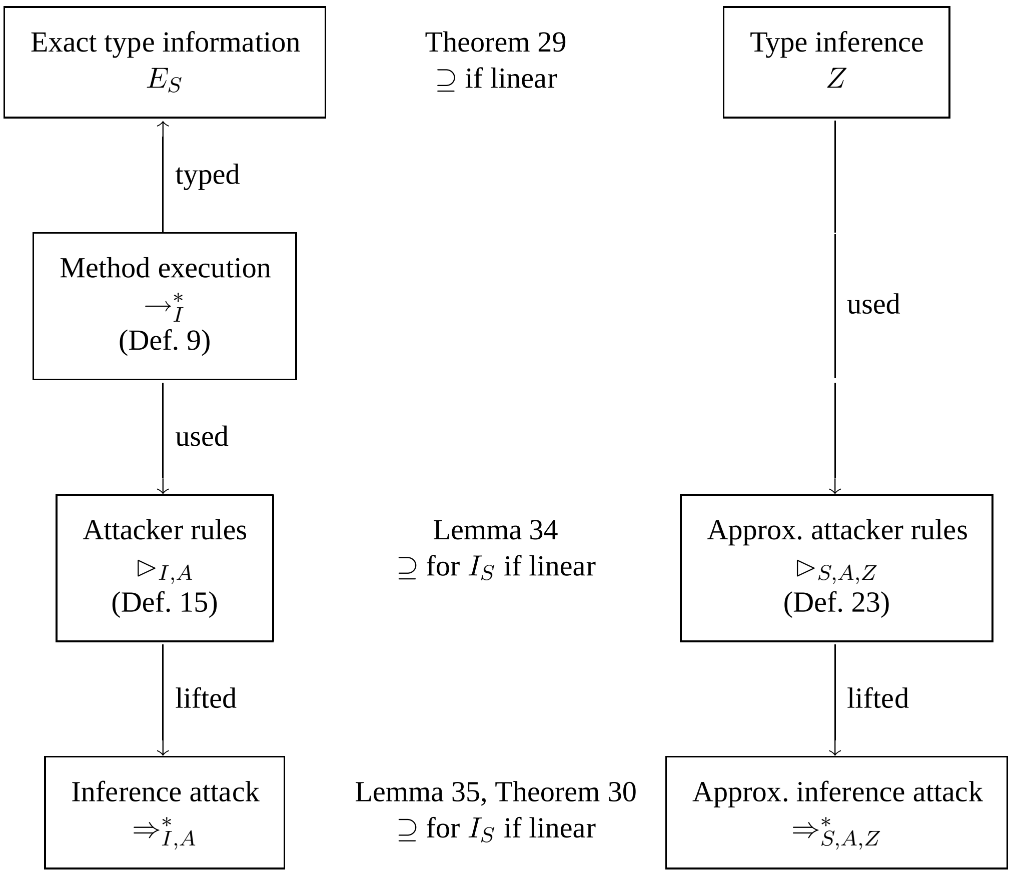
Exact type information E_S
Theorem 29
$\supseteq$ if linear
Type inference Z
typed
Method execution $\rightarrow_I^*$ (Def. 9)
used
used
Attacker rules $\rhd_{I,A}$ (Def. 15)
Lemma 34
$\supseteq$ for I_S if linear
Approx. attacker rules $\rhd_{S,A,Z}$ (Def. 23)
lifted
lifted
Inference attack $\Rightarrow_{I,A}^*$
Lemma 35, Theorem 30
$\supseteq$ for I_S if linear
Approx. inference attack $\Rightarrow_{S,A,Z}^*$

Fig. 11. An overview of the soundness of the approximated inference attacks.

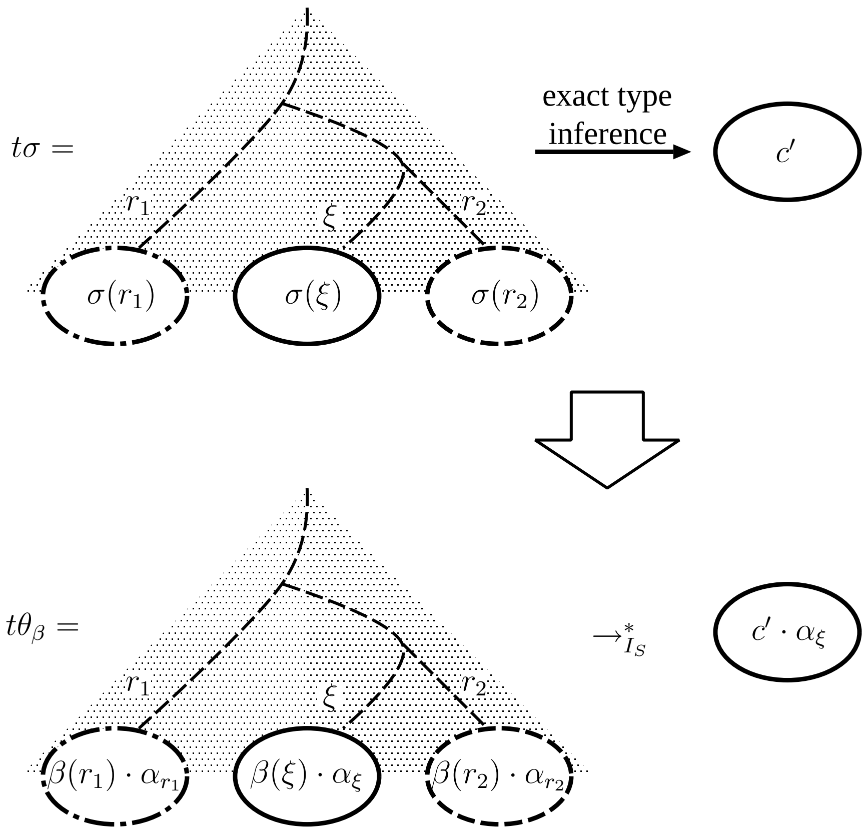

Fig. 12. A CSM β of (t, σ, c') and the principal position ξ of (t, β).

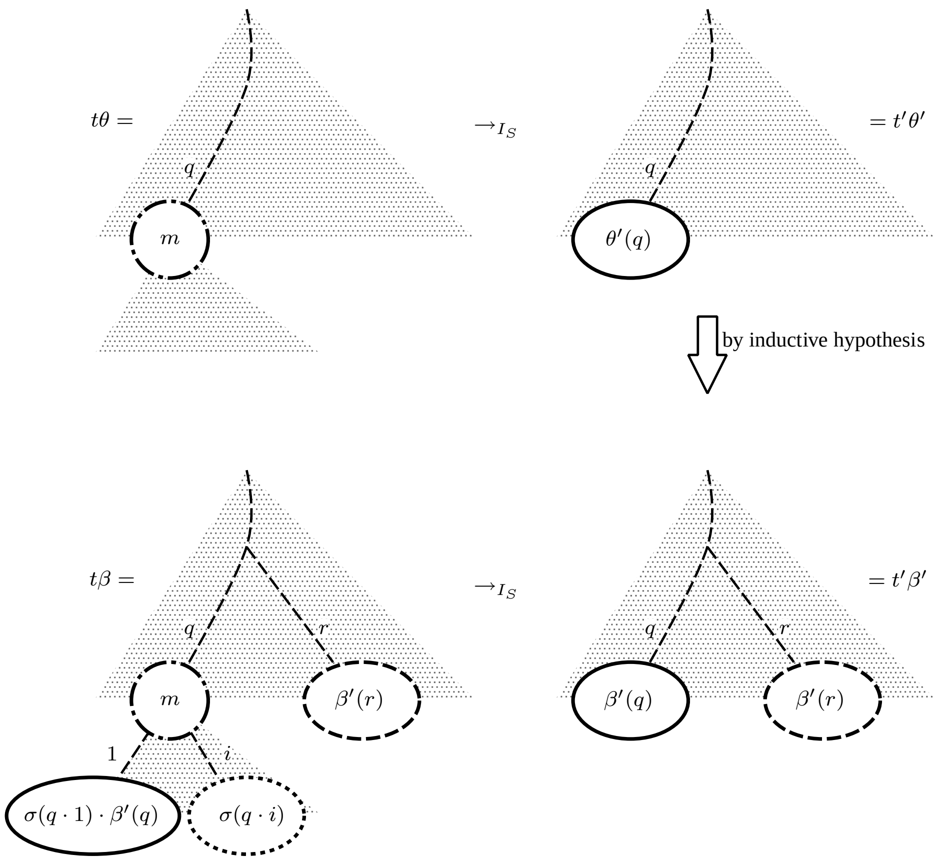

Fig. 13. Inductive construction of CSM β. Case (i).

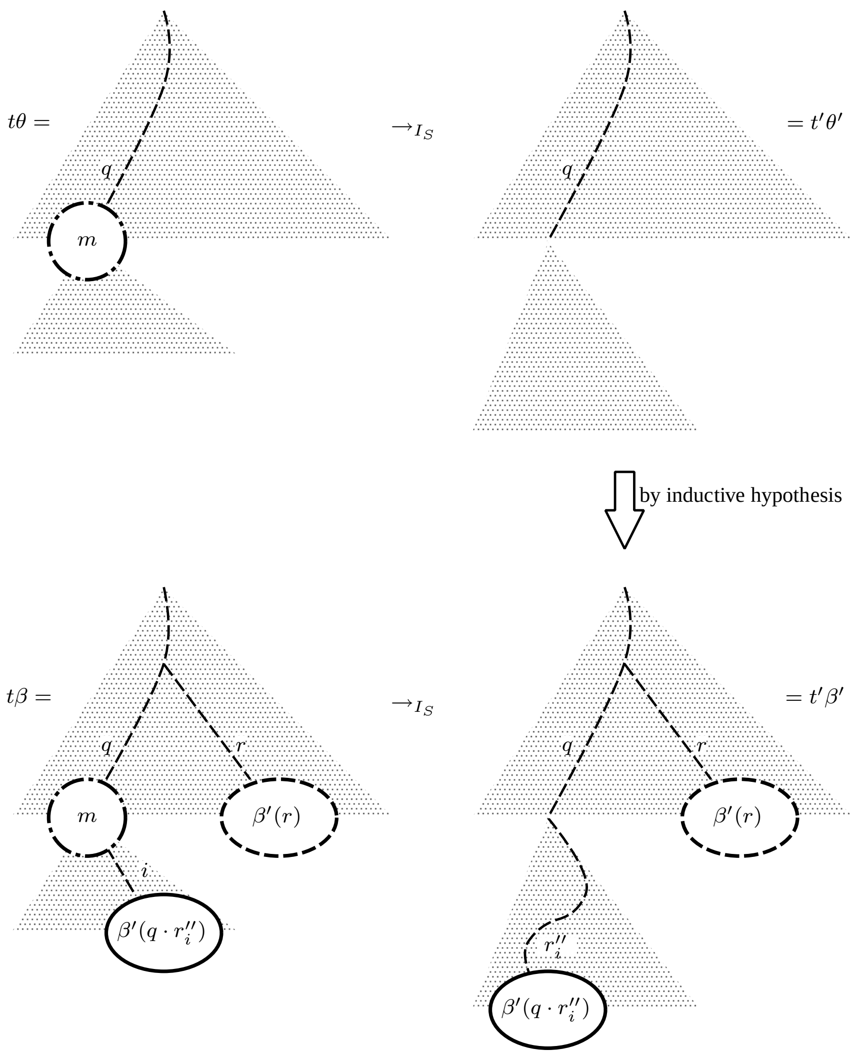

Fig. 14. Inductive construction of CSM β. Case (ii).

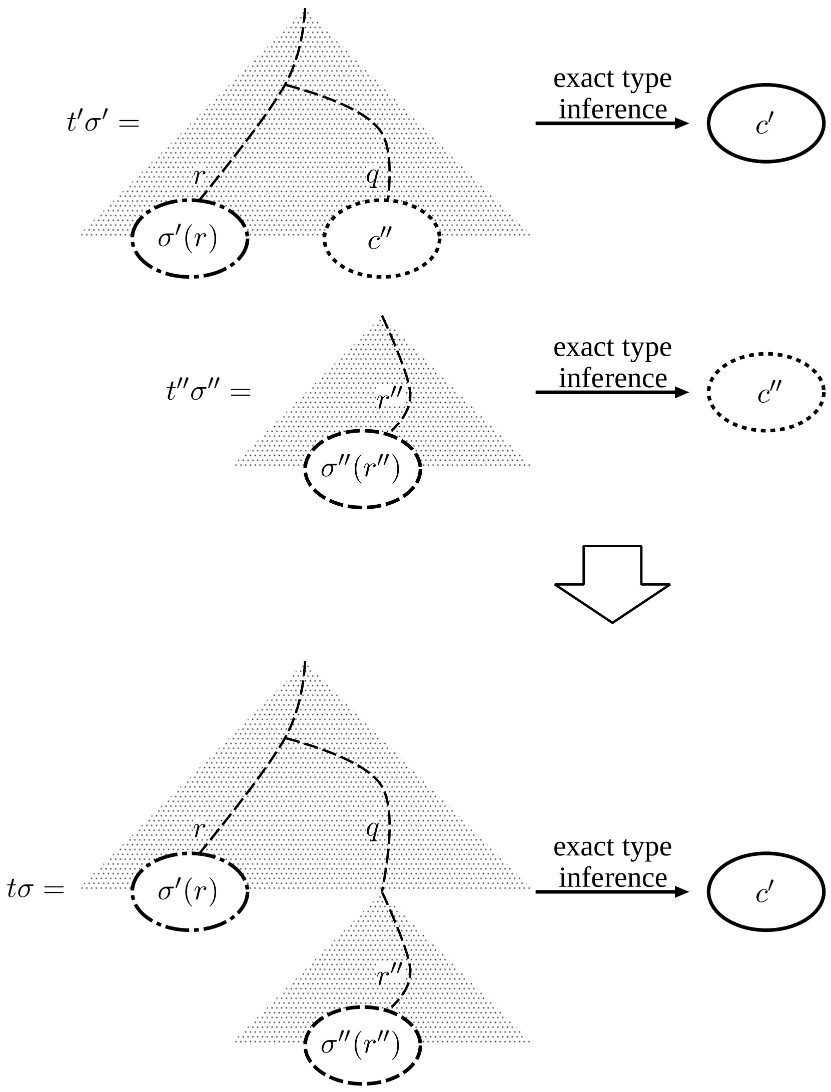

Fig. 15. An overview of Lemma 33.

```
T1    $Q_{\mathrm{ans}} \leftarrow \emptyset$, $Q_\Delta \leftarrow \emptyset$
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 $Q_\Delta$
T5        **if** $m \in M_{\mathrm{c}}$ **then**
T6            let $t$ be $Res(m(\mathbf{c}))$
T7            add $t[\mathbf{c}/\mathbf{x}]$ to $Q_\Delta$
T8    **while** $Q_\Delta \neq \emptyset$
T9        $Q'_\Delta \leftarrow \emptyset$
T10       **for each** $(t, t')$ in
                  $Q_{\mathrm{ans}} \times Q_\Delta \cup Q_\Delta \times Q_{\mathrm{ans}} \cup Q_\Delta \times Q_\Delta$
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 $Q'_\Delta$
T16       $Q_{\mathrm{ans}} \leftarrow Q_{\mathrm{ans}} \cup Q_\Delta$, $Q_\Delta \leftarrow Q'_\Delta$
T17   **output** $Q_{\mathrm{ans}}$ as $Q$
```

Fig. 16. Procedure for computing Q.

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

Fig. 17. Procedure for determining $\tau[\mathbf{c}/\mathbf{x}] \Rightarrow^*_S c$.

List of Tables

Table 1
Relationship between type inferability and security decidability.

(a) Type inferability.

queries schemas	linear	security-decidable but non-linear	general
linear	Y(Th. 29)	N(Th. 37)	N
general	N[8]	N	N[6]

(b) Decidability of the security problem.

queries schemas	linear	type-inferable but non-linear	general
linear	Y(Th. 30)	N(Th. 36)	N
general	N(Th. 21)	N	N