An equational logic based approach to the security problem against inference attacks on object-oriented databases

Yasunori Ishihara a,*, Toshiyuki Morita b,1, 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, Ibaraki, Nara 630-0192, Japan

Received 6 April 2006; received in revised form 22 December 2006
Available online 24 January 2007

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.

© 2007 Elsevier Inc. All rights reserved.

Keywords: 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 [1,2] 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 $m(c_1, \ldots, c_n) \not\in A$, $u$ can invoke $m$ indirectly through another method execution in several models, e.g., protection mode in [3]. Although such indirect invocations are useful for data hiding [3], they may also allow 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 (respectively dotted) arrow denotes an authorized (respectively 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 $\text{office}$ 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 $c_1, \ldots, c_n$ 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 $\text{computer}(\text{Employee})$ and $\text{office}(\text{Employee})$ are authorized, then $\text{location}(x)$ is insecure at Host since the user can infer $\text{location}(\text{mars}) = \text{A626}$ under the database instance shown in Fig. 1. On the other hand, it will be shown later that $\text{computer}(x)$ is secure at Employee when only $\text{location}(\text{Host})$ and $\text{office}(\text{Employee})$ 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 $u$.

![Fig. 1. An example of an insecure method.](image1)

![Fig. 2. An example of a secure method.](image2)
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 [4] 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 \( \text{computer}(\text{John}) \) and \( \text{office}(\text{John}) \) is authorized to the user \( u \), and \( u \) can obtain the execution results under \( I \). This fact is represented by the following rewriting rules:

\[
\begin{align*}
\text{computer}(\text{John}) & \triangleright_I \text{mars}, \\
\text{office}(\text{John}) & \triangleright_I \text{A626}.
\end{align*}
\]

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

\[
\text{location}(\text{computer}(\text{John}))) \triangleright_I \text{A626}.
\]

Moreover, to simulate the inference based on equational logic, some more rewriting rules (e.g., \( \text{location}(\text{mars}) \triangleright_I \text{A626} \)) 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 \( \triangleright_I \). In this example, the execution result of \( \text{location}(\text{mars}) \) can be inferred because of the rule \( \text{location}(\text{mars}) \triangleright_I \text{A626} \). 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 [4]. However, the type inference algorithm proposed in [5] 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 [6] is a kind of logical inference. Reference [7] proposes a mechanism that
accomplishes maximum data availability as long as given sensitive information is secure against logical inference. Reference [8] 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 [8] 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. Reference [9] discusses the inference based on Bayesian methods. In [10], a quantitative measure of inference risk is formally defined. In [11,12], 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 [4,13] 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 [14]. 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 $F_0, F_1, F_2, \ldots$, 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 $U \times \cdots \times U$. 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 $\mathbf{u} \in \mathbf{u}$ if $u = u_i$ for some $i$.

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

- The empty sequence $\varepsilon$ is in $\text{Pos}(t)$.
- For each $1 \leq i \leq n$, if $r \in \text{Pos}(t_i)$, then $i \cdot r \in \text{Pos}(f(t_1, \ldots, t_n))$, where the center dot “$\cdot$” represents the concatenation of sequences.
Each position in $\text{Pos}(t)$ 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:

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

Let $V(t)$ denote the set of positions of $t$ such that $t/r \in X$. That is, $V(t)$ is the positions of variables of $t$, and hence, $V(t) \subseteq \text{Pos}(t)$. Let $\sigma : V(t) \rightarrow T_F(X)$ 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 $t[t/x]$ denote $t\sigma$, where $\sigma$ 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 $M_0, M_1, M_2, \ldots$, 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 $M_{b,n}$ and $M_{c,n}$. Let $M_0 = \bigcup_{n \geq 0} M_{b,n}$ and $M_c = \bigcup_{n \geq 0} M_{c,n}$. Each $m_b \in M_b$ (respectively $m_c \in M_c$) is called a base method name (respectively composite method name). We say that $M$ is a method signature.

**Definition 3 (Method definition).** Let $c \in C^n$. A base method definition of $m_b \in M_{b,n}$ at $c$ is a pair $(m_b(c), c)$ for some $c \in C$. A composite method definition of $m_c \in M_{c,n}$ at $c$ is a pair $(m_c(c), t)$ 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 Definition 8 for the formal definition of objects). Informally, the above base method definition declares that the application of $m_b$ to $o = (o_1, \ldots, o_n)$ results in an object of $c$ or its subclass, while the above composite method definition states that the application of $m_c$ to $o$ results in term rewriting starting from $t[o/x]$.

**Definition 4 (Method schema).** (See [4,13].) A method schema $S$ is a 5-tuple $(C, \preceq, M, \Sigma_b, \Sigma_c)$, where:

1. $C$ is a finite set of class names,
2. $\preceq$ is a partial order on $C$ representing a class hierarchy,
3. $M$ is a method signature,
4. $\Sigma_b$ is a set of base method definitions, and
5. $\Sigma_c$ is a set of composite method definitions.

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

When $c' \preceq c$, we say that $c'$ is a subclass of $c$ and $c$ is a superclass of $c'$. We naturally extend $\preceq$ to $n$-tuples of classes as follows: For two tuples $c = (c_1, \ldots, c_n)$ and $c' = (c'_1, \ldots, c'_n)$, we write $c \preceq c'$ iff $c_i \preceq c'_i$ for all $i$.

**Example 5.** An example of a method schema $S_1$ is shown in Fig. 3. $\text{Manager}$ is a subclass of $\text{Employee}$, and $\text{Server}$ is a subclass of $\text{Host}$. Method $\text{boss}(e)$ returns the direct boss of employee $e$, and method $\text{supervisor}(e)$ returns the “second least manager” among the indirect bosses of $e$.

### 2.3. Inheritance

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

**Definition 6 (Resolution).** Let $S = (C, \preceq, M, \Sigma_b, \Sigma_c)$, $m_b \in M_{b,n}$, and $c \in C^n$. Suppose that $(m_b(c'), c') \in \Sigma_b$ is the base method definition of $m_b$ at the smallest $c'$ above $c$, i.e., whenever $(m_b(c''), c'') \in \Sigma_b$ and $c \preceq c''$, it is the case that
\[ C = \{ \text{Employee, Manager, Host, Server, Room} \} \]
\[ \text{Manager} \leq \text{Employee, Server} \leq \text{Host} \]
\[ M = \{ \text{boss, computer, location, supervisor, office} \} \]
\[ \Sigma_b = \{ (\text{boss(Employee)}, \text{Employee}), \]
\[ (\text{boss(Manager)}, \text{Manager}), \]
\[ (\text{computer(Employee)}, \text{Host}), \]
\[ (\text{location(Host)}, \text{Room}) \} \]
\[ \Sigma_c = \{ (\text{supervisor(Employee)}, \text{supervisor(boss(x_1)))}, \]
\[ (\text{office(Employee)}, \text{location(computer(x_1)))}) \}

Fig. 3. A method schema \( S_1 \).

\( c' \leq c'' \). The resolution \( \text{Res}(m_b(c)) \) of \( m_b \) at \( c \) is defined as \( c' \). If such a unique base method definition does not exist, then \( \text{Res}(m_b(c)) \) is undefined, denoted \( \bot \).

The resolution of a composite method is defined in the same way. Suppose that \( (m_c(c'), t') \in \Sigma_c \) is the composite method definition of \( m_c \) at the smallest \( c' \) above \( c \). Then, \( \text{Res}(m_c(c)) \) is defined as \( t' \). If such a unique composite method definition does not exist, then \( \text{Res}(m_c(c)) \) is undefined, denoted \( \bot \).

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

### 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 \( c \in C \), \( \nu \) 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 \( c = (c_1, \ldots, c_n) \), let \( \nu(c) \) denote \( \nu(c_1) \times \cdots \times \nu(c_n) \).
2. For each \( m_b \in M_{b,n}, \mu(m_b) \) is a partial mapping from \( O_I^b \) to \( O_I \) which satisfies the following two conditions.
   - (a) If \( \text{Res}(m_b(c)) = c' \), then \( \mu(m_b)|_{\nu(c)} \) is a total mapping to \( \bigcup_{c \leq c'} \nu(c) \), where \( \{\} \) denotes that the domain of \( \mu(m_b) \) is restricted to \( \nu(c') \).
   - (b) If \( \text{Res}(m_b(c)) = \bot \), then \( \mu(m_b) \) is undefined everywhere in \( \nu(c) \).
   If \( \mu(m(o)) \) is undefined, then we write \( \mu(m(o)) = \bot \).

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 \( m(o) \ (o \in \nu(c)) \) be a subterm of \( t \) at position \( r \).

1. If \( m \in M_b \) and \( \mu(m)(o) \neq \bot \), then \( t \rightarrow_I t[r \leftarrow \mu(m)(o)] \).
2. If \( m \in M_c \) and \( \text{Res}(m(c)) = t' \neq \bot \), then \( t \rightarrow_I t[r \leftarrow t'[o/x]] \).
Note that taking the innermost reduction strategy (i.e., rewriting only the term in the form of $m(o)$) 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 [14], 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 $I_1 = (v_1, \mu_1)$ of $S_1$ is shown in Fig. 4. $v_1$ is represented by gray rectangles, e.g., $v_1$(Employee) = {Alice, John}. $\mu_1$ is represented by arrows, e.g., $\mu_1$(boss)(John) = Alice, $\mu_1$(computer)(John) = mars. By Definition 9, supervisor(Alice) is executed as follows:

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

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

3. The security problem

3.1. Authorization

Various sophisticated method-based authorization models for OODBs have been proposed [1,2]. 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 $S = (C, \leq, M, \Sigma_b, \Sigma_c)$. A right is a term in the form of $m(c)$, where $m \in M_n$ and $c \in C^n$. 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 $o$ of objects. Let $c$ be the tuple of the classes such that $o \in \nu(c)$. If $m(c) \in A$, 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

![Fig. 4. A database instance $I_1$ of $S_1$.](image)
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:

\[
A_1 = \{\text{computer(Employee)}, \\
\text{supervisor(Employee)}, \text{supervisor(Manager)}, \\
\text{office(Employee)}, \text{office(Manager)}\}.
\]

Consider the instance \(I_1\) in Fig. 4. Executing \text{office(John)} by \(u\) is permitted since \(\text{John} \in v_1(\text{Employee})\) and \text{office(Employee)} \(\in A_1\). On the other hand, executing \text{computer(Sara)} is prohibited since \(\text{Sara} \in v_1(\text{Manager})\) but \text{computer(Manager)} \(\notin A_1\).

### 3.2. Inference attacks and the security problem

In this section, we formalize inference attacks (Definition 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 \text{office(John)} and obtains the result \(A626\). The information that \(u\) obtains is \text{office(John)} \(\downarrow = A626\). 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 Definition 15, how to perform equational reasoning is defined as term rewriting rules. Then, in Definition 17, we define the security problem. Let \(\tau\) 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 \(\tau\), there may exist an indirect way to compute \(\tau\) by equivalently rewriting \(\tau\) to another term involving only authorized methods. The term rewriting rules defined in Definition 15 are the rules that represent how to compute terms in a direct and/or indirect way. Thus, if \(\tau\) cannot be rewritten into an object by the rewriting rules, \(\tau\) 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 (Definition 6) as follows:

\((\ast 1)\) User \(u\) knows \(m(o) \downarrow = o\) iff \(m(c) \in A\), \(o \in v(c)\), and \(m(o) \downarrow = o \in O_f\). That is, \(u\) knows what the result of \(m(o)\) is if executing \(m(o)\) is authorized and the execution is successful.

\((\ast 2)\) User \(u\) knows \(Res(m(c)) = t\) iff \(m(c) \in A\) and \(Res(m(c)) = t\). That is, \(u\) knows the type declaration of \(m\) at \(c\) (when \(m\) is a base method) or the behavioral specification of \(m\) at \(c\) (when \(m\) is a composite method), if executing \(m(o)\) \((o \in v(c))\) is authorized.

In Example 1, \((\ast 1)\) and \((\ast 2)\) are stated informally. Also suppose that user \(u\) knows that \(o \neq o'\) for distinct objects \(o\) and \(o'\) (e.g., \(u\) knows \(\text{John} \neq \text{Alice}\), \(\text{Sara} \neq \text{A626}\), 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 \(f(t) = f(t')\)).

#### 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 \((\ast 1)\) and \((\ast 2)\).

\((Q2)\) 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 \( \text{computer}(\text{John}) \downarrow = \text{mars} \) since there may be another host \( h \) such that \( \text{computer}(\text{John}) \downarrow = h \) and \( \text{location}(h) \downarrow = \text{A626} \). However, if \( u \) knows that \( \text{location}(o) \downarrow \neq \text{A626} \) for any other object \( o \) in the database instance, then \( u \) can conclude that \( \text{computer}(\text{John}) = \text{mars} \).

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 \( A = \{ m_c(c) \mid c \in C \} \) be an authorization for a user \( u \).

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

On the other hand, if \( u \) does not know what \( C \) is, then \( u \) cannot conclude that \( m_c(t') \downarrow = t[t'/x] \downarrow \) 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 \( t'' \in v(c) \) and \( \text{Res}(m_c(c)) \neq t \). However, since type inference \([4,5]\) 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 \( m_c(x) \downarrow = t \).

Thus, when \( C \) is hidden from \( u \), each equality \( \text{Res}(m_c(c)) = t \) obtained by \((*2)\) can be translated into \( \{ m_c(o) \downarrow = t[o/x]\downarrow \mid o \in v(c) \} \), 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 \( \triangleright_{I,A} \) introduced below. From the correctness of Knuth–Bendix completion \([14]\), \( t \downarrow = o \iff t \) is reducible to \( o \) by \( \triangleright_{I,A} \).

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 \( \triangleright_{I,A} \) on \( T_M(O_I) \) satisfying the following three conditions. Intuitively, \( t \triangleright_{I,A} o \) means that the user knows or can infer that \( t \downarrow = o \).

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

\[ m(o) \triangleright_{I,A} o. \]

This corresponds to \((*1)\).

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

\[ t[o/x] \triangleright_{I,A} o. \]

This essentially corresponds to \((*2)\).

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

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

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 \( t \triangleright_{I,A} o \) in \( P_{I,A} \) implies \( t \rightarrow^* o \).

Define \( \Rightarrow_{I,A} \) as the one-step reduction relation by \( \triangleright_{I,A} \). That is, \( t \Rightarrow_{I,A} t' \iff \) there exists a subterm \( t'' \) of \( t \) at \( r'' \) such that \( t'' \triangleright_{I,A} o'' \in P_{I,A} \) and \( t' = t[r'' \leftarrow o''] \). 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 \( \triangleright_I \) and \( P_I \) instead of \( \triangleright_{I,A} \) 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 Definition 15(A), and (B1)--(B8) by Definition 15(B) with composite methods supervisor and office. Rules (C1) and (C2) are obtained by Definition 15(C). For example, (C1) is derived from (A1) and (B5).

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

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

Thus user $u$ can infer that $\tau \downarrow = B533$. On the other hand, $u$ cannot infer the value of $\tau' \downarrow$ (although $\tau' \downarrow = B533$) since no subterm of $\tau'$ 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 $c$ of classes under a schema $S$ and an authorization $A$ if there exists no instance $I = (v, \mu)$ of $S$ such that $\tau[o/x] \Rightarrow _{I,A} o$ for any $o \in v(c)$ and $o \in O_I$. Otherwise, $\tau$ is insecure at $c$ under $S$ and $A$. The security problem is to determine whether a given $\tau$ is secure at a given $c$ 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) [15] to the security problem. The reduction strategy was obtained by modifying that of the proof of the undecidability of the type-consistency problem [16].

Let $(w, u)$ be an instance of the MPCP over alphabet $\Sigma = \{0, 1\}$, where $w = (w_1, \ldots, w_n)$, $u = (u_1, \ldots, u_n)$, and $w_i, u_i \in \Sigma^*$. A solution of $(w, u)$ 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 $S_{w,u}$, a term $\tau$, and an authorization $A$ such that $(w, u)$ has a solution iff there exists a database instance $I$ of $S_{w,u}$ under which the execution result of $\tau$ can be inferred.
$Sw,u$ has classes $c, c_1, \ldots, c_n, c'_0, c'_1, c_{\text{ok}}, \text{and } c_{\text{dummy}}$, 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 $w$ and $u$. $c'_0$ and $c'_1$ represent symbols 0 and 1 in $\Sigma$, respectively. Classes $c$ and $c_{\text{ok}}$ represent the “separators” and “end markers” in database instances, respectively, as explained in the example below. Class $c_{\text{dummy}}$ represents dummy execution results.

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

$\text{(next}(c_i), c)$ for each $1 \leq i \leq n$,

$\text{(next}(c), c_{\text{ok}})$,

$\text{(next}(c'_0), c_{\text{ok}})$,

$\text{(next}(c'_1), c_{\text{ok}})$.

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

**Example 18.** Consider the following instance $(w,u)$ of the MPCP:

$$w_1 = 101, \quad u_1 = 1,$$

$$w_2 = 00, \quad u_2 = 100,$$

$$w_3 = 11, \quad u_3 = 011.$$

A typical database instance $I_2$ of $Sw,u$ is shown in Fig. 6. In the figure, method $\text{next}$ 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 $o_{\text{ok}}$ of class $c_{\text{ok}}$ and $o$ (the lower half of the figure). In the figure, the objects are of classes $c'_1, c'_0, c'_1, c'_1, c'_0, \text{and } c'_0$ (in the reverse order with respect to $\text{next}$), 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 $I'_2$.

![Fig. 6. An example of a database instance $I_2$ of $Sw,u$.](image)

![Fig. 7. Another database instance $I'_2$ of $Sw,u$.](image)
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 \(w_i = w_{i_1}w_{i_2} \cdots w_{i_d}\) for each \(i\) \((1 \leq i \leq n)\), where \(w_{i,j} \in \Sigma\). The definitions of method \(isw\) and its auxiliary composite methods \(isw_{i,j}\) are constructed as follows:

\[
\begin{align*}
(isw(c_1), & isw_{1,1}(\cdots isw_{i,d_i}(isw(next(x))))), \\
(isw(c), & next(x)), \\
(isw_{i,j}(c'_0), & next(x)) \text{ if } w_{i,j} = 0, \\
(isw_{i,j}(c'_1), & next(x)) \text{ if } w_{i,j} = 1, \\
(isw_{i,j}(c_{ok}), & dummy(x)), \\
(isw_{i,j}(c_{dummy}), & x),
\end{align*}
\]

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 \(isw(o_1)\) returns an object of class \(c_{ok}\). Otherwise, \(isw(o_1)\) 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 \(isw(o_1)\) is nonterminating under \(I\). Method \(isu\) is defined in the same way.

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

\[
\begin{align*}
(isw(c_1), & isw_{1,1}(isw_{1,2}(isw_{1,3}(isw(next(x)))))), \\
(isw(c_2), & isw_{2,1}(isw_{2,2}(isw(next(x))))), \\
(isw(c_3), & isw_{3,1}(isw_{3,2}(isw(next(x))))), \\
(isw(c), & next(x)), \\
(isw_{1,1}(c'_1), & next(x)), \quad (isw_{1,2}(c'_0), next(x)), \quad (isw_{1,3}(c'_1), next(x)), \\
(isw_{2,1}(c'_0), & next(x)), \quad (isw_{2,2}(c'_0), next(x)), \\
(isw_{3,1}(c'_1), & next(x)), \quad (isw_{3,2}(c'_1), next(x)), \\
(isw_{i,j}(c_{ok}), & dummy(x)) \quad \text{for any pair } (i, j), \\
(isw_{i,j}(c_{dummy}), & x) \quad \text{for any pair } (i, j).
\end{align*}
\]

The execution of \(isw(o_1)\) under \(I_2\) in Fig. 6 is as follows:

\[
isw(o_1) \rightarrow_{I_2}^* isw_{1,1}(isw_{1,2}(isw_{1,3}(isw_{3,1}(isw_{3,2}(isw_{2,1}(isw_{2,2}(isw(0)))))))) \\
\rightarrow_{I_2}^* o_{ok}.
\]

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

\[
A = \{ post(c_{ok}, c_{ok}), isw(c_1), isu(c_1) \}.
\]

Let \(\tau = post(isw(x), isu(x))\).

**Lemma 20.** \(\tau\) is insecure at \(c_1\) iff \((w, u)\) has a solution.

**Proof.** Only if part. \(\tau\) involves method \(post\). Since methods \(isw\) and \(isu\) never invokes \(post\), in order for the user to know the execution result of \(\tau\), the user must invoke method \(post\) directly. By the definition of \(A\), only \(post(c_{ok}, c_{ok})\) is authorized. Therefore, if \(\tau\) is insecure at \(c_1\), there must be a pair \((I, o_1)\) such that the execution results of both \(isw(o_1)\) and \(isu(o_1)\) are objects of class \(c_{ok}\). Such \((I, o_1)\) represents a solution of \((w, u)\).
If part. Suppose that $(w, u)$ has a solution. Then, there must be a pair $(I, o_1)$ such that the execution results of both $\text{isw}(o_1)$ and $\text{isu}(o_1)$ are objects of class $c_{\text{ok}}$. Therefore, the user can simply execute $\tau[o_1/x]$ and obtain the execution result. That is, $\tau$ is insecure at $c_1$. \hfill $\Box$

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 [16], 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 $e$. 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 $\triangleright_{I,A}$, i.e., if $\tau$ is insecure at $e$, then $\tau[e/x]$ is reducible to a class $c$ by the new rewriting rules. Intuitively, each $t[e/x] \in T_M(C)$ is considered as the set of instantiated terms $t[o/x]$ such that $o \in v(e)$. In order to compute the “execution result” of $t[e/x]$, the result $E_S$ of type inference would be useful, where $E_S$ is defined as follows: $c \in E_S(t, e)$ iff there is a database instance $I = (\nu, \mu)$ of $S$ such that $t(o/x) \downarrow_I \in v(e)$ for some $o \in v(e)$.

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 [4]. However, it is possible to compute an over-estimation $Z(T_M(C)) \rightarrow 2^C$ of $E_S$, that is, $Z(t[e/x]) \supseteq E_S(t, e)$ for every pair of $t$ and $e$. The algorithm in [5] 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 $(m_b(c), c')$ such that $\text{Res}(m_b(c)) = c'$, $\hat{Z}(m_b(c)) = \{c | c \subseteq c'\}$;
- For each pair $(m_c(e), t)$ such that $\text{Res}(m_c(e)) = t$, $\hat{Z}(m_c(e)) = \hat{Z}(t[e/x])$;
- For every term $m(t_1, \ldots, t_n) \in T_M(X)$ and any tuples $c_1, \ldots, c_n$ of classes,
  \[ \hat{Z}(m(t_1[c_1/x_1], \ldots, t_n[c_n/x_n])) = \bigcup_{c' \in \hat{Z}(t_1[c_1/x_1]) \times \cdots \times \hat{Z}(t_n[c_n/x_n])} \hat{Z}(m'(c')). \]

Let $Z$ be the least fixpoint of $\hat{Z}$. Also let $t$ be an arbitrary term in $T_M(X)$. It is guaranteed in [5] that $Z(t[e/x]) \supseteq E_S(t, e)$. Moreover, if $S$ contains only unary methods, then $Z$ is identical with $E_S$.

**Example 22.** Using the algorithm in [5], we can compute $Z$ for schema $S_1$ in Fig. 3. The result is presented in Fig. 8. For example, $Z(\text{boss}(\text{Employee})) = \{\text{Employee}, \text{Manager}\}$ means that for any object $e$ of $\text{Employee}$, the result of $\text{boss}(e)$ is an object of either $\text{Employee}$ or $\text{Manager}$. 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 $Z(t[e/x])$ is, the better approximation we have, although the approximation is still conservative even when $Z(t[e/x]) = C$ for every pair of $t$ and $e$. In the next definition, the class-level inference, which approximates the object-level inference $\triangleright_{I,A}$, is defined as inference rules $\triangleright_{S,A,Z}$ on $T_M(C)$. The definition is similar to the definition...
\[
Z(\text{boss(Employee)}) = \{\text{Employee, Manager}\} \\
Z(\text{boss(Manager)}) = \{\text{Manager}\} \\
Z(\text{computer(Employee)}) = \{\text{Host, Server}\} \\
Z(\text{computer(Manager)}) = \{\text{Host, Server}\} \\
Z(\text{location(Host)}) = \{\text{Room}\} \\
Z(\text{location(Server)}) = \{\text{Room}\} \\
Z(\text{supervisor(Employee)}) = \{\text{Manager}\} \\
Z(\text{supervisor(Manager)}) = \{\text{Manager}\} \\
Z(\text{office(Employee)}) = \{\text{Room}\} \\
Z(\text{office(Manager)}) = \{\text{Room}\} \\
Z(m(c)) = \emptyset \text{ for any other pairs of } m \text{ and } c \\
Z(m(t)) = \bigcup_{c \in Z(m)} Z(m(c))
\]

Fig. 8. \(Z\) for schema \(S_1\).

of \(\supset_{I,A}\) (Definition 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 \(\supset_{S,A,Z}\) on \(T_M(C)\) satisfying the following three conditions:

(A) If \(m(c) \in A\), then \(P_{S,A,Z}\) contains
\[
m(c) \supset_{S,A,Z} c
\]
for each \(c \in Z(m(c))\).

(B) If \(m_c(c) \in A\), \(m_c \in M_c\), and \(\text{Res}(m_c(c)) = t \neq \bot\), then \(P_{S,A,Z}\) contains
\[
t[c/x] \supset_{S,A,Z} c
\]
for each \(c \in Z(t[c/x])\).

(C) If \(P_S\) contains \(t \supset_{S,A,Z} c\) and \(t'' \supset_{S,A,Z} c''\) such that \(t''\) is a proper subterm of \(t\) at \(r''\), then \(P_{S,A,Z}\) contains
\[
t[r'' \leftarrow c''] \supset_{S,A,Z} c'
\]
for each \(c' \in Z(t[r'' \leftarrow c''])\).

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

**Example 24.** Figure 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 Definition 23(A), and (Bi)–(Biv) by Definition 23(B) with composite methods supervisor and office. Rules (Ci) and (Cii) are obtained by Definition 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
\[
Z'(\text{ supervisor(Employee)}) = \{\text{Employee, Manager}\}.
\]
Then, $P_{S_1,A_1,Z'}$ contains the following two rules as well as all the rules in $P_{S_1,A_1,Z}$:

$$\text{supervisor}(\text{Employee}) \triangleright_{S_1} \text{Employee}, \quad (\text{Aiii}')$$
$$\text{supervisor}(\text{boss}(\text{Employee})) \triangleright_{S_1} \text{Employee}. \quad (\text{Bi}')$$

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 sense, $P_{S_1,A_1,Z'}$ 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 $\tau[c/x] \Rightarrow_{S,A,Z}^* c$, then $\tau$ is secure at $c$, i.e., there exists no instance $I = (v, \mu)$ such that $\tau[o/x] \Rightarrow_{I,A}^* o$ for any $o \in v(c)$ 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 $I = (v, \mu)$ such that $t[o/x] \triangleright_I o \in P_I$ for some $o \in v(c)$ and $o \in v(c)$, then $t[c/x] \triangleright_S c \in P_S$.

**Proof.** We use induction on the structure of the definition of $\triangleright_I$ (see Definition 15).

**Basis.** Consider the case that $m(o) \triangleright_I o$ ($o \in v(c)$) is obtained from Definition 15(A). Then, $m(o) \in A$, $o \in v(c)$, and $m(o) \downarrow = o$. Moreover, $c \in Z(m(o))$ from the property of $Z$. From Definition 23(A), $P_S$ contains $m(o) \triangleright_S c$ since $m(c) \in A$ and $c \in Z(m(c))$. The case that $\text{Res}(m(o))[o/x] \triangleright_I o$ is obtained from Definition 15(B) can be treated in the same way.

**Induction.** Suppose that $t''[o''/x''']$ ($o'' \in v(c')$) is a proper subterm of $t[o/x]$ ($o \in v(c)$) at $r''$ and that $t[o/x] \triangleright_I o$ ($o \in v(c)$) and $t''[o''/x'''] \triangleright_I o''$ ($o'' \in v(c'')$) have been obtained. Let $t'[o'/x'] = t[o/x][r'' \leftarrow o'']$ ($o' \in v(c')$), and suppose that $t'[o'/x'] \triangleright_I o$ is obtained from Definition 15(C). By the inductive hypothesis, $P_S$ contains both $t[c/x] \triangleright_S c$ and $t''[c''/x'''] \triangleright_S c''$. From the definition of $t'[o'/x']$, we obtain $t'[c'/x'] = t[c/x][r'' \leftarrow c'']$. Since $t'[o'/x'] \triangleright_I o \in P_I$ implies $t'[o'/x'] \downarrow = o$, it holds that $c \in Z(t'[c'/x'])$. From the above inductive hypothesis and Definition 23(C), we can conclude that $t'[c'/x'] \triangleright_S c \in P_S$. □
Fig. 10. An overview of the conservativeness of the approximated inference attacks.

By Lemma 26, it can be easily shown that if there is \( I = (\nu, \mu) \) such that \( t[o/x] \Rightarrow^* t'[o'/x'] \) for some \( o \in \nu(c) \) and \( o' \in \nu(c') \), then \( t[c/x] \Rightarrow^* t'[c'/x'] \). Theorem 25 is implied by this fact.

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

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

The proposed sufficient condition is obviously decidable, since the right-hand side of each rule \( \triangleright_{S,A,Z} \) 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 \( |\text{Pos}(t)| \), i.e., the number of positions of \( t \). Define the description length of \( \Sigma_c \), denoted \( \|\Sigma_c\| \), as the sum of the size of all \( t \) such that \( (m(c), t) \in \Sigma_c \). Also, define the size of \( S \), denoted \( \|S\| \), as follows:

\[
\|S\| = |C| + |\leq| + |M| + |\Sigma_b| + \|\Sigma_c\|.
\]

Let \( k \) be the maximum arity of all the methods. The height of \( t \) is defined as the maximum length of the positions in \( \text{Pos}(t) \). Let \( L \) and \( H \) be the maximum size and height of all \( t \) in \( \{t \mid (m(c), t) \in \Sigma_c \} \cup \{\tau\} \), respectively. The total time complexity (including computation of \( Z \)) is

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

See Appendix A for details.
5. Linear case

5.1. Type inferability of linear schemas

A schema $S$ is linear if for every composite method definition $(m_c(c), t)$ 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 [5].

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

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

1. For each $c \in C$, define
   $$v_S(c) = \{c \cdot \alpha \mid \alpha \in C^* \text{ and the length of } c \cdot \alpha \text{ is at most } N\}.$$
   Here, $C^*$ denotes the Kleene closure of $C$.

2. For each $m_b \in M_b$, define $\mu_S(m_b)$ as follows:
   a. Suppose that $\text{Res}(m_b(c_1, c_2, \ldots, c_n)) = c'$. Then, for any $o_i \in v_S(c_i)$ ($2 \leq i \leq n$), define $\mu_S(m_b)(c_1, o_2, \ldots, o_n) = c'$. Moreover, for $l \geq 1$,
      $$\mu_S(m_b)(c_1 \cdot c'_1 \cdot c'_2 \cdot \ldots \cdot c'_l, o_2, \ldots, o_n) = \begin{cases} c'_1 \cdot c'_2 \cdot \ldots \cdot c'_l, & \text{if } c'_1 \leq c', \\ c'_2 \cdot \ldots \cdot c'_l, & \text{otherwise.} \end{cases}$$
   b. Suppose that $\text{Res}(m_b(c_1, c_2, \ldots, c_n)) = \bot$. Then, for any $o_i \in v_S(c_i)$ ($1 \leq i \leq n$), $\mu_S(m_b)(o_1, \ldots, o_n)$ 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 $(o_1, \ldots, o_n) \in v_S(c_1, \ldots, c_n)$, $\mu_S(m_b)(o_1, \ldots, o_n)$ 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 [5]. See the four kinds of equations in Section 4.2.1. Let $Z$ be the least fixpoint of $\hat{Z}$. We already have $Z(t[c/x]) \supseteq E_S(t, c)$ by [5] 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 $x$, there always exist objects $o$ of classes $c$ such that $c \in Z(t[o/x])$ implies $t[o/x] \to^*_{I_S} c \cdot \alpha$, which means that $t[o/x] \downarrow_{I_S}$ 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 $c \cdot \alpha \to^*_{I_S} c \cdot \alpha$ for any $\alpha \in C^*$.
- Consider $\hat{Z}(m_b(c))$. Any $c \in \hat{Z}(m_b(c))$ must be a subclass of $\text{Res}(m_b(c))$. 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, $m_b(o) = m_b(o_1, \ldots, o_n) \to^*_{I_S} c \cdot \alpha$ by the definition of $\mu_S$.
- Suppose that $c \in Z(m_c(c))$. Then, there exists $t \in T_M(X)$ such that $\text{Res}(m_c(c)) = t$ and $c \in Z(t[c/x])$. By the inductive hypothesis, for any $\alpha \in C^*$, there exists $o \in v_S(c)$ such that $t[o/x] \to^*_{I_S} c \cdot \alpha$. Therefore, $m_c(o) \to^*_{I_S} t[o/x] \to^*_{I_S} c \cdot \alpha$.
- Suppose that $c \in Z(m(t_1[c_1/x_1], \ldots, t_n[c_n/x_n]))$. Then, there exists $c' = (c'_1, \ldots, c'_n) \in Z(t_1[c_1/x_1]) \times \cdots \times Z(t_n[c_n/x_n])$ such that $c = Z(m(c'))$. By the inductive hypothesis, for any $\alpha \in C^*$, there exists $o' = (o'_1, \ldots, o'_n) \in v_S(c')$ such that $m(o') \to^*_{I_S} c \cdot \alpha$. Note that by the definition of $v_S$, $o'_i = c'_i \cdot \alpha'_i$ 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 $\alpha'_i$, there exists $o_i \in v_S(c_i)$
such that $t_i[o_i/x_i] \rightarrow_{I_S}^* c'_i \cdot \alpha'_i$. Thus, for any $\alpha$, there exist $o_1, \ldots, o_n$ such that $m(t_1[o_1/x_1], \ldots, t_n[o_n/x_n]) \rightarrow_{I_S}^* c \cdot \alpha$.

The theorem has been proved since $Z(t[e/x])$ is computable and $E_S(t, c) = Z(t[e/x])$ if both $S$ and $\tau$ are linear. \hfill $\square$

5.2. Decidability of the security of linear terms under linear schemas

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 $\tau$ are linear and $E_S$ is used as $Z$.

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

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 $E_S(t, c)$, 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 $\Rightarrow^* S, A, E_S$ implies $\Rightarrow^* I_S, A$ (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 Definition 23(C) guarantees the existence of an inference rule in Definition 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

![Fig. 11. An overview of the soundness of the approximated inference attacks.](image-url)
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 \( \sigma : V(t) \rightarrow C \) and \( c' \in E_S(t, \sigma) \). A consumed string mapping (CSM for short) of \((t, \sigma, c')\) is a mapping \( \beta : V(t) \rightarrow C^+ \) 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 \((t, \beta)\), such that
   a. the last (rightmost) symbol of \( \beta(\xi) \) is \( c' \), and
   b. for any mapping \( \theta_\beta : V(t) \rightarrow O_{I_S} \) such that \( \theta_\beta(r) = \beta(r) \cdot \alpha_r \) where \( \alpha_r \in C^* \), we have \( t\theta_\beta \rightarrow^*_{I_S} c' \cdot \alpha_\xi \).

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

**Example 32.** Consider a schema \( S = (C, \leq, M, \Sigma_b, \Sigma_c) \) with

\[
M_b = \{ (m_b(c_1, c_2), c_3), (m'_b(c_3), c_4) \},
\]

\[
M_c = \{ (m_c(c_1, c_2), m_b(x_2, x_1)) \}.
\]

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

\[
m'_b(m_b(c_1 \cdot c_3 \cdot c_4 \cdot \alpha_{1,1}, c_2 \cdot \alpha_{1,2})) \rightarrow_{I_S} m'_b(c_3 \cdot c_4 \cdot \alpha_{1,1}) \rightarrow_{I_S} c_4 \cdot \alpha_{1,1}.
\]
Thus, $\beta$ represents the prefixes of the object names that are “consumed” during the execution of $t$ under $I_S$.

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

$$m_c(c_2 \cdot \alpha_1, c_1 \cdot c_3 \cdot \alpha_2) \rightarrow_{I_S} m_b(c_1 \cdot c_3 \cdot \alpha_2, c_2 \cdot \alpha_1) \rightarrow_{I_S} c_3 \cdot \alpha_2.$$ 

Note that $c' \in E_S(t, \sigma) = E_S(t, e)$, one of the preconditions in Definition 31, is equivalent to $c' \in Z(t[e/x])$ by Theorem 29. Thus, $c' \in E_S(t, \sigma)$ means that fixpoint computation of $t[e/x]$ derives $c'$. By Theorem 29, this computation can be simulated by a syntactic instance $I_S$. For a CSM mapping $\beta$ 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 $\beta$ of $(t, \sigma, c')$ and the principal position of $(t, \beta)$. 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 $\theta : V(t) \rightarrow O_{I_S}$ such that $\theta(t) \rightarrow^* \theta(t) o' \in v_S(c')$ and $\theta(r) \in v_S(\sigma(r))$ for each $r \in V(t)$. The following proof is based on the induction on the execution length of $t \theta \rightarrow^* o'$.

**Basis.** Consider the zero-length reduction $x\theta = \sigma \in v_S(c')$. Then, a mapping $\beta$ such that $\beta(\epsilon) = c'$ is a CSM of $(x, \sigma, c')$. Also, $\epsilon$ is the principal position of $(x, \beta)$.

**Induction.** Consider a reduction $t \theta \rightarrow_{I_S} t' \theta' \rightarrow^* _{I_S} o' \in v_S(c')$, where $\theta' : V(t') \rightarrow O_{I_S}$. Let $\sigma' : V(t') \rightarrow C$ be the mapping such that $\theta'(r') \in v_S(\sigma'(r'))$ 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 $m(x') = t/q$. Assume inductively that $\beta' : V(t') \rightarrow C'$ is a CSM of $(t', \sigma', c')$ and $\xi' \in V(t')$ is the principal position of $(t', \beta')$.

(i) Suppose that $m$ is a base method. Then, the following $\beta$ is a CSM of $(t, \sigma, c')$ (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 $\xi$ of $(t, \beta)$ 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 $x''_i \in X''$ in $Res(t \sigma/q)$ (note that $t \sigma/q$ is in the form of $m(c)$). Then, the following $\beta$ is a CSM of $(t, \sigma, c')$ (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 $\xi$ of $(t, \beta)$ is as follows:

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

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

(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 $r \in V(t) - \{q \cdot i | 1 \leq i \leq n\}$, 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 Definition 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 \( r \in V(t) - \{q \cdot i | 1 \leq i \leq n\} \), 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 Definition 31 holds.

For the condition (2a), there are two cases to be considered. Suppose that \( q \) is a prefix of \( \xi' \). 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 \( \xi' \). 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 $\theta : V(t) \rightarrow O_{I_S}$ be an arbitrary mapping such that $\theta(r) = \beta(r) \cdot \alpha_r$, where $\alpha_r \in C^*$. Let $\theta' : V(t') \rightarrow O_{I_S}$ be a mapping such that

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

Then, $t\theta : I_S \rightarrow t'\theta'$ by the definitions of method execution, and $t'\theta' : I_S \rightarrow c' \cdot \alpha_\xi$ 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 Definition 23(C) implies the existence of an attacker’s inference rule in Definition 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 $\sigma : V(t) \rightarrow C$, $\sigma' : V(t') \rightarrow C$ and $\sigma'' : V(t'') \rightarrow C$ be mappings such that for some $c'' \in C$, 

...
Fig. 15. An overview of Lemma 33.

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

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 \((t', \sigma', c')\) and \((t'', \sigma'', c'')\), respectively. Define \(\theta : V(t) \to O_{I_S}\) as
\[
\theta(r) = \begin{cases} 
  \beta''(r'') \cdot \gamma_q' & \text{if } r = q \cdot r'', \\
  \beta'(r) & \text{otherwise},
\end{cases}
\]
where \(\gamma_q'\) 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 \(c' \in E_S(t, \sigma)\). □

The next lemma states that if \(t \sigma \triangleright_{S,A,E_S} c'\) 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 \(\sigma\) be a mapping from \(V(t)\) to \(C\). If \(t \sigma \triangleright_S c' \in P_S\), then for any mapping \(\theta : V(t) \to O_{I_S}\) such that \(\theta(r) \in v_S(\sigma(r))\) for each \(r \in V(t)\) and \(t \theta \to^*_{I_S} o' \in v_S(c')\), we have \(t \theta \triangleright_{I_S} o' \in P_{I_S}\).

**Proof.** The lemma is shown by induction on the structure of the definition of \(\triangleright_{S}\) (see Definition 23).
Since $t \sigma \triangleright_S c' \in P_S$, $c'$ must be in $E_S(t''',\sigma'')$ by Definition 23. By Lemma 33, $c' \in E_S(t',\sigma)$. Then, by Definition 23 again, $t \sigma \triangleright_S c'$ must be in $P_S$ before $t' \sigma' \triangleright_S c'$ is obtained.

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

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

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

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 $\sigma$ be a mapping from $V(t)$ to $C$. If $t \sigma \Rightarrow_S c'$, then there is a mapping $\theta : V(t) \rightarrow O_{I_S}$ 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 $\sigma' : V(t') \rightarrow C$. 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 $x'$). $\sigma'$ 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 $\sigma'' : V(t'') \rightarrow C$ as

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

Then, $t'' \sigma'' \Rightarrow_S c''$ must be in $P_S$. By Definition 23, $c''$ must be in $E_S(t'',\sigma'')$, and therefore, there is a CSM $\beta'' : V(t'') \rightarrow C^+$ of $(t'',\sigma'', c'')$. By applying Lemma 34 to $\beta''$, we have $t' \beta'' \rightarrow_{I_S} c'' \in P_{I_S}$. On the other hand, by the inductive hypothesis, there is a mapping $\theta' : V(t') \rightarrow O_{I_S}$ such that $t' \theta' \Rightarrow_{I_S} c'$ since $t' \sigma' \Rightarrow_S c'$.
Now, define \( \theta : V(r) \to O_{I_S} \) as follows:
\[
\theta(r) = \begin{cases} 
\beta''(r'') \cdot \gamma'_q & \text{if } r = q \cdot r'', \\
\theta'(r) & \text{otherwise}, 
\end{cases}
\]
where \( \gamma'_q \) is the string obtained from \( \theta'(q) \) by removing its first symbol. Also, let \( \theta'' : V(r'') \to O_{I_S} \) be a mapping such that \( \theta''(r'') = \beta''(r'') \cdot \gamma'_q \) for all \( r'' \in V(r'') \). Then, \( t \theta \Rightarrow_{I_S} t' \theta' \) using \( t'' \theta'' \Rightarrow_{I_S} c'' \cdot \gamma'_q \) 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 \( \tau \) at \( c \) under a schema \( S \) is undecidable even if \( S \) is linear and \( E_S(\tau, c) \) is computable.

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

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

Note that Theorems 30 and 36 show a tight bound of the decidability of the security problem. That is, the non-linearity of only \( \tau \) 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 \( \text{isw}(o_1) \) and \( \text{isu}(o_1) \) return objects of the same class \( c_{\text{ok}} \) under \( I \). Thus, the results of “separated” type inference, i.e., \( E_{S_{w,u}}(\text{isw}(x), c_1) \) and \( E_{S_{w,u}}(\text{isu}(x), c_1) \), 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. \( E_S(\tau, c) \) is uncomputable even if \( S \) is linear and the security of \( \tau \) at \( c \) is decidable.

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

The above theorem states that the security of \( \tau \) may be easily decided using only \( A \). In that case, whether \( \tau \) is secure or not is no help for type inference of \( \tau \). 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 (\( \triangleright S \)) which conservatively approximates object-level inference (\( \triangleright_I \)). 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.

Acknowledgments

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.

Appendix A. 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 [5], where $Z_0$ is the least fixpoint $Z$ whose domain is restricted to $\{m(e) \mid m \in M_n, e \in C^n\}$.

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

(S3) Determine whether there exists a class $c$ such that $\tau[e/x] \Rightarrow^*_{S,A,Z} c$. If such $c$ exists, then output “$\tau$ may be insecure at $e$.” Otherwise, output “$\tau$ is secure at $e$.”

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 $\Sigma_c$, denoted $\|\Sigma_c\|$, as the sum of $l_t$ for all $(m(e), t) \in \Sigma_c$. Also, define the size of $S$, denoted $\|S\|$, as follows:

$$\|S\| = |C| + \|\Sigma_c\|.$$  

For readability, we use $N$ to mean $\|S\|$. Let $k$ be the maximum arity of all the methods. The height of $t$, denoted $h_t$, is defined as the maximum length of the positions in $Pos(t)$. Let $L$ and $H$ be the maximum size and height of all $t$ in $\{t \mid (m(e), t) \in \Sigma_c\} \cup \{\tau\}$, respectively. Note that $k \leq N$ and $k \leq L$ by definition.

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

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

which is given in [5]. In this algorithm, $Z_0$ is implemented by a table, and retrieving an element from $Z_0$ takes $O(kN|C|^k)$ time. In (S1), we also reconstruct $Z_0$ by a more efficient data structure such as binomial heap [17]. The time complexity $\rho_{Z_0}$ of retrieving an element from or inserting an element into $Z_0$ becomes

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

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

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

time. Thus, the complexity of (S1) is

$$O(kN|C|^k(N|C|^k + k \log N)).$$ (A.1)
Next, consider (S2). Define \( Q = \{ t \mid t \triangleright_S c \in P_\Sigma \} \). In order to compute \( P_\Sigma \), it suffices to compute \( Q \), since the right-hand side of \( \triangleright_S \) can be computed from the left-hand side and \( Z \).

Figure 16 shows a procedure for computing \( Q \). Suppose that variables \( Q_{\text{ans}}, Q_\Delta, Q'_\Delta, \) and \( Z \) are implemented by binomial heaps. Let \( \rho_{Q_{\text{ans}}} \) denote the complexity of retrieving an element from or inserting an element into \( Q_{\text{ans}} \). Define \( \rho_{Q_\Delta}, \rho_{Q'_\Delta}, \) and \( \rho_Z \) in the same way. Then,

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

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 \( \triangleright_S \). Formally,

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

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

\[
X_c = C, \quad X_{m(t)} = C \cup \{ m(t') \mid t' \in X_{t'} \}.
\]

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:

\[
|X_c| = |C|, \quad |X_{m(t)}| \leq |C| + \prod_i |X_{t_i}|.
\]

If \( k = 1 \), then

\[
|X_t| \leq (h_t + 1)|C|,
\]

and therefore,

\[
|Q_0| \leq \sum_{(m(c), t) \in \Sigma_c} |X_{t[c/x]}| \leq \sum_{(m(c), t) \in \Sigma_c} (h_t[c/x] + 1)|C| = \sum_{(m(c), t) \in \Sigma_c} l_t[c/x]|C|
\]

\[
= \|\Sigma_c\| \cdot |C| \leq N|C|,
\]
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
\[
|X_t| \leq (|C| + 1)^{K_{h_t}}. \tag{A.3}
\]
If \( h_t = 0 \), then \( |X_t| = |C| \leq (|C| + 1)^{K_0} = |C| + 1 \). Suppose that Eq. (A.3) holds for any term \( t' \) such that \( h_{t'} \leq h \) for some \( h \geq 0 \). Consider a term \( t = m(t') \) such that \( h_t = h + 1 \). Then,
\[
|X_t| \leq |C| + \prod_i |X_{i,t'}| \leq |C| + ((|C| + 1)^{K_h})^k = |C| + (|C| + 1)^{K_{h+1}-1}
\]
\[
\leq (|C| + 1)^{K_{h+1}}.
\]
Therefore, Eq. (A.3) holds and
\[
|Q_0| \leq \sum_{(m(e), t) \in \Sigma_c} |X_{t[e/x]}| \leq \sum_{(m(e), t) \in \Sigma_c} (|C| + 1)^{K_{h(t[\text{res}]})}
\]
\[
\leq \sum_{\Sigma_c} (|C| + 1)^{K_H} \leq N(|C| + 1)^{K_H} \leq N(|C| + 1)^{K_{H+1}}, \tag{A.4}
\]
using \( K_H \leq K_{H+1} \) if \( k \geq 2 \). After all, from Eqs. (A.2) and (A.4), 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 \( \text{Res} \) in
\[
\mathcal{O}(kN|C|^k) \tag{A.5}
\]
time. Next, see (T3) through (T7). In (T3), \( |A| \leq |M| \cdot |C|^k \leq N|C|^k \). If \( \text{Res} \) is implemented by an appropriate data structure, then retrieving an element from \( \text{Res} \) takes
\[
\rho_{\text{Res}} = \mathcal{O}(k \log(|M| \cdot |C|^k)) = \mathcal{O}(k^2 \log N)
\]
time in (T6). In (T7), computing \( t[e/x] \) takes \( \mathcal{O}(L) \) time. Therefore, executing (T3) through (T7) takes
\[
\mathcal{O}(|A| (\rho_{\text{Q}_\Delta} + \log |M_c| + \rho_{\text{Res}} + L + \rho_{\text{Q}_\Delta}))
\]
\[
= \mathcal{O}(N|C|^k (L \log |Q| + k^2 \log N))
\]
\[
= \mathcal{O}(k^{H+1}LN|C|^k \log N), \tag{A.6}
\]
using \( k \leq L \).

See (T8) through (T16). By \( Q_\Delta \) and \( Q'_\Delta \), 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 [17] can check in \( \mathcal{O}(L) \) time whether \( t' \) is a subterm of \( t \). Constructing \( t[r' \rightarrow c] \) in (T15) takes \( \mathcal{O}(L) \) time. Computing \( Q_{\text{ans}} \cup Q_\Delta \) takes \( \mathcal{O}(L \log |Q|) \) time [17]. Therefore, the complexity of (T11) through (T16) except (T13) is
\[
\mathcal{O}(|Q|^2 (L + \rho_Z + \rho_Z + |C|(L + \rho_{Q'_\Delta})) + |Q|^2 \cdot L \log |Q|)
\]
\[
= \mathcal{O}(|Q|^2 \cdot |C| \cdot L \log |Q|)
\]
\[
= \mathcal{O}(k^{H+1}LN^2(|C| + 1)^{2k_{H+1}+1} \log N). \tag{A.7}
\]

On the other hand, in (T13), \( Z(t) \) is computed from \( Z_0 \) as follows:
\[
Z(m(e)) = Z_0(m(e)),
\]
\[
Z(m(t)) = \bigcup_{e \in Z(t_1) \times \cdots \times Z(t_n)} Z_0(m(e)).
\]
The time complexity of computing $Z(t)$ is
$$O\left(\rho_{Z_0} t |C|^{k+1} \right) = O\left(k^2 L |C|^{k+1} \log N \right).$$

The total complexity of (T13) is
$$O\left(|Q| \cdot k^2 L |C|^{k+1} \log N \right) = O\left(k^2 L N (|C| + 1)^{kH+1+k+1} \log N \right).$$  \hspace{1cm} \text{(A.8)}$$

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

Lastly, consider (S3). Let $D = \{ t \mid \tau(c/x) \Rightarrow^*_N t \}$. Then,
$$|D| \leq |X_{\tau(c/x)}| \leq |Q_0| = O(N(|C| + 1)^{kH+1}),$$

since $h_\tau \leq H$.

Figure 17 shows a procedure for determining whether $D$ contains a class. Suppose that $D_{\text{ans}}, D_\Delta, D'_\Delta$ are implemented by binomial heaps. By $D_\Delta$ and $D'_\Delta$, 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 $D_\Delta$ takes
$$\rho_{D'_\Delta} = O(L \log |D|) = O(k^{H+1} L \log N)$$
time. Computing $D \cup D_\Delta$ also takes $O(L \log |D|)$ time. Thus, executing (U2) through (U8) takes
$$O\left(|D| \cdot |Q| (L + \rho_Z + |C(L + \rho_{D'_\Delta})|) + |D| \cdot |Q| \cdot L \log |D| \right)$$
$$= O\left(|D| \cdot |Q| \cdot |C(L + \rho_{D'_\Delta})| \cdot L \log |D| \right)$$
$$= O\left(k^{H+1} L N^2 (|C| + 1)^{2kH+1+1} \log N \right).$$  \hspace{1cm} \text{(A.9)}$$

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

By Eqs. (A.1), (A.7), and (A.9), the time complexity of the algorithm is
$$O\left(k^{H+1} L N^2 (|C| + 1)^{2kH+1+1} \log N \right).$$

References