NAIST-IS-DT9661029

Doctor's Thesis

Efficient Access Control and Detection of Security Flaws under Authorizations in Object-Oriented Databases

Toshiyuki Morita

February 8, 1999

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

Doctor's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Toshiyuki Morita

Thesis committee: Minoru Ito, Professor
Shunsuke Uemura, Professor
Hiroyuki Seki, Professor

Efficient Access Control and Detection of Security Flaws under Authorizations in Object-Oriented Databases*

Toshiyuki Morita

Abstract

Access control is a key technology for providing data security in database management systems (DBMSs). Various authorization models have been investigated and proposed in order to provide an access control mechanism. An authorization is modeled as a finite set of rights. A right is generally represented as a triple (s, o, t), which means that a database user s is permitted to perform operation t on object o in databases. An access control is achieved under an authorization in the following way: When a database user invokes an access request, the DBMS permits the request if the request is permitted by the given authorization, and prohibits the request otherwise.

This thesis proposes an authorization model which is independent of any specific object-oriented database (OODB) schemas and authorization policies, and then defines an authorization specification language which is powerful enough to specify authorization policies proposed in many other papers. Also, this thesis proposes an efficient method of access control under an authorization specified by the proposed language. Furthermore, the run-time efficiency of this method is evaluated by simulating access control.

An authorization is an important and essential technology to protect secret information in databases from prohibited accesses. However, even though the DBMS enforces access control under an authorization, security flaws can occur under the given authorization. Informally, a security flaw means that a user

*Doctor's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9661029, February 8, 1999.

can obtain prohibited information by using only authorized information under an authorization. In addition to enforcing access control under an authorization, detecting a security flaw is important in order to keep the database more secure from malicious user's attack. This thesis discusses the following two problems of detecting security flaws in OODBs:

(1) The *detection problem of security flaws for OODB instances* is to decide whether or not, when a database schema S, a database instance I of S, an authorization A, and a term τ representing a program which retrieves secret information are given, a user can infer the execution result of τ under S, I, and A.

(2) The *detection problem of security flaws for OODB schemas* is to decide whether or not, when S, A, and τ are given, there exists a database instance I such that a user can infer the execution result of τ under S, I, and A.

It is shown in this thesis that the problem (1) is solvable in polynomial time in practical cases. Next, this thesis shows that the problem (2) is undecidable and proposes a decidable sufficient condition for a given method schema S to have no security flaw on τ. It is also shown that the sufficient condition is also a necessary one if the given schema is monadic (i.e., every method is unary). Furthermore, this thesis proposes an algorithm to decide the sufficient condition and then evaluates the time complexity of the algorithm. Lastly, variations of the problem (1) are mentioned.

Keywords:

object-oriented databases, access control, authorization, security flaw, term rewriting system

Acknowledgments

I have been fortunate to have received support and assistance from many individuals. I would especially like to thank Professor Minoru Ito for his invaluable support, discussions, and encouragement throughout the work. I am also grateful to Professors Shunsuke Uemura and Hiroyuki Seki for their invaluable suggestions and discussions on the work.

I would like to thank Research Associate Yasunori Ishihara for his valuable comments and continuous support throughout the work. I am also obliged to thank Research Associate Ryuichi Nakanishi for his insightful suggestions on the work.

I am grateful to Associate Professor Yuichi Kaji for his valuable comments. I am also grateful to Associate Professor Shin Ishii for his valuable support. I thank to Research Associate Hajime Watanabe for his kind support.

Lastly, I would like to thank all the members of Ito Laboratory of Nara Institute of Science and Technology.

List of Publications

Journal Papers

[1] T. Morita, Y. Ishihara, H. Seki, and M. Ito, "An authorization model for object-oriented databases and its efficient access control," IEICE Transactions on Information and Systems, vol.E81-D, no.6, pp.521–531, June 1998.

[2] T. Morita, Y. Ishihara, H. Seki, and M. Ito, "A formal approach to detecting security flaws in object-oriented databases," IEICE Transactions on Information and Systems, vol.E82-D, no.1. pp.89–98, Jan. 1999.

Workshops

[3] T. Morita, Y. Ishihara, H. Seki, and M. Ito, "An authorization model for object-oriented databases," Proceedings of the 7th Data Engineering Workshop, pp.181–186, March, 1996 (in Japanese).

[4] K. Sakaguchi, T. Morita, Y. Ishihara, H. Seki, and M. Ito, "A content-based authorization model for object-oriented databases," IEICE Technical Report, DE96-80, pp.37–42, Jan. 1997.

[5] T. Morita, Y. Ishihara, H. Seki, and M. Ito, "On the detection problem of data flaws in object-oriented databases," IEICE Technical Report, COMP97-66, pp.49–56, Nov. 1997.

[6] T. Morita, Y. Ishihara, and M. Ito, "A formal approach to detecting security flaws in object-oriented database schemas," IEICE Technical Report, COMP98-46, pp.65–72, Oct. 1998.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Access control is a key technology for providing data security in database management systems (DBMSs). Various authorization models have been investigated and proposed in order to provide an access control mechanism. An *authorization* is modeled as a finite set of *rights*. A right is generally represented as a triple (s, o, t), which means that a database user s is permitted to perform operation t on object o in databases. An access control is achieved under an authorization in the following way: When a database user invokes an access request, the DBMS permits the request if the request is permitted by the given authorization, and prohibits the request otherwise.

Authorization models for relational databases (RDBs) have been proposed [3, 6]. However, authorization models for RDBs are insufficient for object-oriented databases (OODBs), since such characteristics of OODBs as class hierarchies, inheritance, dynamic binding, and encapsulation are not incorporated in those models. Recently, various authorization models for OODBs have been proposed [9, 11, 16]. However, many of the models assume some fixed database schemas or authorization policies (e.g., "each subclass s_i $(1 \leq i \leq n)$ of a class s has all the rights that s has"), and therefore the models lack flexibility and generality. Furthermore, efficient methods of access control have scarcely been considered, although many papers propose a straightforward access control method or give a rough evaluation of the time complexity of the access control [18, 25].

This thesis proposes an authorization model which is independent of any specific OODB schemas and authorization policies, and then defines an authorization

specification language which is powerful enough to specify authorization policies proposed in many other papers [9, 11, 16]. A right is defined as a 5-tuple (s, o, t, δ, p), which means that s is *permitted* to perform operation t on object o with priority p if $\delta = +$ and is *prohibited* from performing t on o with p if $\delta = -$. Setting $\delta = -$ can specify an exception to a (positive) authorization (meaning that a subject is prohibited from performing an operation on an object). Assigning a priority to each right can simulate many policies to resolve conflicts proposed in the literature.

Specifying an authorization by *inference rules* is a good way to make the underlying authorization policy more evident and to save the storage space [6, 8].

Example 1.1: Consider the class hierarchy shown in Figure 1.1, where person, employee, student, staff, adviser, and manager are classes. For example, employee is a subclass of person. Also, consider the following authorization policy: "each subclass of person has all the rights that person has." Rather than specifying rights for employee, student, staff, adviser, and manager one by one explicitly, an inference rule is specified as follows:

$$auth(s, o, t, \delta, p) \ :- \ s \leq_{\mathsf{s}} \mathsf{person}, \ auth(\mathsf{person}, o, t, \delta, p).$$

Atom $s \leq_{\mathsf{s}} \mathsf{person}$ means that s is a subclass of person in the class hierarchy shown in Figure 1.1, and atom $auth(\mathsf{person}, o, t, \delta, p)$ means that person is permitted to perform t on o with priority p if $\delta = +$ and is prohibited from performing t on o with p if $\delta = -$. The rule states that the left-hand side $auth(s, o, t, \delta, p)$ holds if both $s \leq_{\mathsf{s}} \mathsf{person}$ and $auth(\mathsf{person}, o, t, \delta, p)$ in the right-hand side hold. □

Furthermore, this thesis proposes an efficient method of access control under an authorization specified by the proposed language. The idea of our method is (i) to partially compute inference rules and retain the results in compile-time (i.e., before an access request is given), and (ii) to perform the remainder of inference in run-time (i.e., when an access request is given). The run-time efficiency of the proposed method was evaluated by simulating access control. The simulation results conclude that the proposed method makes access control more efficient than conventional methods.

An authorization is an important and essential technology to protect secret information in databases from prohibited accesses. However, even though the

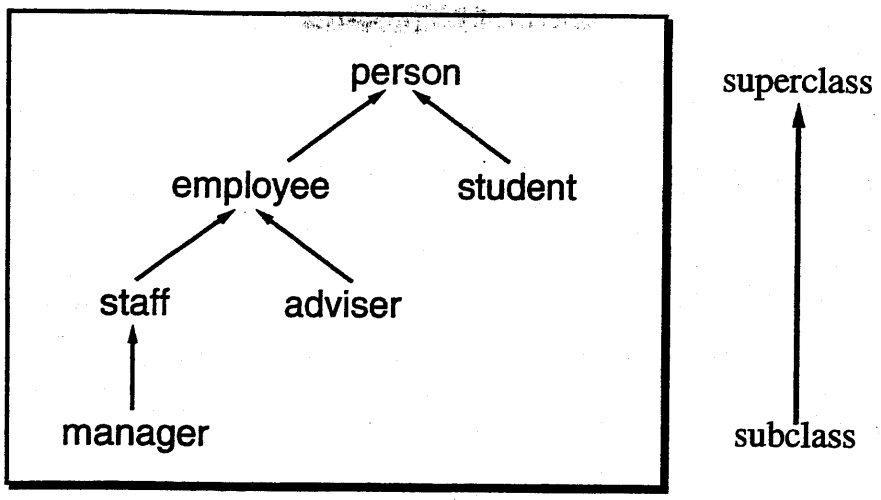


Figure 1.1 An example of a class hierarchy.

DBMS enforces access control by an authorization, *security flaws* can occur under the given authorization. Informally, a security flaw means that a user can obtain prohibited information by using only permitted information under an authorization. In addition to enforcing access control under an authorization, detecting security flaws is important in order to keep the database more secure from malicious user's attack.

Example 1.2: Suppose that a method *hostname* returns a name of host which a given employee uses, a method *service* returns a name of service as which a given host is used, and a method *admin*, whose implementation body is $admin(x) = service(hostname(x))$, returns a name of service which a given employee administrates. Also, suppose that *hostname*(Black) = Mars and *service*(Mars) = Xterm, where Black, Mars, and Xterm are objects of class employee, host, and use respectively (see Figure 1.2).

Consider the case that user Black is permitted to perform *hostname* and *admin* and is prohibited from performing *service* under an authorization. Suppose that Black knows the implementation body of permitted methods. Then, Black obtains the information that host Mars is used as X-terminal Xterm since he can infer that *service*(Mars) = Xterm by *admin*(Black) =

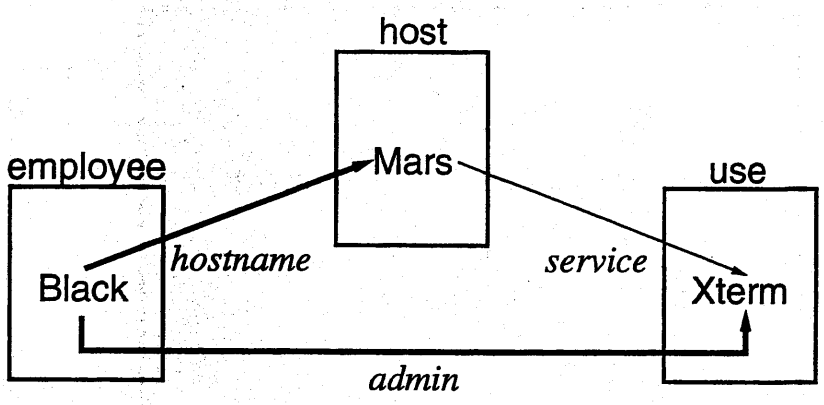


Figure 1.2 An example of method executions.

service(*hostname*(Black)) = Xterm and *hostname*(Black) = Mars, although *service* is prohibited under the given authorization. □

Recently, various models of security flaws have been discussed [10, 17, 20, 32, 33]. Generally, the user's attack is modeled by *precise inference* or *imprecise inference*. Precise inference means that a user can infer only the exact value of the result of a prohibited method. On the other hand, imprecise inference means that a user can infer several candidates of the result of a prohibited method. Example 1.2 shows an example of precise inference. Reference [32] discusses precise inference and imprecise one for OODBs. Reference [19] discusses imprecise inference for RDBs, and Reference [20] discusses a similar imprecise inference for OODBs. This thesis focuses on precise inference for OODBs.

The following problems of detecting security flaws in OODBs are considered:

(1) The *detection problem of security flaws for OODB instances* is to decide whether or not, when a database schema S, a database instance I of S, an authorization A, and a term τ to be verified (to be kept secret) are given, a user can infer the execution result of τ under S, I, and A.

(2) The *detection problem of security flaws for OODB schemas* is to decide whether or not, when S, A, and τ are given, there exists a database instance I such that a user can infer the execution result of τ under S, I, and A.

Example 1.3: In Example 1.2, let $\tau_I = service(\mathsf{Mars})$ be a term which a user wants to verify the possibility of a security flaw. Then, an example of the problem (1) is to decide whether or not a user can infer the execution result of τ_I. In this case, a security flaw on τ_I occurs. On the other hand, let $\tau_S = service(\mathsf{host})$ be a term which a user wants to verify the possibility of a security flaw. Then, an example of the problem (2) is to decide whether or not there exists a database instance such that a user can infer the execution result of a term obtained by replacing host of τ_S by an object of host. In this case, a security flaw on τ_S occurs. □

As a formal model of database schemas, this thesis adopts method schemas proposed by References [1, 2] since they have the basic features of OODBs. The semantics is simply defined based on term rewriting. In this formalization, an important point is that the above detection problems are also defined based on term rewriting.

It is shown in this thesis that the problem (1) is solvable in polynomial time in practical cases. Next, this thesis shows that the problem (2) is undecidable and proposes a decidable sufficient condition for a given method schema S to have no security flaw on τ. It is also shown that the sufficient condition is also a necessary one if the given schema is monadic (i.e., every method is unary). Furthermore, this thesis proposes an algorithm to decide the sufficient condition, and then evaluates the time complexity of the algorithm. For a monadic method schema, with the proposed algorithm, whether a security flaw on τ occurs or not is decidable in polynomial time of the size of the schema.

As a variation of the problem (1), this thesis also mentions the following problem of finding a safe authorization. An authorization A is called *safe* on a term τ if no security flaw on τ occurs under A.

(3) The *finding problem of a safe authorization* is to find, when a database schema, a database instance, an authorization, and a term to be verified are given, a maximal safe subset of the authorization on the term.

This thesis shows that the problem (3) is solvable in polynomial time in practical cases.

Chapter 2

Efficient Access Control under Authorizations in Object-Oriented Databases

2.1 Introduction

Recently, various authorization models for OODBs have been proposed. However, many of the models [7, 11, 15] are coupled with some fixed database schemas or authorization policies (e.g., "each subclass s_i $(1 \leq i \leq n)$ of a class s has all the rights that s has"), and therefore the models lack flexibility and generality. Furthermore, although some papers propose a straightforward access control method or give a rough evaluation of the complexity of the access control [18, 25], efficient methods of access control have scarcely been considered.

An authorization is modeled as a finite set of rights. A (positive) right is often represented as a triple (s, o, t), where s, o, t are an *access subject*, an *access object*, an *access type* respectively. By (s, o, t), we mean that s is permitted to perform operation t on o. In order to specify an exception to a positive right, a *negative* right is useful which prohibits a subject from performing an operation on an object. This thesis defines a right as a 5-tuple (s, o, t, δ, p), which means that s is permitted to perform operation t on o with priority p if $\delta = +$ and is prohibited from performing t on o with p if $\delta = -$. Setting $\delta = -$ can specify an exception to a positive right. Assigning a priority to each right can simulate

many policies to resolve conflicts proposed in the literature (see Section 2.2.3).

Specifying an authorization by inference rules is a good way to make the underlying authorization policy more evident and to save the storage space [6, 8]. An example of an inference rule was shown in Example 1.1. Furthermore, inference rules often have to depend on the contents of objects [4, 9].

Example 2.1: Suppose that those who belong to subclasses of **staff** in Figure 1.1 are permitted to **display** their own objects. This is specified by the following inference rule:

$$auth(w_s, w_o, \mathsf{display}, +, 200) \ :- \ w_s = w_o.\mathsf{owner},\ in(w_s, v_s),\ v_s \leq_s \mathsf{staff},$$

where w_s and w_o are instance variables denoting an access subject and an access object respectively, and v_s is a class variable denoting an access subject. Atom $w_s = w_o.\mathsf{owner}$ means that the owner of w_o is w_s, and $in(w_s, v_s)$ means that w_s is an object of class v_s. This rule states that, for any ground substitution θ, $auth(\theta(w_s), \theta(w_o), \mathsf{display}, +, 200)$ holds (i.e., the right $(\theta(w_s), \theta(w_o), \mathsf{display}, +, 200)$ is given) if all the atoms in the right-hand side hold under θ. □

In order that database administrators can specify such inference rules as Examples 1.1 and 2.1, this thesis defines an authorization specification language which is powerful enough to specify authorization policies proposed in many other papers [9, 11, 16].

Several access control algorithms have been discussed in References [5, 18, 25]. By supposing somewhat simple authorization models, Reference [25] roughly estimates the time complexity of access control for each of the supposed models. Reference [18] proposes a straightforward algorithm which decides whether a given access request is permitted or prohibited. However, References [18, 25] do not elaborate systematic methods of dealing with inference rules. A method proposed in Reference [5] is that, before an access request is given, a DBMS computes all the rights derived from inference rules. Whenever an access request is given, a DBMS may indeed quickly decide whether the request is permitted or prohibited, by referring to the computed results. However, when rights depend on the contents of objects (see Example 2.1), this method would be impractical since the size of the storage space to save the results depends on the total number of objects.

This thesis proposes a method which (i) partially computes inference rules and retains the results in compile-time (i.e., before an access request is given), and (ii) performs the remainder of inference in run-time (i.e., when an access request is given). In this method, a DBMS can efficiently decide whether a given access request is permitted or prohibited. Furthermore, the size of the storage space required by a DBMS is independent of the total number of objects. The run-time efficiency of the proposed method was evaluated by simulating access control. The simulation results conclude that the proposed method makes the access control more efficient than conventional methods.

This chapter is organized as follows: Section 2.2 introduces a database schema and then defines an authorization. Section 2.3 defines an authorization specification language and Section 2.4 proposes an efficient access control method. Section 2.5 summarizes this chapter.

2.2 Authorization Model

2.2.1 Database Schema

In this chapter, we define a simple database schema which has the main features of object-oriented data models [9, 21, 23]. Three class hierarchies[1] are useful to incorporate various specific authorization models (e.g., role-based authorization models [28] and method-based ones [16, 31]) in addition to general ones: the first class hierarchy is on access subjects and represents relationships among database users; the second one is on access objects and represents relationships among databases, classes, and objects; and the last one is on access types and represents relationships among methods and basic operations.

Definition 2.1: A *database schema* is a 5-tuple $S = (H_s, H_o, H_t, At, Ad)$ defined as follows:

1. H_s, H_o, and H_t are class hierarchies on access subjects, objects, and access types respectively.

[1] Many papers use the terminology "class hierarchy" only when the relationship is a forest. In this thesis, however, a partial order on a set is called a hierarchy even when it is not a forest.

2. At is a finite set of *attributes.*

3. Ad is a finite set of *attribute declarations.*

H_s consists of a finite set C_s of classes representing access subjects and a binary relation $\leq_s^d$ (or $\geq_s^d$) representing *is-a* relationship on C_s. Let $\leq_s^+$ (resp. $\geq_s^+$) denote the transitive closure of $\leq_s^d$ (resp. $\geq_s^d$) and $\leq_s$ (resp. $\geq_s$) the reflexive transitive closure of $\leq_s^d$ (resp. $\geq_s^d$). By $s_1 \leq_s^d s_2$ (or $s_2 \geq_s^d s_1$), we mean that s_1 is a *direct subclass* of s_2, while by $s_1 \leq_s^+ s_2$ (or $s_2 \geq_s^+ s_1$), we mean that s_1 is a (*indirect*) *subclass* of s_2. Class hierarchies on access objects and access types, denoted $H_o = (C_o, \leq_o^d)$ and $H_t = (C_t, \leq_t^d)$ respectively, are defined similarly. This thesis assumes that C_s, C_o, and C_t are disjoint; this assumption makes access control efficient (see Sections 2.2.2 and 2.4.3.2).

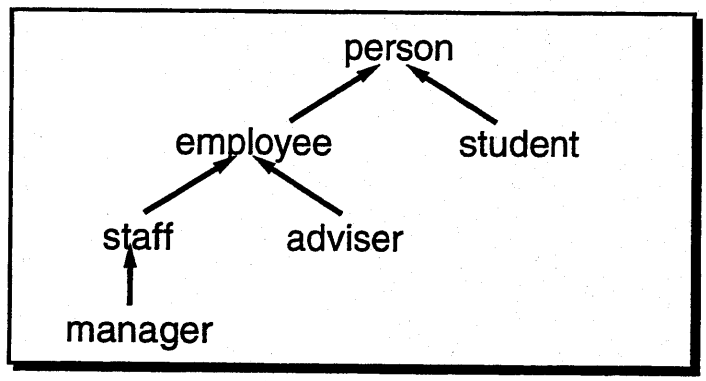
An attribute declaration has the form $(c_1, at) : c_2$, where $c_1, c_2 \in C_o$ and $at \in At$. By $(c_1, at) : c_2$, we mean that the value of attribute at of an object of c_1 must be an object of c_2 or its subclass (see the last paragraph of Section 2.2.2 for details). Classes of basic values such as integers and strings also belong to C_o and basic values are identified with objects. Attribute declarations are inherited along the class hierarchy H_o. Let $\tilde{Ad}$ be the set of *inherited attribute declarations* derived from Ad by some inheritance rule. This thesis does not assume any specific inheritance mechanism of attribute declarations since it is not our main concern. □

In this chapter, we do not incorporate method implementation bodies into a schema since a method-based authorization which depends on method implementation bodies does not be considered here. Such an authorization is discussed in Reference [31] and is used in Chapter 3.

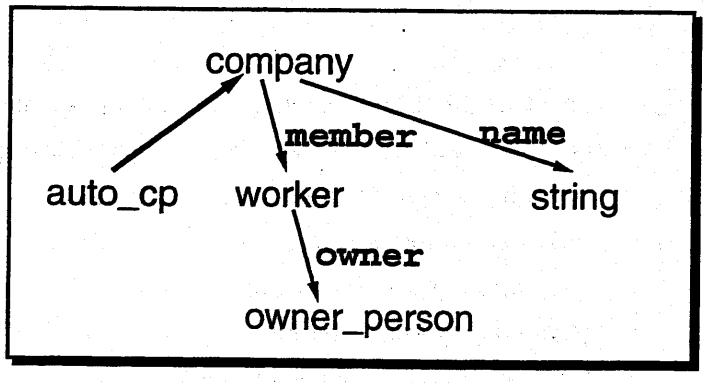
Example 2.2: Figures 2.1 and 2.2 show a database schema $S_1 = ((C_s, \leq_s^d), (C_o, \leq_o^d), (C_t, \leq_t^d), At, Ad)$. In Figure 2.2, the thick arrow from employee to person represents that employee $\leq_s^d$ person, while the thin arrow labeled by member from company to worker represents that (company, member) : worker is in Ad. Suppose

$$
\begin{aligned}
C_{\mathrm{s}} &= \{\mathsf{person}, \mathsf{student}, \mathsf{employee}, \mathsf{staff}, \mathsf{manager}, \mathsf{adviser}\},\\
C_{\mathrm{o}} &= \{\mathsf{company}, \mathsf{auto\_cp}, \mathsf{worker}, \mathsf{owner\_person}, \mathsf{string}\},\\
C_{\mathrm{t}} &= \{\mathsf{operation}, \mathsf{register}, \mathsf{modify}, \mathsf{enter}, \mathsf{display}\},\\
At &= \{\mathsf{member}, \mathsf{name}, \mathsf{owner}\},\\
Ad &= \{(\mathsf{company}, \mathsf{member}) : \mathsf{worker},\\
&\qquad (\mathsf{company}, \mathsf{name}) : \mathsf{string},\\
&\qquad (\mathsf{worker}, \mathsf{owner}) : \mathsf{owner\_person}\}.
\end{aligned}
$$

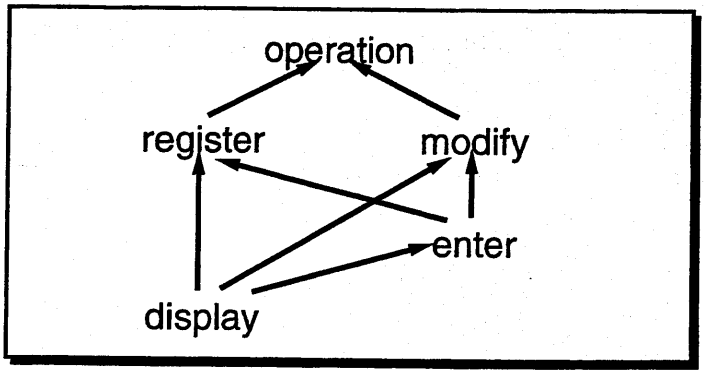
Figure 2.1 A database schema S_1.



Access Subject



Access Object



Access Type

Figure 2.2 A class hierarchy of S_1.

that, by a suitable inheritance mechanism, $\tilde{Ad}$ is derived as follows:

$$\begin{aligned}\tilde{Ad} = \{&(\mathsf{company}, \mathsf{member}) : \mathsf{worker},\\ &(\mathsf{company}, \mathsf{name}) : \mathsf{string},\\ &(\mathsf{worker}, \mathsf{owner}) : \mathsf{owner\_person},\\ &(\mathsf{auto\_cp}, \mathsf{member}) : \mathsf{worker},\\ &(\mathsf{auto\_cp}, \mathsf{name}) : \mathsf{string}\}.\end{aligned}$$

□

2.2.2 Database Instance

A *database instance*, simply called a *database*, is a directed graph $I = (O, E)$, where O is a finite set of *objects* and E is a finite set of labeled edges representing attribute-values of objects. On $O \times C$, a binary predicate in is defined such that, for any $i \in O$, there exists at most one class $c_s \in C_s$ such that $in(i, c_s)$ holds (similarly for C_o and C_t). When $in(i, c)$ holds, we say that i is an object of class c. Let

$$\begin{aligned}O_s &= \{i \in O | in(i, c_s) \text{ holds for some } c_s \in C_s\},\\ O_o &= \{i \in O | in(i, c_o) \text{ holds for some } c_o \in C_o\},\\ O_t &= \{i \in O | in(i, c_t) \text{ holds for some } c_t \in C_t\}.\end{aligned}$$

From the definition, O_s, O_o, and O_t are not necessarily disjoint. For example, there may exist an object i such that both $in(i, c_s)$ and $in(i, c_o)$ hold for some $c_s \in C_s$ and $c_o \in C_o$. In this case, i is regarded as an object of both class c_s of access subject and class c_o of access object. Thus, while keeping the class hierarchies H_s and H_o disjoint, this thesis can specify an inference rule which compares an object of access subject with an object of access object (e.g., "if a subject w_s of a subclass of staff is the owner of an object w_o, then w_s can display w_o"; see Example 2.1).

When E contains an edge labeled by at from i_1 to i_2 ($i_1, i_2 \in O_o$), we say that the value of attribute at of object i_1 is object i_2, and write $i_1.at$ to mean i_2. The attribute-value should be consistent with given attribute declarations Ad. That is, there must exist an attribute declaration $(c_1, at) : c_2 \in Ad$ satisfying all of $in(i_1, c_1)$, $in(i_2, c_2')$, and $c_2' \leq_o c_2$ for a class $c_2' \in C_o$.

2.2.3 Authorization

An authorization for an OODB is modeled as a finite set of rights. Basically, a right can be represented as a triple (s, o, t), where s is an access subject, o is an access object, and t is an access type. By (s, o, t), we mean that s is permitted to perform operation t on o. We often want to specify an exception. For example, suppose that we want to permit all subclasses of a class s except s_0 to perform an operation on an object. In such a case, a negative right $(s, o, t, -)$ is useful which specifies that subject s is prohibited from performing operation t on object o.

However, a *conflict* between a positive right $(s, o, t, +)$ and a negative right $(s, o, t, -)$ may occur. Several policies to resolve conflicts have been proposed:

- A negative right always takes precedence of a positive one [5].
- A right given explicitly takes precedence of one given implicitly, e.g., one derived from some inference rules [18].
- Each right is either a strong right or a weak one. A strong right cannot be overridden by other strong or weak rights, while a weak right can be overridden by other strong rights. Among weak rights, a more specific right takes precedence of a more general one [7].
- To each right a priority is assigned, and a right with a higher priority takes precedence of one with a lower priority [11].

Similar policies are proposed in References [6, 8]. The last policy may need to assign a priority to each right but is the most general policy of all since it can simulate the other policies by assigning a priority properly. This thesis adopts the last policy and defines the formal model as follows. Let P be a finite, totally-ordered set of priorities. Hereafter, without loss of generality, this thesis assumes that P is a finite subset of non-negative integers and that n is a higher priority than n' *iff* $n > n'$.

Definition 2.2: A *right* is a tuple $(s, o, t, \delta, p) \in (C_s \cup O_s) \times (C_o \cup O_o) \times (C_t \cup O_t) \times \{+, -\} \times P$. By (s, o, t, δ, p), we mean that s is permitted to perform operation t on object o with priority p if $\delta = +$ and is prohibited from performing t on o with p if $\delta = -$. An *authorization* is a finite set A of rights. □

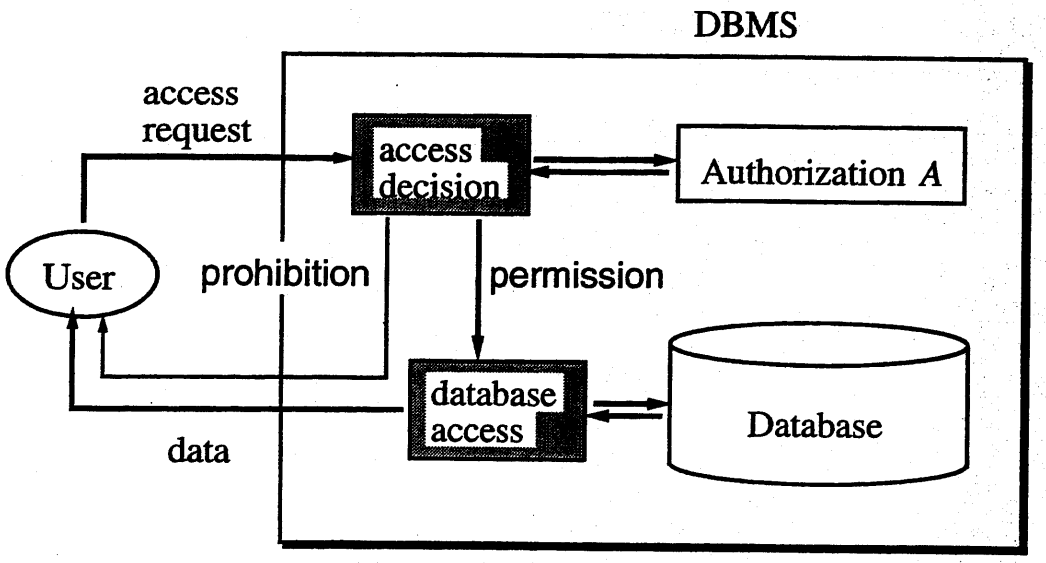


Figure 2.3 Access control.

Next, the semantics of an authorization is defined. An *access request* is a triple $(s, o, t) \in (C_s \cup O_s) \times (C_o \cup O_o) \times (C_t \cup O_t)$, denoting that s requests to perform t on o. When an access request req is invoked under an authorization, the DBMS decides whether req is permitted or prohibited under A as follows. Let

$$A_{req} = \{(s, o, t, \delta, p) \in A | req = (s, o, t)\}.$$

Suppose that there exists a unique right (req, δ_M, p_M) with the highest priority in A_{req}. Then, req is permitted if $\delta_M = +$ and is prohibited if $\delta_M = -$ (see Figure 2.3). If such a unique right does not exist, i.e., $A_{req} = \phi$ or both positive and negative rights with the (same) highest priority exist, then a conflict occurs at req[2]. When a conflict occurs at req, the administrator determines whether req is permitted or prohibited.

Example 2.3: Let A be the following finite set of rights for S_1 in Figures 2.1 and 2.2:

$$\begin{aligned} A = \{ & a_1 : (\mathsf{adviser}, \mathsf{company}, \mathsf{register}, +, 100), \\ & a_2 : (\mathsf{adviser}, \mathsf{worker}, \mathsf{display}, -, 100), \\ & a_3 : (\mathsf{adviser}, \mathsf{worker}, \mathsf{display}, -, 500), \\ & a_4 : (\mathsf{adviser}, \mathsf{worker}, \mathsf{display}, +, 300), \\ & a_5 : (\mathsf{person}, \mathsf{company}, \mathsf{operation}, +, 700), \\ & a_6 : (\mathsf{person}, \mathsf{company}, \mathsf{operation}, -, 700)\}. \end{aligned}$$

For $req_1 = (\mathsf{adviser}, \mathsf{company}, \mathsf{register})$, $A_{req_1} = \{a_1\}$. Hence, req_1 is permitted since a_1 is a positive right. For $req_2 = (\mathsf{adviser}, \mathsf{worker}, \mathsf{display})$, $A_{req_2} = \{a_2, a_3, a_4\}$. Hence, req_2 is prohibited since the priority of a_3 is the highest of all rights in A_{req_2} and a_3 is a negative right. For $req_3 = (\mathsf{person}, \mathsf{company}, \mathsf{operation})$, $A_{req_3} = \{a_5, a_6\}$. Then, a conflict occurs at req_3 since the priorities of a_5 and a_6 are the same. □

In Reference [11, 15], several interpretations of rights are stated. For example, *dynamic class semantics* has the following properties (see Reference [11]):

- A class name in a right stands for all its objects. That is, if a right (s, o, t, δ, p) $(s \in C_s)$ is given, then (i, o, t, δ, p) is also implicitly given for all

[2]For simplicity, the term "conflict" is used even if $A_{req} = \phi$.

i such that $in(i, s)$ holds (similarly for o and t). Note that the converse is not true.

- Suppose that a right a and another a_e representing an exception of a are given. If an access request matches a, then the exception a_e is not considered. For example, suppose that both rights $a = (s, o, t, +, p)$ and $a_e = (i, o, t, -, p_e)$ are given, where $in(i, s)$ holds. Then, access request (s, o, t) is permitted, while request (i, o, t) is prohibited.

In our formulation, the former property can be achieved by specifying the following inference rule (inference rules will be formally defined in Section 2.3):

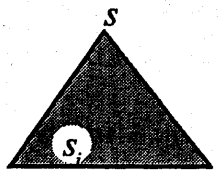
$$auth(i, o, t, \delta, p) :- in(i, s),\ auth(s, o, t, \delta, p).$$

The latter property can be achieved by assigning a higher priority to a_e than that of a.

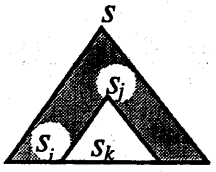
2.3 Authorization Specification

Various authorization policies that use inference rules have been proposed. In Reference [11], for example, rights $(s_i, o, t, +, p)$ are derived from $(s, o, t, +, p)$ for all subclasses s_i of s (see Figure 2.4(a)). In References [15, 16], inference is blocked by giving a negative $(s_j, o, t, -, p_j)$ $(p_j > p)$ for a subclass s_j of s, so that $(s_k, o, t, +, p)$ is implicitly ungiven for all subclasses s_k of s_j (see Figure 2.4(b)). In Reference [7], a more complicated authorization policy is proposed under the assumption that access types are fixed. In all the above policies, however, inference rules are fixed and database administrators cannot modify them at all.

References [9, 27] define the syntax of inference rules by which administrators can specify authorizations freely. In Reference [9], however, inference is fixed on the class hierarchies, and in Reference [27], recursive inference rules are disallowed. This thesis defines the syntax of inference rules such that administrators can specify the above policies [7, 9, 11, 15, 16, 27] and efficient access control can be achieved.



(a) The first policy.



(b) The second policy.

Figure 2.4 Authorization policies.

2.3.1 Syntax

Let V_δ be a set of variables over $\{+,-\}$ and V_p a set of variables over P. Also, let V_s, V_o, and V_t be sets of variables over C_s, C_o, and C_t respectively ($V = V_\mathrm{s} \cup V_\mathrm{o} \cup V_\mathrm{t} \cup V_\delta \cup V_\mathrm{p}$ and $C = C_\mathrm{s} \cup C_\mathrm{o} \cup C_\mathrm{t}$). Furthermore, W_s, W_o, and W_t be sets of variables over O_s, O_o, and O_t respectively ($W = W_\mathrm{s} \cup W_\mathrm{o} \cup W_\mathrm{t}$). A *term* is either an element of $C \cup O \cup P \cup \{+,-\} \cup V \cup W$ or $x.a$, where $x \in C_\mathrm{o} \cup O_\mathrm{o} \cup V_\mathrm{o} \cup W_\mathrm{o}$ and $a \in At$. An *atom* is $q(u_1, u_2, \ldots, u_n)$, where q is a predicate name and $u_1, u_2, \ldots, u_n$ are terms. A set of predicate names are supposed to be finite. Atoms are classified into the following four types:

(P1) Atoms on classes C_s, C_o, and C_t.

(P2) Atoms on O, P, and $\{+,-\}$.

(P3) An atom $in(i, c)$.

(P4) An atom $auth(s, o, t, \delta, p)$.

Type (P1) atom is, for example, $c_1 \leq^d_\mathrm{s} c_2$, where $c_1, c_2 \in C_\mathrm{s} \cup V_\mathrm{s}$, and $\leq^d_\mathrm{s}$ is an infix predicate. The atom has already been introduced in Section 2.2.1. Atoms $o_1 \leq^d_\mathrm{o} o_2$ and $t_1 \leq^d_\mathrm{t} t_2$, which are on class hierarchies on access objects and access types respectively, are defined similarly to $\leq^d_\mathrm{s}$. Atom $c_3.at \rightarrow_\mathrm{o} c_4$ can be also specified, where $c_3, c_4 \in C_\mathrm{o} \cup V_\mathrm{o}$, $at \in At$, and $\rightarrow_\mathrm{o}$ is an infix predicate, meaning that the domain of attribute at of class c_3 is class c_4. Any atoms on classes in distinct hierarchies are excluded (e.g., on $C_\mathrm{s} \times C_\mathrm{o}$, $C_\mathrm{s} \times C_\mathrm{o} \times C_\mathrm{t}$, and so on) since such atoms are not used in the inference rules proposed in the literature. Type (P2) atoms are, for example, arithmetic comparisons between the attribute-values of objects, and ones computing a priority or a sign representing either a positive right (i.e., $+$) or a negative one (i.e., $-$). Type (P3) atoms have already been introduced in Section 2.2.2. Type (P4) atom $auth(s, o, t, \delta, p)$ means that a right (s, o, t, δ, p) is given, where $s \in C_\mathrm{s} \cup O_\mathrm{s}$, $o \in C_\mathrm{o} \cup O_\mathrm{o}$, $t \in C_\mathrm{t} \cup O_\mathrm{t}$, $\delta \in \{+,-\}$, and $p \in P$.

Definition 2.3: Let B_c be a conjunction of (P1) atoms and B_i a conjunction of (P2) and (P3) atoms. Also, let $s_l, s_r \in C_\mathrm{s} \cup O_\mathrm{s} \cup V_\mathrm{s} \cup W_\mathrm{s}$, $o_l, o_r \in C_\mathrm{o} \cup O_\mathrm{o} \cup V_\mathrm{o} \cup W_\mathrm{o}$,

$t_l, t_r \in C_t \cup O_t \cup V_t \cup W_t$, $\delta_l, \delta_r \in \{+, -\} \cup V_\delta$, and $p_l, p_r \in P \cup V_p$. An *inference rule* is in one of the following forms:

(R1) $auth(s_l, o_l, t_l, \delta_l, p_l) \ :- \ B_i, \ B_c,$

(R2) $auth(s_l, o_l, t_l, \delta_l, p_l) \ :- \ B_i, \ B_c, \ auth(s_r, o_r, t_r, \delta_r, p_r).$

Intuitively, (R1) means that $auth(s_l, o_l, t_l, \delta_l, p_l)$ of the left-hand side holds if both B_i and B_c of the right-hand side hold. (R2) means that $auth(s_l, o_l, t_l, \delta_l, p_l)$ holds if all of B_i, B_c and $auth(s_r, o_r, t_r, \delta_r, p_r)$ hold. □

2.3.2 Semantics

Let S be a database schema. The semantics (interpretation) of type (P1) atoms is defined by S, and that of types (P2) and (P3) atoms is defined by a database instance I of S. Let $B = pred_1, pred_2, \ldots, pred_m$ be a conjunction of (P1)–(P3) atoms that contain no variables. (Thus, each $pred_i$ must be either true or false under I.) Define the *satisfaction* relation $\models$ as follows. If every $pred_i$ is true under I, then we write $I \models B$. The semantics of the type (P4) atom $auth(s, o, t, \delta, p)$ is defined recursively by a set R of inference rules as follows:

- For an inference rule in the form of (R1) and a ground substitution θ, if $I \models \theta(B_i)$ and $I \models \theta(B_c)$, then $I \models \theta(auth(s_l, o_l, t_l, \delta_l, p_l))$.
- For an inference rule in the form of (R2) and a ground substitution θ, if $I \models \theta(B_i)$, $I \models \theta(B_c)$ and $I \models \theta(auth(s_r, o_r, t_r, \delta_r, p_r))$, then $I \models \theta(auth(s_l, o_l, t_l, \delta_l, p_l))$.

When a conjunction B of atoms is true under any database instance of S, we write $\models B$. Lastly, the authorization $A^R(I)$ specified by R under I is defined as follows:

$$A^R(I) = \{(s, o, t, \delta, p) | I \models auth(s, o, t, \delta, p)\}.$$

For an access request $req = (s, o, t)$, A_{req} is defined by letting $A = A^R(I)$ in Section 2.2.3. We often write A instead of $A^R(I)$ when R and I are obvious from the context.

2.3.3 Restrictions on Inference Rules

To achieve efficient access control, this thesis will put restrictions on inference rules. Let B_s, B_o, B_t be conjunctions of atoms on C_s, C_o, C_t in (P1) respectively ($B_c = B_s, B_o, B_t$). Consider an inference rule in the following form:

(R3) $auth(s_l, o_l, t_l, \delta_l, p_l) \ :- \ B_i,\ B_s,\ B_o,\ B_t,\ auth(s_r, o_r, t_r, \delta_r, p_r),$

where $auth(s_r, o_r, t_r, \delta_r, p_r)$ is optional. A class variable v_s in V_s is called a *bridge* on subjects if v_s satisfies the following two conditions:

- v_s appears in $auth(s_l, o_l, t_l, \delta_l, p_l)$ or B_i,
- v_s appears in B_s or $auth(s_r, o_r, t_r, \delta_r, p_r)$.

Bridges on objects and types are defined similarly. This thesis supposes that each inference rule satisfies the following three restrictions:

(Q1) If s_r (resp. o_r, t_r) is an instance variable, then it is identical with s_l (resp. o_l, t_l). That is, if $s_r \in W_s$, then $s_l = s_r$; if $o_r \in W_o$, then $o_l = o_r$; and if $t_r \in W_t$, then $t_l = t_r$.

(Q2) There exists at most one bridge on subjects, objects, and types.

(Q3) Let (s, o, t) be an access request and v_s, v_o, and v_t be the bridges on subjects, objects, and types respectively. There exists a class $c_s \in C_s$ such that $\theta(v_s) = c_s$ for any ground substitution θ satisfying $\theta(s_l) = s$ and $I \models \theta(B_i)$ (similarly for bridges on objects and types).

Without (Q1), it would take $\mathcal{O}((|O|+|C|)^3)$ time to decide whether a given access request is permitted or prohibited, where $|X|$ denotes the number of elements of a set X. It is impractical to take a polynomial time of the total number of objects whenever an access request is given. The second restriction (Q2) states that B_s (resp. B_o, B_t) and B_i depend on each other via only the bridge v_s (resp. v_o, v_t). The third restriction (Q3) states that the values of the bridges are uniquely determined whenever an access request is given. Even if restrictions (Q1)–(Q3) are imposed, our specification language is powerful enough to specify the inference rules proposed in the literature.

Example 2.4: Let R_1 be the following set of inference rules for S_1 in Figures 2.1 and 2.2:

$$
\begin{aligned}
R_1 = \{\ & r_1 : auth(\mathsf{student}, \mathsf{company}, \mathsf{display}, +, 300), \\
& r_2 : auth(\mathsf{adviser}, v_\mathrm{o}, v_\mathrm{t}, +, 100), \\
& r_3 : auth(w_\mathrm{s}, w_\mathrm{o}, \mathsf{modify}, +, 500)\ :- \\
& \qquad\qquad w_\mathrm{s} = w_\mathrm{o}.\mathsf{owner},\ \ in(w_\mathrm{s}, v_\mathrm{s}),\ \ v_\mathrm{s} \leq_\mathrm{s}^+ \mathsf{employee}, \\
& r_4 : auth(v_\mathrm{s}, v_\mathrm{o}, v_\mathrm{t}, +, v_\mathrm{p})\ :- \\
& \qquad\qquad v_\mathrm{s} \leq_\mathrm{s}^+ v'_\mathrm{s},\ \ v'_\mathrm{s} \geq_\mathrm{s}^d v''_\mathrm{s},\ \ v_\mathrm{o} \underset{o}{\leftarrow} \mathsf{company.member}, \\
& \qquad\qquad auth(v''_\mathrm{s}, \mathsf{company}, v_\mathrm{t}, +, v_\mathrm{p}), \\
& r_5 : auth(v_\mathrm{s}, \mathsf{company}, v_\mathrm{t}, -, 400)\ :- \\
& \qquad\qquad v_\mathrm{s} \leq_\mathrm{s}^d v'_\mathrm{s},\ \ v'_\mathrm{s} \geq_\mathrm{s}^+ \mathsf{staff},\ \ v_\mathrm{t} \geq_\mathrm{s} \mathsf{enter}\ \},
\end{aligned}
$$

where $v_\mathrm{s}, v'_\mathrm{s}, v''_\mathrm{s} \in V_\mathrm{s}$, $v_\mathrm{o} \in V_\mathrm{o}$, $v_\mathrm{t} \in V_\mathrm{t}$, $w_\mathrm{s} \in W_\mathrm{s}$, $w_\mathrm{o} \in W_\mathrm{o}$, and $v_\mathrm{p} \in V_\mathrm{p}$. By r_1, we mean that student is permitted to display objects of company with priority 300. By r_5, we mean that $\theta(v_\mathrm{s})$ is prohibited from performing operation $\theta(v_\mathrm{t})$ on company with priority 400 if all of $I \models \theta(v_\mathrm{s} \leq_\mathrm{s}^d v'_\mathrm{s})$, $I \models \theta(v'_\mathrm{s} \geq_\mathrm{s}^+ \mathsf{staff})$, and $I \models \theta(v_\mathrm{t} \geq_\mathrm{s} \mathsf{enter})$ hold under a database instance I. In r_3, the bridge on subjects is v_s, and r_3 has no bridge on objects and types. In r_4, the bridges on subjects, objects, and types are v_s, v_o, and v_t respectively. In r_5, the bridges on subjects and types are v_s and v_t respectively, and the bridge on objects is not present. □

2.4 Access Control

When a request to access a database during an execution of a query program is invoked by a user, a DBMS decides whether the access request is permitted or prohibited under an authorization. This is called *access control* [29].

2.4.1 Straightforward Methods

There are two straightforward methods for access control:

Method I: The DBMS keeps the inference rules R intact and decides whether an access request is permitted or prohibited each time the request is given.

Method II: For all the possible access requests, the DBMS decides in advance whether the requests are permitted or prohibited, and retains the results as a table.

As for the time complexity, Method I does not have to compute anything in compile-time, but in run-time, the DBMS has to decide whether a given access request is permitted or prohibited, by using inference rules whenever the request is given, even if the same access request is given frequently. On the other hand, Method II can determine whether a given access request is permitted or prohibited only by retrieving the table in which the results computed in compile-time are stored, but it takes much time to compute all the access requests in compile-time. Moreover, it requires to reconstruct the whole table whenever inference rules or database instances are modified. As for the run-time space complexity, Method I needs a polynomial size of N to perform inference, where N is the total size of a database schema S and R (the formal definition of N is defined in Section 2.4.3), while Method II does constant space. Lastly, as for the table size to retain the results in compile-time, it takes a polynomial size of $|O|$ in Method II, while Method I does not use such a table.

The method proposed in Reference [5] is similar to Method II: a DBMS retains all rights derived from explicit rights in compile-time. In run-time, it decides whether a given access request is in these rights or not. The DBMS also retains the intermediate results of inference of rights in compile-time, and hence it suffices to partially reconstruct the table when inference rules are modified. However, Reference [5] does not consider inference dependent on the contents of objects. If such inference was incorporated, then the authorization model in Reference [5] would be impractical, because it would take a polynomial size of $|O|$ to retain all derived rights as well as the intermediate results of the inference.

2.4.2 The Proposed Method

This thesis improves Method I by adopting the following strategy:

(i) In compile-time, compute type (P1) atoms on classes in each inference rule, independent of the contents of objects. More precisely, let v_s be a bridge on subjects in rule (R3) in Section 2.3.3. For each $c_s \in C_s$, compute all

$\theta(s_r)$ such that $\theta(v_s) = c_s$ and $\models \theta(B_s)$ (similarly for bridges on objects and types).

(ii) In run-time, whenever an access request is given, perform the remainder of each inference related with the request, dependent on the contents of objects.

This method is called Method III in the following.

2.4.2.1 Pre-Computation in Compile-Time

Let $req = (s, o, t) \in (C_s \cup O_s) \times (C_o \cup O_o) \times (C_t \cup O_t)$ be an access request. By the definitions of A_{req} (see Section 2.2.3) and $A^R(I)$ (see Section 2.3.2), a right $a = (s, o, t, \delta, p)$ is in A_{req} *iff* there exists a rule r of the form (R3) in R such that the following condition (D1) holds under I. If $auth(s_r, o_r, t_r, \delta_r, p_r)$ is absent in the right-hand side of (R3), then omit condition (d) in the following (D1) and (D2).

(D1) There exists a ground substitution θ under I such that

(a) $\theta(s_l, o_l, t_l, \delta_l, p_l) = (s, o, t, \delta, p)$,

(b) $I \models \theta(B_i)$,

(c-1) $\models \theta(B_s)$,

(c-2) $\models \theta(B_o)$,

(c-3) $\models \theta(B_t)$,

(d) $I \models \theta(auth(s_r, o_r, t_r, \delta_r, p_r))$.

If there exists a bridge v_s on subjects in r, then (c-1) is equivalent to the following condition (c-1a):

(c-1a) There exist a class $c_s \in C_s$ and a ground substitution θ_v for V such that $\theta_v(v_s) = c_s$ and $\models \theta_v(B_s)$.

Note that, by (Q3) in Section 2.3.3, a class c_s which satisfies (c-1a) is uniquely determined if exists. For each class $c_s \in C_s$, define $T_{subj}(r, c_s)$ as follows:

$$T_{subj}(r, c_s) = \{\theta_v(s_r) | \theta_v(v_s) = c_s \text{ and } \models \theta_v(B_s)\},$$

where θ_v is an arbitrary substitution for V. On the other hand, if there exists no bridge on subjects in r, then (c-1) is equivalent to the following condition (c-1b):

(c-1b) There exists a ground substitution θ_v such that $\models \theta_v(B_s)$.

Then, define $T_{subj}(r)$ as follows:

$$T_{subj}(r) = \{\theta_v(s_r) | \models \theta_v(B_s)\},$$

where θ_v is an arbitrary substitution for V. T_{obj} and T_{type} are defined in the same way. Therefore, (D1) is equivalent to the following condition (D2):

(D2) There exist $c_s, c_o, c_t, s_j, o_j, t_j$ and ground substitutions θ_w for W and θ_v for V such that

(a) $\theta_w \circ \theta_v(s_l, o_l, t_l, \delta_l, p_l) = (s, o, t, \delta, p)$,

(b) $I \models \theta_w \circ \theta_v(B_i)$,

(c-1) $\theta_v(v_s) = c_s$ and $s_j \in T_{subj}(r, c_s)$ if there exists a bridge v_s on subjects, and $s_j \in T_{subj}(r)$ otherwise,

(c-2) $\theta_v(v_o) = c_o$ and $o_j \in T_{obj}(r, c_o)$ if there exists a bridge v_o on objects, and $o_j \in T_{obj}(r)$ otherwise,

(c-3) $\theta_v(v_t) = c_t$ and $t_j \in T_{type}(r, c_t)$ if there exists a bridge v_t on types, and $t_j \in T_{type}(r)$ otherwise,

(d) $I \models \theta_w(auth(s_j, o_j, t_j, \theta_v(\delta_r), \theta_v(p_r)))$,

where "o" is a composition operator. Retrieving T_{subj}, T_{obj}, T_{type} can decide whether (D2) holds or not. Hence, before an access request is given, these sets are computed in advance for any possible c_s, c_o, c_t of values of bridges if exist. When atom *auth* is not present in the right-hand side of a rule r, if $\theta_v(v_s) = c_s$ and $\models \theta_v(B_s)$, then $T_{subj}(r, c_s) = \{\mathbf{true}\}$, where **true** is a special symbol denoting true, and otherwise, $T_{subj}(r, c_s) = \phi$. T_{obj} and T_{type} are defined in the same way.

Example 2.5: For S_1 in Figures 2.1 and 2.2 and R_1 in Example 2.4, compute T_{subj}, T_{obj}, T_{type}. For $r_4 \in R_1$, first consider access subject. Since v_s is a bridge and $B_s = v_s \leq_s^+ v_s', v_s' \geq_s^d v_s''$,

$$
\begin{aligned}
T_{\text{subj}}(r_4, \text{person}) &= \phi, \\
T_{\text{subj}}(r_4, \text{employee}) &= \{\text{employee}, \text{student}\}, \\
T_{\text{subj}}(r_4, \text{student}) &= \{\text{employee}, \text{student}\}, \\
T_{\text{subj}}(r_4, \text{staff}) &= \{\text{employee}, \text{student}, \text{staff}, \text{adviser}\}, \\
T_{\text{subj}}(r_4, \text{adviser}) &= \{\text{employee}, \text{student}, \text{staff}, \text{adviser}\}, \\
T_{\text{subj}}(r_4, \text{manager}) &= \{\text{employee}, \text{student}, \text{staff}, \text{adviser}, \text{manager}\}.
\end{aligned}
$$

For the access object in r_4, since v_o is a bridge and $B_o = v_o \xleftarrow[o]{} \text{company.member}$,

$$
\begin{aligned}
T_{\text{obj}}(r_4, \text{company}) &= \phi, \\
T_{\text{obj}}(r_4, \text{auto\_cp}) &= \phi, \\
T_{\text{obj}}(r_4, \text{worker}) &= \{\text{company}\}.
\end{aligned}
$$

For the access type in r_4, since v_t is a bridge and $B_t = true$,

$$
\begin{aligned}
T_{\text{type}}(r_4, \text{operation}) &= \{\text{operation}\}, \\
T_{\text{type}}(r_4, \text{register}) &= \{\text{register}\}, \\
T_{\text{type}}(r_4, \text{modify}) &= \{\text{modify}\}, \\
T_{\text{type}}(r_4, \text{enter}) &= \{\text{enter}\}, \\
T_{\text{type}}(r_4, \text{display}) &= \{\text{display}\}.
\end{aligned}
$$

For the other rules, T_{subj}, T_{obj}, T_{type} are computed in the same way. Table 2.1 shows the results for all rules in R_1 (r_1 and r_2 are omitted). □

2.4.2.2 Computation in Run-Time

For an access request $req = (s, o, t)$, A_{req} can be computed from T_{subj}, T_{obj}, T_{type} by testing recursively whether condition (D2) holds or not.

Example 2.6: Consider S_1 in Figures 2.1 and 2.2, R_1 in Example 2.4, and T_{subj}, T_{obj}, T_{type} in Table 2.1. Suppose that $in(\text{bob}, \text{staff})$ and $in(\text{worker\_bob}, \text{worker})$ hold in access subjects and objects respectively, and bob = worker_bob.owner under a database instance I_1 of S_1. Let $req_1 = (\text{adviser}, \text{company}, \text{register})$ be an access request. For $r_2 \in R_1$, clearly $a_1 = (\text{adviser}, \text{company}, \text{register}, +, 100)$ is in A_{req_1}. For r_5,

Table 2.1 Example of tables T_{subj}, T_{obj}, T_{type}.

(a) Table T_{subj}.

	r_3	r_4	r_5
person	ϕ	ϕ	ϕ
employee	ϕ	employee,student	**true**
student	ϕ	employee,student	**true**
staff	**true**	employee,student, staff,adviser	**true**
adviser	**true**	employee,student, staff,adviser	**true**
manager	**true**	employee,student, staff,adviser,manager	ϕ

(b) Table T_{obj}.

	r_3	r_4	r_5
company	**true**	ϕ	**true**
auto_cp		ϕ	
worker		company	

(c) Table T_{type}.

	r_3	r_4	r_5
operation	**true**	operation	**true**
register		register	**true**
modify		modify	**true**
enter		enter	**true**
display		modify	ϕ

$$T_{\text{subj}}(r_5, \text{adviser}) = \{\textbf{true}\},$$
$$T_{\text{obj}}(r_5) = \{\textbf{true}\},$$
$$T_{\text{type}}(r_5, \text{register}) = \{\textbf{true}\}.$$

Hence, $a_2 = (\text{adviser}, \text{company}, \text{register}, -, 400)$ is in A_{req_1}. By computing with the other rules, we have $A_{req_1} = \{a_1, a_2\}$. Since a_2 is negative with the highest priority in A_{req_1}, req_1 is prohibited.

On the other hand, let $req_2 = (\text{bob}, \text{worker\_bob}, \text{modify})$ be an access request. For r_3, bob = worker_bob.owner and $in(\text{bob}, \text{staff})$ are true, so the value of the bridge v_s of access subject is staff. Since

$$T_{\text{subj}}(r_3, \text{staff}) = \{\textbf{true}\},$$
$$T_{\text{obj}}(r_3) = \{\textbf{true}\},$$
$$T_{\text{type}}(r_3) = \{\textbf{true}\},$$

we know that $I_1 \models auth(\text{bob}, \text{worker\_bob}, \text{modify}, +, 500)$. Hence, $a_3 = (\text{bob}, \text{worker\_bob}, \text{modify}, +, 500)$ is in A_{req_2}. Since $in(\text{bob}, \text{staff})$ and $in(\text{worker\_bob}, \text{worker})$, whether request (staff, worker_bob, modify) is permitted or prohibited also has to be tested. For r_4, the value of bridges is (staff, worker_bob, modify) and the corresponding portion of the table for r_4 are

$$T_{\text{subj}}(r_4, \text{staff}) = \{\text{employee}, \text{student}, \text{staff}, \text{adviser}\},$$
$$T_{\text{obj}}(r_4, \text{worker}) = \{\text{company}\},$$
$$T_{\text{type}}(r_4, \text{modify}) = \{\text{modify}\}.$$

Hence, $a_4 = (\text{bob}, \text{worker\_bob}, \text{modify}, +100)$ is in A_{req_2} since

$$I_1 \models auth(\text{adviser}, \text{company}, \text{modify}, +, 100)$$

by r_2. By computing with the other rules, we have $A_{req_2} = \{a_3, a_4\}$. Thus, a_3 is positive with the highest priority in A_{req_2}, and hence req_2 is permitted. □

2.4.3 Complexity

Let $S = ((C_s, \leq_s^d), (C_o, \leq_o^d), (C_t, \leq_t^d), At, Ad)$ be a database schema and R a finite set of inference rules. Define the size $\|S\|$ of S as follows:

$$\|S\| = |C_s| + |C_o| + |C_t| + |\leq_s^d| + |\leq_o^d| + |\leq_t^d| + |At| + |Ad|.$$

The total size of S and R is denoted by

$$N = \|S\| + \|R\|,$$

where $\|R\|$ is the number of atoms appearing in R.

Let $f_c(N)$ and $f_i(N, |O|)$ denote the time to find all the ground substitutions θ such that $\models \theta(B_c)$ and $I \models \theta(B_i)$ under I respectively. In Sections 2.4.3.1 and 2.4.3.2, $f_c(N)$ and $f_i(N, |O|)$ will be evaluated under certain assumptions.

2.4.3.1 Compile-Time Complexity

Let r be an inference rule of the form (R3). For each $c_s \in C_s$, (i) compute all s' such that $\theta_v(v_s) = c_s$ (v_s is the bridge), $\models \theta_v(B_s)$, and $\theta_v(s_r) = s'$ for some ground substitution θ_v for V, and (ii) add all such s' to $T_{\text{subj}}(r, c_s)$ (see the definition of T_{subj} in Section 2.4.2.1). For each $c_o \in C_o$ and $c_t \in C_t$, tables $T_{\text{obj}}(r, c_o)$ and $T_{\text{type}}(r, c_t)$ are constructed similarly. Step (i) can be executed in

$$\mathcal{O}(f_c(N))$$

time for each c_s. If T_{subj} is implemented as a 2-3 tree, then step (ii) can be executed in

$$\mathcal{O}((|C| + \|R\|)(\log \|R\| + \log |C|)) \leq \mathcal{O}(N \log N)$$

time. Therefore, the time complexity is

$$\mathcal{O}(\|R\||C|(f_c(N) + N \log N)) = \mathcal{O}(N^2(f_c(N) + N \log N)).$$

Theorem 2.1: Let $f_c(N)$ be the time complexity to compute the set of classes satisfying B_c for an inference rule. Tables T_{subj}, T_{obj}, T_{type} can be obtained in

$$\mathcal{O}(N^2(f_c(N) + N \log N))$$

time. □

This thesis evaluates $f_c(N)$ in the case that each inference rule means traversing C_s from some class v_1 as the initial point through $v_2, v_3, \cdots, v_n$ to v_{n+1} as the end point linearly, without visiting the same class repeatedly. This strategy is valid if an inference rule r satisfies the following condition on B_s:

(B1) $(B_{\rm s} = v_1 \leq^d_{\rm s} v_2,\ \ldots,\ v_n \leq^d_{\rm s} v_{n+1})$
$\wedge(v_1 \in C_{\rm s} \cup \{s_l\}) \wedge (v_2, \cdots, v_n \in C_{\rm s} \cup V_{\rm s} - \{s_l, s_r\}) \wedge (v_{n+1} \in C_{\rm s} \cup \{s_r\})$,

where $v_i \neq v_j$ if $i \neq j$, and the number of atoms in each inference rule is bounded by $\mathcal{O}(\|R\|)$, i.e., $n \leq \mathcal{O}(N)$. To the author's knowledge, most of the inference rules in the literature satisfy this condition. First, by $v_1 \in C_{\rm s} \cup \{s_l\}$, all c_2 satisfying $c_1 \leq^d_{\rm s} c_2$ can be computed, where $c_1 = \theta_v(v_1) \in C_{\rm s}$ for some ground substitution θ_v for V. Next, all c_3 satisfying $c_2 \leq^d_{\rm s} c_3$ for some c_2 obtained above can be computed. Repeating this, all c_{n+1} can be obtained in $\mathcal{O}(N^2)$ time, where $c_{n+1} = \theta_v(v_{n+1})$ and $\theta_v(B_{\rm s})$ holds for some θ_v. If $B_{\rm o}$ and $B_{\rm t}$ satisfy similar conditions to (B1), then the time complexity in Theorem 2.1 becomes $\mathcal{O}(N^2(f_{\rm c}(N) + N \log N)) = \mathcal{O}(N^4)$.

2.4.3.2 Run-Time Complexity

When an access request $req = (s, o, t)$ is given, rights (s, o, t, δ, p) such that $I \models auth(s, o, t, \delta, p)$ holds are computed. For each inference rule, it takes

$$\mathcal{O}(f_{\rm i}(N, |O|))$$

to compute $B_{\rm i}$. Since the values of the bridges on subjects, objects, and types are obtained by computing $B_{\rm i}$ by (Q3), it takes $\mathcal{O}(\log N)$ to retrieve $T_{\rm subj}$, $T_{\rm obj}$, $T_{\rm type}$ by using the values of the bridges, instead of computing $B_{\rm c}$. For all combinations $(s', o', t') \in C_{\rm s} \times C_{\rm o} \times C_{\rm t}$ obtained from $T_{\rm subj}$, $T_{\rm obj}$, $T_{\rm type}$, if $I \models auth(s', o', t', \delta', p')$ holds for some δ' and p', then there exist $\delta \in \{+, -\}$ and $p \in P$ such that $I \models auth(s, o, t, \delta, p)$ holds, and hence (s, o, t, δ, p) is in A_{req}. It takes

$$\mathcal{O}(\log N + |C|^3 \log(\|R\| + |C|)) = \mathcal{O}(N^3 \log N)$$

to add rights to A_{req}, without testing the same rights repeatedly. This is repeated for each rule. Therefore, it takes

$$\mathcal{O}(N^4(f_{\rm i}(N, |O|) + N^3 \log N))$$

to compute A_{req} since $|A_{req}| \leq \mathcal{O}((|C| + \|R\|)^3) = \mathcal{O}(N^3)$ by (Q1) in Section 2.3.3 and $\|R\| \leq \mathcal{O}(N)$.

Table 2.2 Time complexity.

(a) Compile-time.

Method	TIME
I	–
II	$\mathcal{O}(\|O\|^3 N^4 (f_c(N) + f_i(N, \|O\|) + N^3 \log N))$
III	$\mathcal{O}(N^2 (f_c(N) + N \log N))$

(b) Run-time.

Method	TIME
I	$\mathcal{O}(N^4 (f_c(N) + f_i(N, \|O\|) + N^3 \log N))$
II	$\mathcal{O}(\log \|O\|)$
III	$\mathcal{O}(N^4 (f_i(N, \|O\|) + N^3 \log N))$

Table 2.3 Run-time space complexity and table size.

Method	SPACE	TABLE SIZE
I	$\mathcal{O}(N^3)$	–
II	$\mathcal{O}(1)$	$\mathcal{O}(\|O\|^3)$
III	$\mathcal{O}(N^3)$	$\mathcal{O}(N^3)$

Theorem 2.2: For a given access request req, A_{req} can be obtained in

$$\mathcal{O}(N^4(f_{\mathrm{i}}(N,|O|)+N^3\log N))$$

time from T_{subj}, T_{obj}, T_{type}. □

This thesis evaluates $f_{\mathrm{i}}(N,|O|)$ in the case that when an access request is given, the values of all the instance variables in an inference rule can be uniquely determined immediately in some fixed order. This strategy is valid if an inference rule r satisfies the following condition on B_{i}:

$$\begin{aligned}(\mathrm{B2})\ (B_{\mathrm{i}} = (&w_{\mathrm{o}_1}.at_1 = w_{\mathrm{o}_2},\ldots,w_{\mathrm{o}_n}.at_n = w_{\mathrm{o}_{n+1}}, in(w_{\mathrm{o}_1},v_{\mathrm{o}_1}),\ldots,in(w_{\mathrm{o}_{n+1}},v_{\mathrm{o}_{n+1}}),\\ &in(w_{\mathrm{s}_1},v_{\mathrm{s}_1}),in(w_{\mathrm{t}_1},v_{\mathrm{t}_1})))\\ \wedge\ &(o_l = w_{\mathrm{o}_1}) \wedge (w_{\mathrm{o}_i} \in O_{\mathrm{o}} \cup W_{\mathrm{o}}) \wedge (v_{\mathrm{o}_i} \in C_{\mathrm{o}} \cup V_{\mathrm{o}})\\ \wedge\ &(s_l = w_{\mathrm{s}_1} \in O_{\mathrm{s}} \cup W_{\mathrm{s}}) \wedge (v_{\mathrm{s}_1} \in C_{\mathrm{s}} \cup V_{\mathrm{s}})\\ \wedge\ &(t_1 = w_{\mathrm{t}_1} \in O_{\mathrm{s}} \cup W_{\mathrm{t}}) \wedge (v_{\mathrm{t}_1} \in C_{\mathrm{t}} \cup V_{\mathrm{t}}).\end{aligned}$$

Let (s,o,t) be an access request such that $s \in O_{\mathrm{s}}, o \in O_{\mathrm{o}}$, and $t \in O_{\mathrm{t}}$. For a ground substitution θ, the value $\theta(w_{\mathrm{o}_1})$ of the variable w_{o_1} must be o since $o_l = w_{\mathrm{o}_1}$ ($\theta(w_{\mathrm{s}_1}) = s$ and $\theta(w_{\mathrm{t}_1}) = t$ for the same reason). Then, $\theta(w_{\mathrm{o}_2})$ such that $\theta(w_{\mathrm{o}_1}.at_1) = \theta(w_{\mathrm{o}_2})$ can be uniquely obtained in constant time. Repeatedly, $\theta(w_{\mathrm{o}_3}), \theta(w_{\mathrm{o}_4}), \ldots, \theta(w_{\mathrm{o}_{n+1}})$ can be obtained. Since the values of each variable $w_{\mathrm{o}_1}, w_{\mathrm{o}_2}, \ldots, w_{\mathrm{o}_{n+1}}$ over objects have been obtained, $\theta(v_{\mathrm{o}_1})$ such that $\theta(in(w_{\mathrm{o}_1}, v_{\mathrm{o}_1}))$ holds can be uniquely obtained in constant time ($\theta(v_{\mathrm{s}_1})$ and $\theta(v_{\mathrm{t}_1})$ can be obtained in the same way). Hence, it takes $\mathcal{O}(N)$ time to decide whether $I \models \theta(B_{\mathrm{i}})$ holds or not.

Table 2.2 shows the time complexities of Methods I–III, and Table 2.3 shows the run-time space complexities and the table sizes to retain the results of precomputation of Methods I–III.

2.4.4 Simulation Results

The run-time efficiency of Method III was evaluated by simulating access control under the following assumptions. Class hierarchies are balanced binary trees with multiple inheritances. In inference rules, content-dependent atoms are not specified, in order to make a clear distinction between Methods I and III. Tables 2.4(a)

Table 2.4 Experimental data on access control.

(a) Class hierarchies of height two.

The numbers of rules in R	Method I (sec/request)	Method III (sec/request)
2	0.00047	0.00033
3	0.39309	0.32373
4	0.86531	0.74884
5	1.45473	1.14462
6	2.66962	1.99640

(b) Class hierarchies of height three.

The numbers of rules in R	Method I (sec/request)	Method III (sec/request)
2	0.00037	0.00031
3	2.57375	1.86211
4	15.68312	11.79447
5	45.88496	29.30559
6	95.74206	53.06423

(c) Example for complicated rules.

Method I (sec/request)	Method III (sec/request)
0.18584	0.07363

and (b) show the average time for all possible access requests when the class hierarchies are of height two and height three respectively. The experimental results conclude that Method III is faster than Method I. Also, the more complicated inference rules is specified, the faster Method III computes relatively to Method I. Especially, when recursive inference rules including atoms that traverse class hierarchies up and down are specified, there is a great difference between Methods I and III. For example, suppose that the following rules are specified under class hierarchies of height two:

$$
\begin{aligned}
&auth(\mathsf{s00}, \mathsf{o00}, \mathsf{t00}, +, 20), \\
&auth(v_{\mathrm{s}}, v_{\mathrm{o}}, v_{\mathrm{t}}, v_{\delta}, v_{\mathrm{p}}) \ :- \ v_{\mathrm{s}} \leq_{\mathrm{s}} v'_{\mathrm{s}},\ v_{\mathrm{o}} \leq^{+}_{\mathrm{o}} v'_{\mathrm{o}},\ v'_{\mathrm{o}} \geq_{\mathrm{o}} v''_{\mathrm{o}},\ v_{\mathrm{t}} \leq_{\mathrm{t}} v'_{\mathrm{t}}, \\
&\qquad\qquad\qquad\qquad\qquad\qquad auth(v'_{\mathrm{s}}, v''_{\mathrm{o}}, v'_{\mathrm{t}}, v_{\delta}, v_{\mathrm{p}}),
\end{aligned}
$$

where s00, o00, t00 are constants. From experimental data for the above rules, Method III is more than twice as fast as Method I (see Table 2.4(c)).

2.5 Conclusions

In this chapter, we have proposed an authorization model which is independent of OODB schemas and authorization policies, and have defined an authorization specification language which is powerful enough to specify authorization policies proposed in the literature. Furthermore, an efficient access control method has been proposed. The proposed method partially computes inferences of authorizations from S and R in compile-time and decides whether a given access request is permitted or prohibited by using the results of compile-time. The time complexities are $\mathcal{O}(N^2(f_{\mathrm{c}}(N) + N \log N))$ for compile-time and $\mathcal{O}(N^4(f_{\mathrm{i}}(N, |O|) + N^3 \log N))$ for run-time. From the simulation results, the proposed method makes the access control more efficient than conventional methods.

As a future work, this thesis intends to extend the authorization model to be able to specify the situation where users can grant their rights to others and revoke their rights from others [3], and then intends to propose an efficient access control method in such a model.

Chapter 3

Detection of Security Flaws under Authorizations in Object-Oriented Databases

3.1 Introduction

In Chapter 2, it is stated that an authorization is an important and essential technology to protect secret information in databases from prohibited accesses. However, even though the DBMS enforces access control by an authorization, security flaws can occur under the given authorization. Informally, a security flaw means that a user can obtain prohibited information by using only permitted information under an authorization. In addition to enforcing access control under an authorization, detecting security flaws is important in order to keep the database more secure from malicious user's attack.

A general authorization model has been provided in Chapter 2. In order to investigate detection of a security flaw, this chapter focuses on what a user u is permitted to invoke and what u is prohibited from invoking. In this chapter, the following authorization which simplifies one provided in Chapter 2 and concentrates on a method invocation is adopted. An authorization A for a user u is represented as a set of $(m, (c_1, c_2, \ldots, c_n))$, which means that u is permitted to invoke method m on any tuple $(o_1, o_2, \ldots, o_n)$ of objects such that o_i is an object of class c_i for each i $(1 \leq i \leq n)$. This thesis assumes the following database

management policies. Let $(m, (c_1, c_2, \ldots, c_n))$ be in an authorization for a user u.

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

(2) If m is a primitive method (i.e., m is a base method; see Section 3.2.1), then the type declaration of m at $(c_1, c_2, \ldots, c_n)$ is open to u. If m is not primitive (i.e., m is a user method; see Section 3.2.1), then its external specification (its implementation body) is open to u.

An example of a security flaw was shown in Example 1.2. The following more complicated example, which may be intensional, also shows a security flaw.

Example 3.1: Let m, m_1, m_2 be unary methods, c a class, and o an object of class c. Suppose that the authorization A for a user u is $\{(m_1, c), (m_2, c)\}$, and the implementation bodies of m_1 and m_2, which u knows by Policy (2), are $m_1(x) = m(m(m(x)))$ and $m_2(x) = m(m(m(m(m(x)))))$ respectively. Also, suppose that $m_1(o) = o$ and $m_2(o) = o$, which u knows by Policy (1). Then, u can infer that $m(o) = o$ although m is not contained in A. □

The following problems of detecting security flaws in OODBs are considered:

(1) The *detection problem of security flaws for OODB instances* is to decide whether or not, when a method schema S, a database instance I of S, an authorization A, and a term τ to be verified (to be kept secret) are given, a user can infer the execution result of τ under S, I, and A.

(2) The *detection problem of security flaws for OODB schemas* is to decide whether or not, when S, A, and τ are given, there exists a database instance I such that a user can infer the execution result of τ under S, I, and A.

In this chapter, as a formal model of database schemas, method schemas proposed by References [1, 2] are adopted since they have the basic features of OODBs as method overloading, dynamic binding, and complex objects. The semantics is simply defined based on term rewriting. In this formalization, an important point is that the above detection problems are also defined based on term rewriting.

It is shown in this thesis that the problem (1) is reducible to the congruence closure problem [14, 24] and is solvable in polynomial time in practical cases. Next, this thesis shows that the problem (2) is undecidable and proposes a decidable sufficient condition for a given method schema S to have no security flaw on τ. Intuitively, the main idea of a sufficient condition is to introduce new rewriting rules at class level approximating rewriting rules at instance level. It is also shown that the sufficient condition is also a necessary one if the given schema is monadic (i.e., every method is unary). Furthermore, this thesis proposes an algorithm to decide the sufficient condition, and then evaluates the time complexity of the algorithm. For a monadic method schema, with the proposed algorithm, whether a security flaw on τ occurs or not is decidable in polynomial time of the size of the schema.

As a variation of the problem (1), this thesis also mentions the following problem of finding a safe authorization. An authorization A is called safe on a term τ if no security flaw on τ occurs under A:

(3) The *finding problem of a safe authorization* is to find, when a database schema, a database instance, an authorization, and a term to be verified are given, a maximal safe subset of the authorization on the term.

This thesis shows that the problem (3) is solvable in polynomial time in practical cases, using the result of the problem (1).

Various models of security flaws have been discussed [10, 17, 19, 20, 32, 33]. Generally, user's attack is modeled by precise inference or imprecise inference. Precise inference means that a user can infer only the exact value of the result of a prohibited method. On the other hand, imprecise inference means that a user can infer several candidates of the result of a prohibited method. Examples 1.2 and 3.1 show an example of precise inference. Reference [32] discusses precise inference and imprecise one for OODBs. Reference [19] discusses imprecise inference for relational databases, and Reference [20] does for OODBs. This thesis focuses on precise inference for OODBs.

In Reference [32], security flaws are classified into inferability and controllability. Roughly speaking, inferability means that a user can infer the returned value of a method invocation, and controllability means that a user can control (alter arbitrarily) the attribute-value of an object in a database instance. Since our

query language does not support update operations for database instances, only inferability is considered as security flaws in this thesis. However, since our query language supports recursion while the one in Reference [32] does not, detecting inferability in our formalization is not trivial.

This chapter is organized as follows: Section 3.2 introduces method schemas as a formal database model and considers method executions. Section 3.3 defines an authorization and analyzes user's inference under a given authorization. Sections 3.4–3.6 discuss the above problems (1)–(3) respectively. Section 3.7 summarizes this chapter.

3.2 Method Schemas

3.2.1 Syntax

Before defining the syntax of method schemas, some notations are introduced. Let F be a family of disjoint finite sets $F_0, F_1, F_2, \ldots$, where F_n $(n = 0, 1, 2, \ldots)$ is a set of function symbols of arity n. For a countable set X of variables, let $T_F(X)$ denote the set of all terms freely generated by F and X. For a term $t \in T_F(X)$ and variables x_i $(1 \leq i \leq n)$ in X, let $t[t_1/x_1, t_2/x_2, \ldots, t_n/x_n]$ denote the term obtained by replacing every x_i in t with a term t_i $(1 \leq i \leq n)$. For example, $f(x_1, g(x_1, x_2))[a/x_1, b/x_2] = f(a, g(a, b))$.

For a term t, define the set of *occurrences* $OC(t)$ as the smallest set of sequences of positive integers which satisfies the following two conditions:

- $\varepsilon \in OC(t)$, where ε is the empty sequence.
- If $r \in OC(t_i)$, then $i \cdot r \in OC(f(t_1, t_2, \ldots, t_n))$ $(1 \leq i \leq n)$, where the center dot "$\cdot$" represents the concatenation of sequences.

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

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

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

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

Definition 3.1: A *base method definition of* m_b *at* $(c_1, c_2, \ldots, c_n)$ is an expression

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

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

Let o_i be an object of class c_i $(1 \leq i \leq n)$ (see Definitions 3.3 and 3.5 for formal definitions). Informally, the above base method definition declares that the application of m_b to $o_1, o_2, \ldots, o_n$ results in an object of c or its subclass.

Definition 3.2: A *user method definition of* m_u *at* $(c_1, c_2, \ldots, c_n)$ is an expression

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

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

Let o_i be an object of c_i $(1 \leq i \leq n)$. The above user method definition states that the application of m_u to $o_1, o_2, \ldots, o_n$ results in term rewriting starting from $t[o_1/x_1, o_2/x_2, \ldots, o_n/x_n]$. The formal definition is presented in Section 3.2.3.

Definition 3.3: A *method schema*, which is originally introduced by Abiteboul et al. [1, 2], is a 5-tuple $(C, \leq, M, \Sigma_b, \Sigma_u)$ defined as follows:

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

5. Σ_u is a set of user method definitions.

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

Example 3.2: Figure 3.1 shows an example of a method schema S_2 which represents relationships on users and computer systems in an office. For example, a user method $admin(x)$ is supposed to return a service name of a host which a given employee x administrates. □

3.2.2 Method Inheritance

Method definitions are inherited along the class hierarchy. For example, suppose that method definitions of m are given at (staff, employee) and (employee, employee) (see Figure 3.2), where class staff is a subclass of employee. Intuitively, the inherited method definition of m at (staff, staff) should be the definition at (staff, employee), not the one at (employee, employee), since (staff, employee) is smaller than (employee, employee) with respect to $\leq$. The following formally defines the inheritance of a method definition.

Definition 3.4: Let $(C, \leq, M, \Sigma_b, \Sigma_u)$ be a method schema, $m_b \in M_{b,n}$, and $c_1, c_2, \ldots, c_n \in C$. Suppose that $(m_b, (c'_1, c'_2, \ldots, c'_n \rightarrow c')) \in \Sigma_b$ is the base method definition of m_b at the "componentwise smallest" $(c'_1, c'_2, \ldots, c'_n)$ above $(c_1, c_2, \ldots, c_n)$ in the sense that whenever $(m_b, (c''_1, c''_2, \ldots, c''_n \rightarrow c'')) \in \Sigma_b$ and $c_i \leq c''_i$ $(1 \leq i \leq n)$, it is the case that $c'_i \leq c''_i$ $(1 \leq i \leq n)$. We write $Res(m_b, (c_1, c_2, \ldots, c_n)) = (c'_1, c'_2, \ldots, c'_n \rightarrow c')$, which is called *the resolution of* m_b *at* $(c_1, c_2, \ldots, c_n)$. We often write $Res(m_b, (c_1, c_2, \ldots, c_n)) = c'$ when $(c'_1, c'_2, \ldots, c'_n)$ is irrelevant. If such a unique base method definition does not exist, then the resolution of m_b at $(c_1, c_2, \ldots, c_n)$ is *undefined*, and we write $Res(m_b, (c_1, c_2, \ldots, c_n)) = \perp$.

Similarly, for $m_u \in M_{u,n}$ and $c_1, c_2, \ldots, c_n \in C$, if $(m_u, (c'_1, c'_2, \ldots, c'_n), t)$ is the user method definition of m_u at the "componentwise smallest" $(c'_1, c'_2, \ldots, c'_n)$ above $(c_1, c_2, \ldots, c_n)$, then we define *the resolution of* m_u *at* $(c_1, c_2, \ldots, c_n)$ as

$C = \{\mathsf{employee}, \mathsf{staff}, \mathsf{host}, \mathsf{server}, \mathsf{use}\}$

$\mathsf{staff} \leq \mathsf{employee},\ \mathsf{server} \leq \mathsf{host},$

$M = \{leader, hostname, service, boss, admin\}$

$\Sigma_b = \{(leader, (\mathsf{employee} \rightarrow \mathsf{employee})),$
$(leader, (\mathsf{staff} \rightarrow \mathsf{staff})),$
$(hostname, (\mathsf{employee} \rightarrow \mathsf{host})),$
$(service, (\mathsf{host} \rightarrow \mathsf{use}))\}$

$\Sigma_u = \{(boss, (\mathsf{employee}), boss(leader(x))),$
$(boss, (\mathsf{staff}), leader(x)),$
$(admin, (\mathsf{employee}), service(hostname(x)))\}$

Figure 3.1 A method schema S_2.

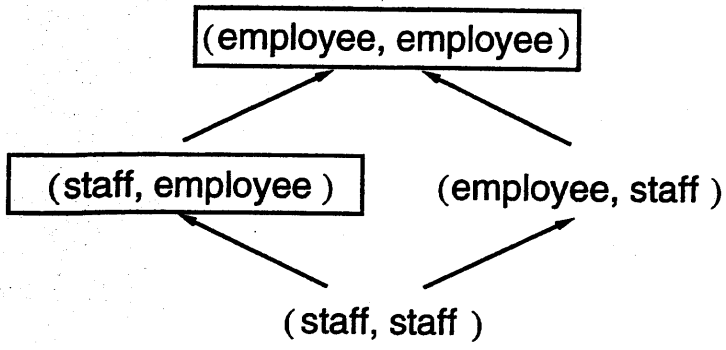


Figure 3.2 An example of method inheritances.

$Res(m_u, (c_1, c_2, \ldots, c_n)) = ((c'_1, c'_2, \ldots, c'_n), t)$, or simply, t. If such $(m_u, (c'_1, c'_2, \ldots, c'_n), t)$ does not exist, then $Res(m_u, (c_1, c_2, \ldots, c_n)) = \perp$. □

In Figure 3.2, $Res(m, (\mathsf{staff}, \mathsf{staff}))$ is the method definition of m at (staff, employee). However, if the method definition of m also exists at (employee, staff), then $Res(m, (\mathsf{staff}, \mathsf{staff})) = \perp$ since (staff, employee) and (employee, staff) are incomparable with respect to $\leq$.

3.2.3 Semantics

The semantics of a method schema, which is introduced by Abiteboul et al. [1, 2], is defined as follows. To each class name, a set of objects is assigned. Also, to each base method name m_b, a mapping over appropriate sets of objects is assigned as its interpretation. The semantics of a user method is defined by the interpretation of base methods and term rewriting [13]. For a set V, let V^m denote the Cartesian product $\underbrace{V \times V \times \cdots \times V}_{m}$.

Definition 3.5: An *interpretation*, also called a *database instance*, of a method schema $S = (C, \leq, M, \Sigma_b, \Sigma_u)$ is a pair $I = (\nu, \mu)$ defined as follows:

1. To each $c \in C$, ν assigns a finite disjoint set, denoted $\nu(c)$. Each $o \in \nu(c)$ is called an *object* of c. Let $O_{S,I} = \bigcup_{c \in C} \nu(c)$. We simply write O instead of $O_{S,I}$ if S and I are understood from the context.

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

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

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

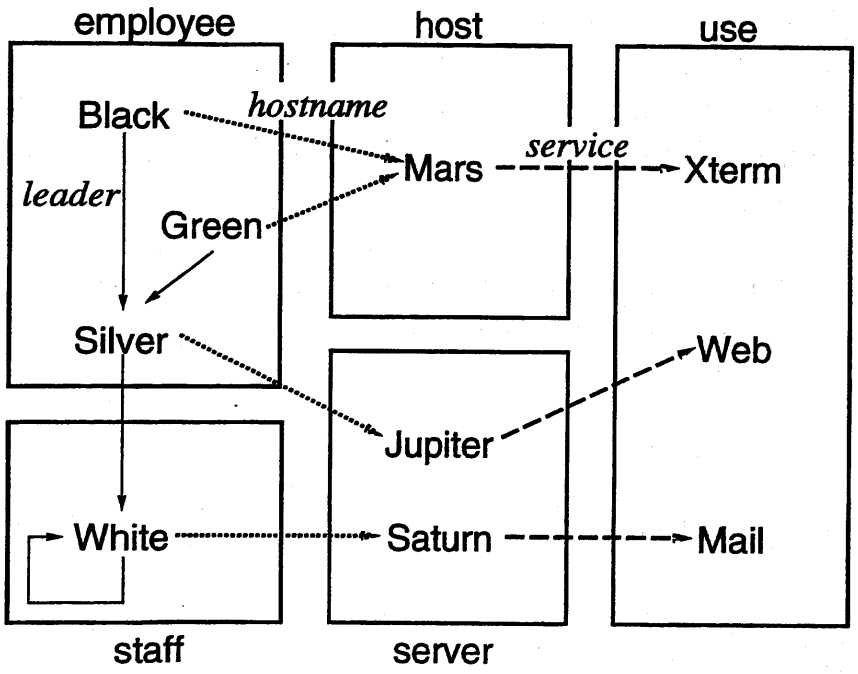


Figure 3.3 An interpretation I_2 of S_2.

Example 3.3: Figure 3.3 shows an example of an interpretation $I_2 = (\nu_2, \mu_2)$ of S_2 in Figure 3.1. For example, $\nu_2(\mathsf{employee}) = \{\mathsf{Black}, \mathsf{Green}, \mathsf{Silver}\}$, which represents that Black, Green, and Silver are objects of a class employee, and $\mu_2(\mathit{hostname})(\mathsf{Black}) = \mathsf{Mars}$, which represents that the application of a method *hostname* to an object Black is an object Mars. □

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

A term $t \in T_M(O_{S,I})$ is called an *instantiated term.* That is, an instantiated term consists of method names in M and objects in $O_{S,I}$. The *reduction relation* $\underset{S,I}{\rightarrow}$ on the instantiated terms, based on the leftmost innermost reduction strategy, is defined as follows.

Definition 3.6: For an instantiated term $t \in T_M(O_{S,I})$, let r be the leftmost innermost occurrence such that the subterm of t at r is $m(\vec{o})$ for some $m \in M$ and $\vec{o} \in \nu(\vec{c})$.

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

Note that, by Definition 3.6, for any instantiated term t, there exists at most one term t' such that $t \underset{S,I}{\rightarrow} t'$.

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

Example 3.4: For a method schema S_2 in Figure 3.1 and an interpretation I_2 in Figure 3.3, a method *boss*(Black) is executed by Definition 3.6 in the following way:

$$\begin{aligned} \mathit{boss}(\mathsf{Black}) &\underset{S_2,I_2}{\rightarrow} \mathit{boss}(\mathit{leader}(\mathsf{Black})) \\ &\underset{S_2,I_2}{\rightarrow} \mathit{boss}(\mathsf{Silver}) \\ &\underset{S_2,I_2}{\rightarrow} \mathit{boss}(\mathit{leader}(\mathsf{Silver})) \\ &\underset{S_2,I_2}{\rightarrow} \mathit{boss}(\mathsf{White}) \\ &\underset{S_2,I_2}{\rightarrow} \mathit{leader}(\mathsf{White}) \\ &\underset{S_2,I_2}{\rightarrow} \mathsf{White} \end{aligned}$$

Thus, $\mathit{boss}(\mathsf{Black})\downarrow = \mathsf{White}$. □

3.2.4 Successful Method Execution

In order to detect security flaws, this thesis is interested in the results of successful method executions. More precisely, for each $m \in M_n$ and $\vec{o} \in O^n_{S,I}$, we want to compute o such that $m(\vec{o}) \xrightarrow{*} o$ if exists. Since the term rewriting system induced by Definition 3.6 is ground, such o can be computed using a congruence closure algorithm [14, 24] in the following way.

Definition 3.7: An equivalence relation $\sim$ on $T_F(X)$ is called a *congruence relation* if it satisfies substitutivity, i.e., $t_i \sim t'_i$ $(1 \leq i \leq n)$ implies $f(t_1, t_2, \ldots, t_n) \sim f(t'_1, t'_2, \ldots, t'_n)$ for any $f \in F_n$. The *congruence closure* of a binary relation R on $T_F(X)$ is the least congruence relation containing R. □

Let Q be a subset of $T_M(O_{S,I})$ satisfying the following conditions:

1. Q contains $m(\vec{o})$ and t if $m \in M_n$, $\vec{o} \in O^n_{S,I}$, and $m(\vec{o}) \rightarrow t$.
2. Q is closed with respect to subterms, i.e., if Q contains t, then it does all the subterms of t.

Q is not required to be the *smallest* subset satisfying the above conditions. However, if we want to compute the results of successful method executions efficiently, then it is desirable that Q is as small as possible. Define $\underset{Q}{\rightarrow}$ as $\rightarrow$ restricted to

Q, and let $\underset{Q}{\sim}$ be the congruence closure of $\underset{Q}{\rightarrow}$. In what follows, this thesis shows that, for any $t \in Q$ and $o \in O_{S,I}$, $t \underset{Q}{\sim} o$ *iff* $t \xrightarrow{*} o$. Thus, it suffices to compute $\underset{Q}{\sim}$ in order to achieve our goal here.

Lemma 3.1: For any $t, t' \in Q$, if $t \xrightarrow{*} t'$, then $t \underset{Q}{\sim} t'$.

Proof: Reference [24] shows that the decision problem for the theory of equality can be reduced to the congruence closure problem. The lemma holds by this fact. □

Lemma 3.2: For any $t, t' \in Q$ and $o \in O_{S,I}$, if $t \underset{Q}{\sim} t'$, then $t \xrightarrow{*} o$ is equivalent to $t' \xrightarrow{*} o$.

Proof: We use induction on the length k of a proof of $t \underset{Q}{\sim} t'$.

Basis: If $k = 1$, then consider the following two cases:

1. If $t \underset{Q}{\rightarrow} t'$, then $t \rightarrow t'$ by the definition of $\underset{Q}{\rightarrow}$. Since there exists no t'' such that $t'' \neq t'$ and $t \rightarrow t''$ by Definition 3.6, the lemma holds.

2. If $t' = t$, then the lemma holds evidently.

Induction: Suppose that $t \underset{Q}{\sim} t'$ is shown by a k-step proof. Consider the following three cases:

1. If symmetry (i.e., if $t' \underset{Q}{\sim} t$, then $t \underset{Q}{\sim} t'$) is used at the k-th step of the proof, then $t' \xrightarrow{*} o$ *iff* $t \xrightarrow{*} o$ by the inductive hypothesis. Thus, the lemma holds.

2. If transitivity (i.e., if $t \underset{Q}{\sim} t''$ and $t'' \underset{Q}{\sim} t'$, then $t \underset{Q}{\sim} t'$) is used at the k-th step of the proof, then $t \xrightarrow{*} o$ is equivalent to $t'' \xrightarrow{*} o$, and $t'' \xrightarrow{*} o$ is equivalent to $t' \xrightarrow{*} o$ by the inductive hypothesis. Hence, $t \xrightarrow{*} o$ is equivalent to $t' \xrightarrow{*} o$, and the lemma holds.

3. If substitutivity (i.e., if $t_i \underset{Q}{\sim} t'_i$ $(1 \leq i \leq n)$, then $m(t_1, t_2, \ldots, t_n) \underset{Q}{\sim} m(t'_1, t'_2, \ldots, t'_n)$) is used at the k-th step of the proof, then $t_i \xrightarrow{*} o_i$ is equivalent to $t'_i \xrightarrow{*} o_i$ $(1 \leq i \leq n)$ by the inductive hypothesis. Let $t = m(t_1, t_2, \ldots, t_n)$ and $t' = m(t'_1, t'_2, \ldots, t'_n)$. Suppose that $t \xrightarrow{*} o$. Since $\xrightarrow{*}$ is an innermost reduction by Definition 3.6, there exists $m(o_1, o_2, \ldots, o_n)$

such that $t \xrightarrow{*} m(o_1, o_2, \ldots, o_n) \xrightarrow{*} o$, and therefore $t_i \xrightarrow{*} o_i$ $(1 \leq i \leq n)$. By this and the inductive hypothesis, it follows that $t' \xrightarrow{*} m(o_1, o_2, \ldots, o_n)$. Therefore, it has been proved that $t' \xrightarrow{*} o$ if $t \xrightarrow{*} o$. Conversely, it is provable in the same way that, if $t' \xrightarrow{*} o$, then $t \xrightarrow{*} o$. Consequently, the lemma holds.

By the above steps, the lemma has been proved. □

The following theorem holds by Lemmas 3.1 and 3.2.

Theorem 3.1: For any $t \in Q$ and $o \in O_{S,I}$, $t \underset{Q}{\sim} o$ *iff* $t \xrightarrow{*} o$. □

Example 3.5: For S_2 in Figure 3.1 and I_2 in Figure 3.3, the results of successful method executions are computed as follows. First, Q_2 is constructed from the definition. Contents of Q_2 are shown in Figure 3.4. Next, constructing $\underset{Q_2}{\sim}$ on Q_2 by a known algorithm, the congruence closure shown in Figure 3.4 can be obtained. For each term t_1 and t_2 in $Q_2^{(i)}$ $(1 \leq i \leq 10)$, it holds that $t_1 \underset{Q_2}{\sim} t_2$. For example, *boss*(Black) $\underset{Q_2}{\sim}$ White. Hence, *boss*(Black) $\xrightarrow{*}$ White from Theorem 3.1. See Example 3.4. □

The following summarizes the time complexity to compute the results of successful method executions. For each $m \in M_n$ and $\vec{o} \in O_{S,I}^n$, the following algorithm computes o such that $m(\vec{o}) \xrightarrow{*} o$ if exists:

Step 1: Construct Q and $\underset{Q}{\rightarrow}$.

Step 2: From $\underset{Q}{\rightarrow}$ in Step 1, compute the congruence closure $\underset{Q}{\sim}$ by a known algorithm [14].

Step 3: For each $t = m(\vec{o}) \in Q$, obtain $o \in O_{S,I}$ such that $t \underset{Q}{\sim} o$ (i.e., $t \xrightarrow{*} o$ by Theorem 3.1).

Let $\|t\|$ be the size of a term t as $|OC(t)|$, where $|X|$ denotes the number of elements of a set X, i.e., $\|t\|$ is the number of nodes in the tree representing t. Define the description length of Σ_u, denoted $\|\Sigma_u\|$, as the sum of $\|t\|$ for all $(m, \vec{c}, t) \in \Sigma_u$. Also, define the size of S, denoted $\|S\|$, as follows:

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

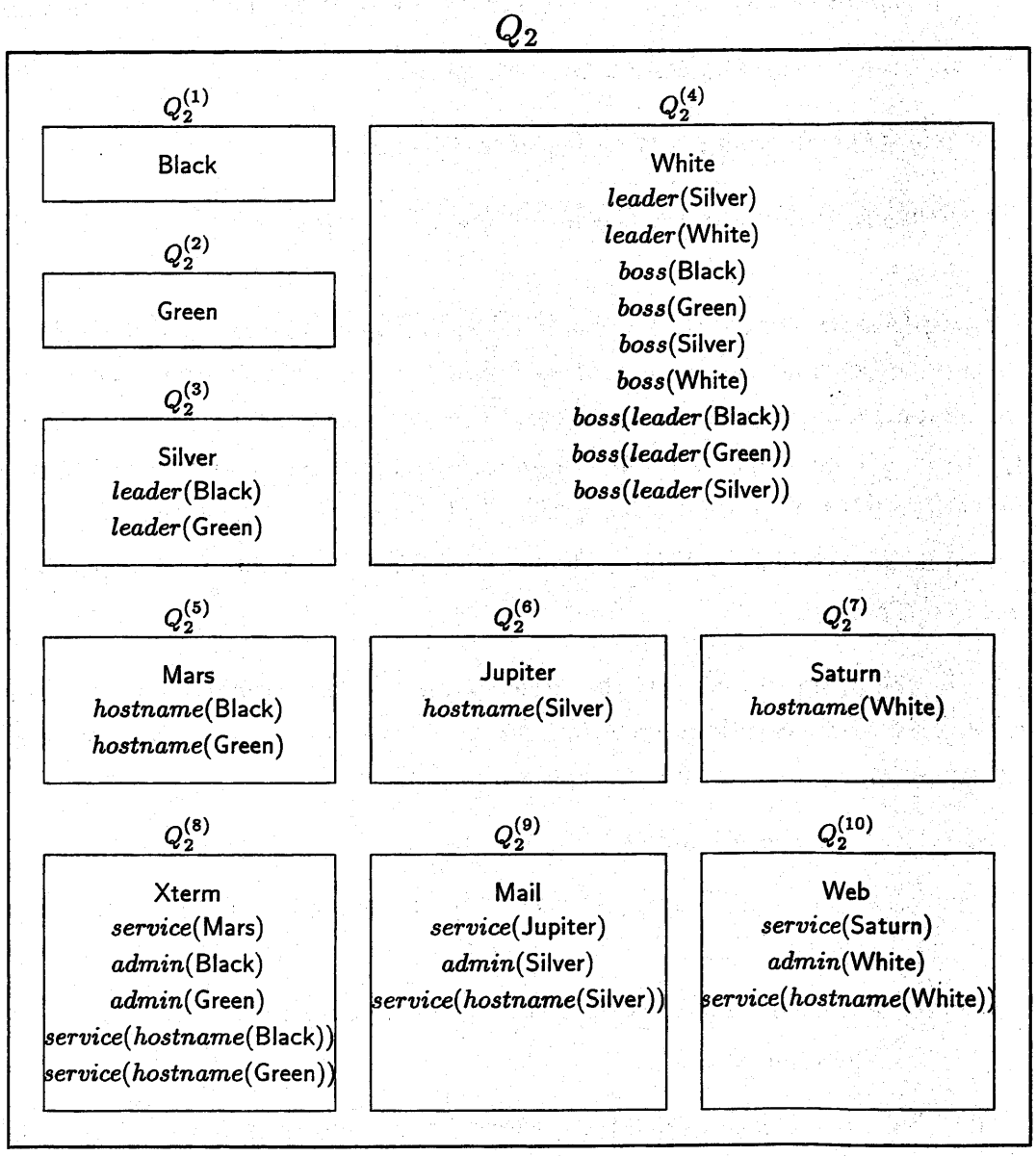


Figure 3.4 Contents of Q_2.

procedure COMPUTE-METHOD-EXECUTIONS
Input : a method schema S
Output: an array $Array_R$ storing the results of method executions
begin

1: $Q \leftarrow \phi$;
2: **foreach** $m(\vec{o})$ **in** $T_M(O)$
3: **if** there exists $t \in T_M(O)$ such that $m(\vec{o}) \rightarrow t$ **then**
4: $Q \leftarrow Q \cup \{m(\vec{o}), t\}$;
5: $Array_Q[m(\vec{o})] \leftarrow t$;
6: **foreach** t **in** Q
7: $Q \leftarrow Q \cup \{t' | t'$ is a subterm of $t\}$;
8: $cc \leftarrow$ CONSTRUCT-CONGRUENCE-CLOSURE$(Q, Array_Q)$;
9: **foreach** o **in** $Q \cap O$
10: label FIND-REPRESENTATIVE(cc, o) by o;
11: **foreach** $m(\vec{o})$ **in** Q
12: **if** FIND-REPRESENTATIVE$(cc, m(\vec{o}))$ is labeled by $o \in O$ **then**
13: $Array_R[m(\vec{o})] \leftarrow o$;

end

Figure 3.5 Procedure to compute method executions.

For simplicity, $O_{S,I}$ is denoted by O here. Furthermore, let k be the maximum number of arity of all methods. To achieve the algorithm, suppose that a partial mapping μ is given as a table and binary relations $\underset{Q}{\rightarrow}$ and $\xrightarrow{*}$ are retained as arrays $Array_Q$ and $Array_R$ respectively. For example, $Array_R[m(\vec{o})] = o$ if $m(\vec{o}) \xrightarrow{*} o \in O_{S,I}$. Also, suppose that the table and these arrays are accessed in constant time.

A procedure COMPUTE-METHOD-EXECUTIONS in Figure 3.5 achieves the above algorithm. Step 1 corresponds to lines (1)–(7) in Figure 3.5. The procedure shows that constructing Q and $\underset{Q}{\rightarrow}$ takes $\mathcal{O}(\|S\||Q|)$ time, where

$$|Q| \leq |M_b||O|^k + |M_u||O|^k + \|\Sigma_u\||O|^k = \mathcal{O}(\|S\||O|^k)$$

by the definition of Q. Step 2 corresponds to line (8) in Figure 3.5. A procedure CONSTRUCT-CONGRUENCE-CLOSURE($Q, Array_Q$) computes the congruence closure of $\underset{Q}{\rightarrow}$ and returns a disjoint-set forest cc [12] that represents the congruence closure. This can be done in $\mathcal{O}(|Q|(\log|Q|)^2)$ time [14]. Step 3 corresponds to lines (9)–(13) in Figure 3.5. A procedure FIND-REPRESENTATIVE(cc, t) is an operation to return a representative of the congruence class in cc to which a term t belongs. A representative of the congruence class to which an object o belongs is labeled by o. Then, an object o such that $m(\vec{o}) \xrightarrow{*} o$ can be immediately obtained by looking up the label of the representative of $m(\vec{o})$. It is easily verified that the total time for Step 3 is $\mathcal{O}(|Q|)$. After all, computing the results of successful method executions takes

$$\mathcal{O}(|Q|((\log|Q|)^2 + \|S\|)) \leq \mathcal{O}(\|S\||O|^k((\log\|S\||O|^k)^2 + \|S\|))$$

time.

3.3 Security Analysis

3.3.1 Authorization

A general authorization model has been provided in Chapter 2. In order to investigate detection of a security flaw, this chapter focuses on what a user u is permitted to invoke and what u is prohibited from invoking. In this chapter, the following authorization which simplifies one provided in Chapter 2 concentrates on a method invocation is adopted.

Definition 3.8: Let $S = (C, \leq, M, \Sigma_b, \Sigma_u)$ be a method schema. An *authorization* A for a user u under S is a finite set of $(m, \vec{c})$, where $m \in M_n$ and $\vec{c} \in C^n$. Intuitively, $(m, \vec{c}) \in A$ means that u is permitted to invoke a method m on any tuple $\vec{o}$ of objects such that $\vec{o} \in \nu(\vec{c})$. We simply write (m, c) instead of $(m, (c))$ for unary methods. □

As stated in Chapter 2, an authorization is generally modeled by a base authorization (e.g., rules r_1 and r_2 in Example 2.4) and a set of inference rules (e.g., rules r_3 through r_5 in Example 2.4). For example, consider such an inference rule that "if user u is permitted to invoke m on objects of c, then u is also permitted to invoke m on objects of the subclasses of c." By this rule, a base authorization $\{(m, c)\}$ is expanded into $\{(m, c), (m, c_1), (m, c_2)\}$ if $c_1 \leq c$ and $c_2 \leq c$. This chapter assumes that a given authorization has already been expanded.

Example 3.6: Let A_2 be an authorization for a user u under S_2 in Figure 3.1 as follows:

$$\begin{aligned} A_2 = \{&(\mathit{boss}, \mathsf{employee}),\\ &(\mathit{boss}, \mathsf{staff}),\\ &(\mathit{hostname}, \mathsf{employee}),\\ &(\mathit{admin}, \mathsf{employee}),\\ &(\mathit{admin}, \mathsf{staff})\}. \end{aligned}$$

Consider the interpretation I_2 in Figure 3.3. Executing $\mathit{hostname}(\mathsf{Black})$ by u is permitted since $\mathsf{Black} \in \nu_2(\mathsf{employee})$ and $(\mathit{hostname}, \mathsf{employee}) \in A_2$. On the other hand, executing $\mathit{leader}(\mathsf{Silver})$ by u is prohibited since $\mathsf{Silver} \in \nu_2(\mathsf{employee})$ but $(\mathit{leader}, \mathsf{employee}) \notin A_2$. □

3.3.2 Formal Definition of User's Inference

Let $S = (C, \leq, M, \Sigma_b, \Sigma_u)$ be a method schema, $I = (\nu, \mu)$ an interpretation of S, A the authorization for a user u, and $O_{S,I} = \bigcup_{c \in C} \nu(c)$. For a given term $\tau \in T_M(O_{S,I})$, we are interested in deciding whether or not u can infer the object $o \in O_{S,I}$ such that $\tau \downarrow = o$.

This section defines the information which u can obtain. First, consider the information on the existence of objects. Let $\tilde{O}$ denote the set of objects such that

$o \in \tilde{O}$ *iff* u knows that o exists in $O_{S,I}$. Formally, $\tilde{O}$ is defined as the smallest set satisfying the following (*1) and (*2):

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

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

By (*1), we assume that u has partial knowledge on I in advance. (*2) is derived from Policy (1) in Section 3.1, i.e., u can obtain the result of $m(\vec{o})$. Secondly, consider the information on classes.

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

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

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

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

(*6) User u knows that $Res(m, \vec{c}) = t$ holds if $(m, \vec{c}) \in A$ and $Res(m, \vec{c}) = t$ actually holds.

(*5) is derived from Policy (1) in Section 3.1. (*6) corresponds to Policy (2) in Section 3.1, i.e., u can obtain the type declaration of m at $\vec{c}$ (if m is a base method) and the external specification of m at $\vec{c}$ (if m is a user method).

In our formalization, all data are objects. This thesis assumes that users do not know whether or not an object actually represents a basic value such as integer and string. This means that users cannot use any domain-dependent properties as exhibited in the next example. Therefore, this thesis assumes that users have no a priori knowledge on base methods.

Example 3.7: Suppose that basic values are implemented as objects and users have a priori knowledge on base methods. Let $factorial(x)$ be a base method to compute the factorial of x. Assume that a user knows $factorial(x) = 6$ for an object x. Then, the user can infer $x = 3$ since the user knows both the behavior of $factorial$ and the meaning of object 6 (if the user is permitted to invoke $factorial$). On the other hand, without a priori knowledge on base methods, the user cannot infer $x = 3$ only from the information that $factorial(x) = 6$. □

What information can the user infer from the above information (*1)–(*6)? Suppose that the user can use at least four kinds of inference rules: reflexivity, symmetry, transitivity, and substitutivity. These inference rules yield equalities from equalities, and their contrapositions yield inequalities from inequalities.

Example 3.8: Consider the following schema:

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

Also, consider the interpretation (ν, μ) shown in Figure 3.6. In the figure, the arrow labeled by m from o_1 to o_2 shows that $\mu(m)(o_1) = o_2$. Figure 3.7 shows the results of all the method executions. Suppose that $\tilde{O} = \{o_1, o_2, o_3\}$.

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

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

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

$$\nu(c) = \{o_1, o_2, o_3\}$$

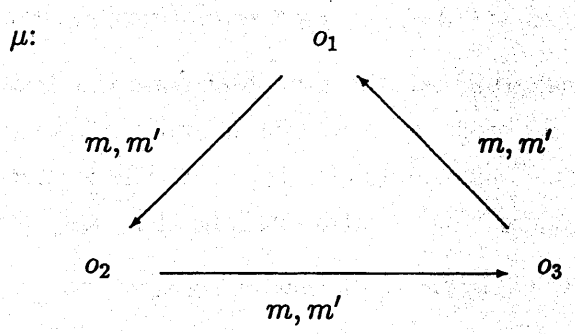


Figure 3.6 An example of an interpretation.

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

Figure 3.7 An example of method executions.

Hereafter, suppose that following two conditions hold:

(a) The user does not know what $O_{S,I}$ is.

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

In many cases, these conditions are satisfied by just hiding $O_{S,I}$ and C from the user. In what follows, how these conditions affect user's inference is examined. First, consider Condition (a).

Example 3.9: Consider the same schema and interpretation of Example 3.8 and the following authorization for a user u :

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

As stated in the latter part of Example 3.8, u can infer $m(o_1)\downarrow \neq o_1$ and $m(o_1)\downarrow \neq o_3$. If u knows that $O_{S,I} = \{o_1, o_2, o_3\}$, then u can obtain $m(o_1)\downarrow = o_2$.

On the other hand, if u does not know what $O_{S,I}$ is, then u cannot infer $m(o_1)\downarrow = o_2$ since u cannot neglect the case that there exists another object $o \neq o_2$ such that $m(o_1)\downarrow = o$. □

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

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

Example 3.10: Consider the following schema:

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

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

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

and an interpretation such that

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

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

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

Thus, what the user can infer can be modeled as a congruence closure of a finite set of ground equalities induced by (*5) and (*6). Consequently, we can use an algorithm to compute a congruence closure. The following definition introduces a congruence relation $\underset{A,I}{\approx}$ on $T_M(O_{S,I})$. Intuitively, $t \underset{A,I}{\approx} t'$ means that user u can infer that t and t' have the same "value" under A and I, i.e., $t\downarrow = t'\downarrow$.

Definition 3.9: Define $\underset{A,I}{\approx}$ as the least congruence relation on $T_M(O_{S,I})$ satisfying the following two conditions:

(1) If $(m, \vec{c}) \in A$, $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, and $m(\vec{o}) \xrightarrow{*} o' \in O_{S,I}$, then $m(\vec{o}) \underset{A,I}{\approx} o'$ (see (*5)).

(2) If $(m, \vec{c}) \in A$, $m \in M_u$, $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, and $Res(m, \vec{c}) = t \in T_M(\{\vec{x}\})$, then $m(\vec{o}) \underset{A,I}{\approx} t[\vec{o}/\vec{x}]$ (see (*6)). □

We have another characterization of user's inference.

Definition 3.10: Define P_I as the minimum set of rewriting rules $\underset{A,I}{\triangleright}$ on $T_M(O_{S,I})$ satisfying the following three conditions. Intuitively, $t \underset{A,I}{\triangleright} o$ means that user u can infer that the result of successful execution of t is o under A and I, i.e., $t\downarrow = o$.

(1) If $(m, \vec{c}) \in A$, $m \in M_n$, $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, and $m(\vec{o})\downarrow = o \in O_{S,I}$, then P_I contains

$$m(\vec{o}) \underset{A,I}{\triangleright} o.$$

This rules corresponds to (*5).

(2) If $(m_u, \vec{c}) \in A$, $m_u \in M_{u,n}$, $\vec{o} \in \tilde{O}^n$, $\vec{o} \in \nu(\vec{c})$, $m_u(\vec{o})\downarrow = o \in O_{S,I}$, and $Res(m_u, \vec{c}) = t \in T_M(\{\vec{x}\})$, then P_I contains

$$t[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o.$$

This rule corresponds to (*6).

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

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

This rule simulates Knuth-Bendix completion procedure [26].

Note that the existence of $t \underset{A,I}{\triangleright} o \in P_I$ implies $t \xrightarrow{*} o$.

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

Although $\overset{*}{\underset{A,I}{\Rightarrow}}$ is not identical to $\underset{A,I}{\approx}$, we have $t \overset{*}{\underset{A,I}{\Rightarrow}} o$ *iff* $t \underset{A,I}{\approx} o$ for any $o \in O_{S,I}$ because of the correctness of Knuth-Bendix completion [26].

We use $\underset{A,I}{\approx}$ in Section 3.4 since we want to use fast known algorithms for computing congruence closures. On the other hand, we use $\overset{*}{\underset{A,I}{\Rightarrow}}$ in Section 3.5 since rewriting rules $\underset{A,I}{\triangleright}$ on $T_M(O_{S,I})$ can be naturally approximated by rewriting rules on $T_M(C)$.

Example 3.11: For S_2 in Figure 3.1, I_2 in Figure 3.3, and A_2 in Example 3.6, let $\tilde{O}_0 = \{\mathsf{Black}, \mathsf{Green}, \mathsf{Silver}, \mathsf{White}\}$. Then, $\tilde{O} = O_{S_2,I_2}$. The congruence closure $\underset{A_2,I_2}{\approx}$ on a subset Q_{2_A} of $T_M(O_{S_2,I_2})$ is shown in Figure 3.8 (compare with Q_2 in Example 3.5). For each term t_1 and t_2 in $Q_{2_A}^{(i)}$ $(1 \leq i \leq 6)$, it holds that $t_1 \underset{A_2,I_2}{\approx} t_2$. For example,

$$admin(\mathsf{Black}) \underset{A_2,I_2}{\approx} \mathsf{Xterm}$$

since $admin(\mathsf{Black})$ and Xterm are in the same congruence class. This means that user u knows that executing $admin(\mathsf{Black})$ is permitted under A_2 and its result is Xterm.

On the other hand, let us compute $\underset{A_2,I_2}{\approx}$ using $\underset{A_2,I_2}{\Rightarrow}$ in Definition 3.10. Figure 3.9 shows P_{I_2}. Rules (1)–(11) and (12)–(19) in Figure 3.9 are obtained by Definitions 3.10(1) and (2) respectively. Also, rule (20) is obtained using rules (5) and (16) (or (6) and (17)) by Definition 3.10(3) and rule (21) obtained using (7) and (18). For example, the following relations are obtained by using these rules:

$$boss(\mathsf{Black}) \underset{A_2,I_2}{\Rightarrow} \mathsf{White},$$
$$admin(\mathsf{White}) \underset{A_2,I_2}{\Rightarrow} \mathsf{Web}.$$

Then,

$$admin(boss(\mathsf{Black})) \underset{A_2,I_2}{\approx} \mathsf{Web}$$

since $admin(boss(\mathsf{Black})) \underset{A_2,I_2}{\overset{*}{\Rightarrow}} \mathsf{Web}$. □

3.4 The Detection Problem of Security Flaws for Database Instance

3.4.1 The Problem

Definition 3.11: Let S be a method schema, I an interpretation, A an authorization, and τ a term in $T_M(O_{S,I})$ to be verified. The *detection problem of security flaws for database instance*, or simply the *detection problem for instances*, is to decide whether or not, for given S, I, A, and τ, there exists an object $o \in O_{S,I}$ such that $\tau \underset{A,I}{\approx} o$. □

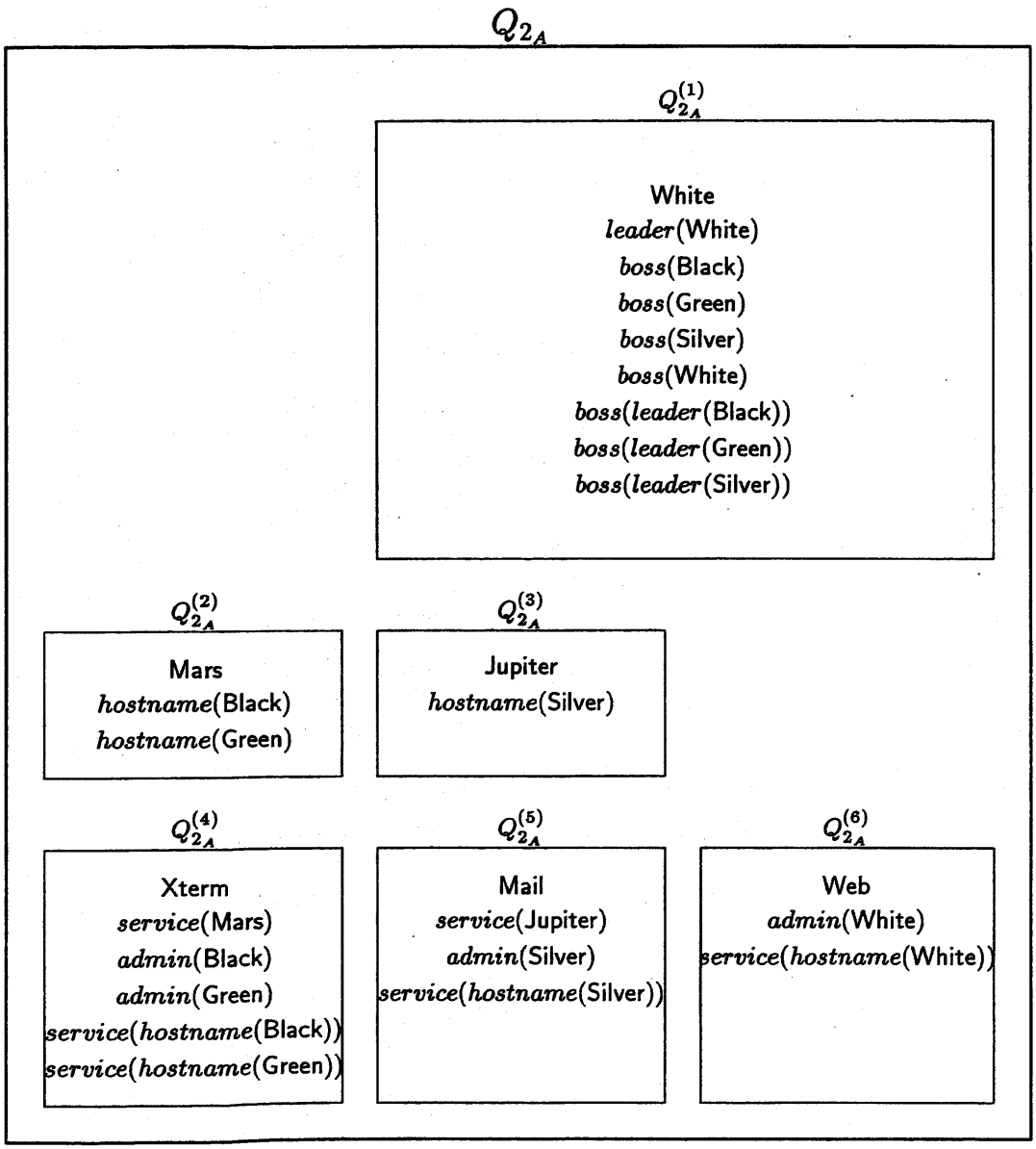


Figure 3.8 Contents of Q_{2_A}.

(1) $boss(\text{Black}) \underset{A_2,I_2}{\triangleright} \text{White}$

(2) $boss(\text{Green}) \underset{A_2,I_2}{\triangleright} \text{White}$

(3) $boss(\text{Silver}) \underset{A_2,I_2}{\triangleright} \text{White}$

(4) $boss(\text{White}) \underset{A_2,I_2}{\triangleright} \text{White}$

(5) $hostname(\text{Black}) \underset{A_2,I_2}{\triangleright} \text{Mars}$

(6) $hostname(\text{Green}) \underset{A_2,I_2}{\triangleright} \text{Mars}$

(7) $hostname(\text{Silver}) \underset{A_2,I_2}{\triangleright} \text{Jupiter}$

(8) $admin(\text{Black}) \underset{A_2,I_2}{\triangleright} \text{Xterm}$

(9) $admin(\text{Green}) \underset{A_2,I_2}{\triangleright} \text{Xterm}$

(10) $admin(\text{Silver}) \underset{A_2,I_2}{\triangleright} \text{Mail}$

(11) $admin(\text{White}) \underset{A_2,I_2}{\triangleright} \text{Web}$

(12) $boss(leader(\text{Black})) \underset{A_2,I_2}{\triangleright} \text{White}$

(13) $boss(leader(\text{Green})) \underset{A_2,I_2}{\triangleright} \text{White}$

(14) $boss(leader(\text{Silver})) \underset{A_2,I_2}{\triangleright} \text{White}$

(15) $leader(\text{White}) \underset{A_2,I_2}{\triangleright} \text{White}$

(16) $service(hostname(\text{Black})) \underset{A_2,I_2}{\triangleright} \text{Xterm}$

(17) $service(hostname(\text{Green})) \underset{A_2,I_2}{\triangleright} \text{Xterm}$

(18) $service(hostname(\text{Silver})) \underset{A_2,I_2}{\triangleright} \text{Mail}$

(19) $service(hostname(\text{White})) \underset{A_2,I_2}{\triangleright} \text{Web}$

(20) $service(\text{Mars}) \underset{A_2,I_2}{\triangleright} \text{Xterm}$

(21) $service(\text{Jupiter}) \underset{A_2,I_2}{\triangleright} \text{Mail}$

Figure 3.9 Contents of P_{I_2}.

Let Q be a subset of $T_M(O_{S,I})$ defined in Section 3.2.4 and

$$Q' = Q \cup \{\tau' | \tau' \text{ is a subterm of } \tau\}.$$

In order to compute $\underset{A,I}{\approx}$, $\tilde{O}$ defined in Section 3.3.2 is computed in advance. From the above discussion, the detection problem for instances is solvable by computing $\underset{A,I}{\approx}$ on Q'. Since the results of successful method executions are computed in the way described in Section 3.2.4, for each $m(\vec{o}) \in Q$, we can obtain $o \in O_{S,I}$ such that $m(\vec{o}) \xrightarrow{*} o$ before computing $\underset{A,I}{\approx}$ on Q'.

Example 3.12: For S_2 in Figure 3.1, I_2 in Figure 3.3, A_2 in Example 3.6, and $\tilde{O}$ in Example 3.11, let $\tau_1 = admin(boss(\mathsf{Black}))$ be a term to be verified. From the definition,

$$Q'_2 = Q_2 \cup \{admin(boss(\mathsf{Black}))\}.$$

By Definition 3.9, for example, the following relations are obtained:

$$boss(\mathsf{Black}) \underset{A_2,I_2}{\approx} \mathsf{White},$$
$$admin(\mathsf{White}) \underset{A_2,I_2}{\approx} \mathsf{Web}.$$

Computing the congruence closure by a known algorithm, the following can be obtained:

$$\tau_1 \underset{A_2,I_2}{\approx} \mathsf{Web},$$

since τ_1, $admin(\mathsf{White})$, and Web are in the same congruence class. Thus, u can infer that $\tau_1 \downarrow = \mathsf{Web}$, that is, a security flaw on τ_1 occurs.

Next, let $\tau_2 = service(\mathsf{Jupiter})$ and $\tau_3 = service(\mathsf{Saturn})$. Then, $Q'_2 = Q_2$, and $\underset{A_2,I_2}{\approx}$ on Q'_2 can be obtained in the same way. Then,

$$\tau_2 \underset{A_2,I_2}{\approx} \mathsf{Mail}.$$

Thus, u can infer that $\tau_2 \downarrow = \mathsf{Mail}$ although u is prohibited from executing τ_2 under A_2. On the other hand, there exists no object o such that $\tau_3 \underset{A_2,I_2}{\approx} o$. Hence, u cannot infer the value of $\tau_3 \downarrow$, that is, no security flaw on τ_3 occurs. □

3.4.2 The Algorithm and its Complexity

Before providing the algorithm to detect a security flaw on τ, the results of successful method executions and $\tilde{O}$ have to be computed. The former results can be obtained in

$$\mathcal{O}(\|S\||O|^k((\log \|S\||O|^k)^2 + \|S\|))$$

time by the algorithm in Section 3.2.4. On the other hand, $\tilde{O}$ can be obtained by a procedure COLLECT-OBJECTS shown in Figure 3.10. Then, computing $\tilde{O}$ is

$$\mathcal{O}(\|S\||O|^k(\log \|S\||O|^k + \|S\|))$$

time. An explanation of the procedure is omitted.

The algorithm for detecting a security flaw on τ consists of the following four steps:

Step 1: Construct Q' from Q and τ.

Step 2: Compute all the pairs of terms in Q' satisfying condition (1) or (2) in Definition 3.9.

Step 3: Compute the congruence closure $\underset{A,I}{\approx}$ on Q' by a known algorithm.

Step 4: Output "A security flaw occurs" if $\tau \underset{A,I}{\approx} o$ for some $o \in O_{S,I}$, and output "No security flaw occurs" otherwise.

Let l be the size of a given term τ to be verified. Suppose that pairs of terms satisfying Definition 3.9(1) are retained as an array $Array_{A_1}$ and pairs of terms satisfying Definition 3.9(2) are retained as an array $Array_{A_2}$. Suppose that these arrays are accessed in constant time.

The above algorithm is achieved by a procedure DETECT-A-SECURITY-FLAW in Figure 3.11. Step 1 corresponds to line (1), which is computed in $\mathcal{O}(l)$. Note that

$$|Q'| = \mathcal{O}(|Q| + l) = \mathcal{O}(\|S\||O|^k + l).$$

Step 2 corresponds to lines (2)–(5). Lines (2)–(5) are derived from Definition 3.9. Retrieving an element from Res takes $t_{Res} = \mathcal{O}(k \log \|S\|)$ time if Res is implemented as a 2-3 tree. Then, the time complexity of Step 2 is

procedure COLLECT-OBJECTS

Input : a method schema S, an interpretation I, an authorization A,
a finite set $\tilde{O}_0$ of objects which a user knows in advance

Output: a total set $\tilde{O}$ of objects which a user can obtain

begin

1: $\tilde{O} \leftarrow \tilde{O}_0$;
2: *checked* $\leftarrow \phi$;
3: **repeat**
4: **foreach** $m, \vec{o}$ **in** $M, \tilde{O}^n$
5: **if** $m(\vec{o}) \notin$ *checked*, $\vec{o} \in \nu(\vec{c})$, and $(m, \vec{c}) \in A$ **then**
6: $\tilde{O} \leftarrow \tilde{O} \cup \{m(\vec{o})\downarrow\}$;
7: *checked* $\leftarrow$ *checked* $\cup \{m(\vec{o})\}$;
8: **until** $\tilde{O}$ does not change

end

Figure 3.10 Procedure to compute $\tilde{O}$.

procedure DETECT-A-SECURITY-FLAW

Input : a method schema S, an interpretation I, an authorization A,
a term τ in $T_M(O)$ to be verified, a finite set Q,
and an array $Array_Q$

Output: "A security flaw occurs" or "No security flaw occurs"

begin

1: $Q' \leftarrow Q \cup \{\tau' | \tau'$ is a subterm of $\tau\}$;
2: **foreach** $(m, \vec{c})$ **in** A
3: $Array_{A_1}[m(\vec{o})] \leftarrow Array_R[m(\vec{o})]$;
4: **if** $m \in M_u$ **then**
5: $Array_{A_2}[m(\vec{o})] \leftarrow Res(m, \vec{c})[\vec{o}/\vec{c}]$;
6: $cc' \leftarrow$ CONSTRUCT-CONGRUENCE-CLOSURE$(Q', Array_{A_1} \cup Array_{A_2})$;
7: **foreach** o **in** $Q' \cap O$
8: label FIND-REPRESENTATIVE(cc', o) by o;
9: **if** FIND-REPRESENTATIVE(cc', τ) is labeled by $o \in O$ **then**
10: **output** "A security flaw occurs";
11: **else**
12: **output** "No security flaw occurs";

end

Figure 3.11 Procedure to detect a security flaw.

$\mathcal{O}(|C||O|^k t_{Res}) = \mathcal{O}(k|C||O|^k \log \|S\|)$. To construct the congruence closure $\underset{A,I}{\approx}$ on Q' in Step 3, a procedure DETECT-A-SECURITY-FLAW($Q', Array_{A_1} \cup Array_{A_2}$) at line (6) is called. It takes $\mathcal{O}(|Q'|(\log |Q'|)^2)$ time to do this. Step 4 corresponds to lines (7)–(12), which takes $\mathcal{O}(|Q'|)$ time. Refer to the algorithm described in Section 3.2.4. Thus, the time complexity of the algorithm is

$$\begin{aligned}&\mathcal{O}(l + k|C||O|^k \log \|S\| + |Q'|(\log |Q'|)^2 + |Q'|) \\ &= \mathcal{O}((\|S\||O|^k + l)((\log(\|S\||O|^k + l))^2 + k \log \|S\|)).\end{aligned}$$

Theorem 3.2: The detection problem for instances can be solved in

$$\mathcal{O}((\|S\||O|^k + l)((\log(\|S\||O|^k + l))^2 + k \log \|S\|))$$

time, where k is the maximum number of arity of all methods, and l is the size of a given term to be verified. □

Suppose that μ is implemented as a table, that is, the results of base methods are stored in a table statically. This means that, when a user invokes a base method, the user can obtain the result of the method by only retrieving the table representing μ. Let $|\mu|$ be the size of the table representing μ. In such a case, we can assume that $|O|^k = \mathcal{O}(|\mu|)$.

Corollary 3.1: If $|O|^k = \mathcal{O}(|\mu|)$, then the detection problem for instances is solvable in polynomial time. □

3.5 The Detection Problem of Security Flaws for Database Schema

3.5.1 The Problem

Definition 3.12: Let S be a method schema, A an authorization, and τ a term in $T_M(C)$ to be verified. The *detection problem of security flaws for OODB schemas*, or simply the *detection problem for schemas*, is to decide whether or not, for given S, A, and τ, there exist an interpretation $I = (\nu, \mu)$ and $\tilde{O}$ such that $\tau[\vec{o}/\vec{c}] \underset{A,I}{\overset{*}{\Rightarrow}} o$ for some $\vec{o} \in \nu(\vec{c})$ and $o \in O_{S,I}$. □

This problem is undecidable for general method schemas. However, it is decidable in polynomial time for monadic method schemas as will be stated in Section 3.5.3.

Theorem 3.3: The detection problem for schema with methods of arity two is undecidable.

Sketch of Proof: Reference [22] shows that the type-consistency problem for method schemas with methods of arity two is undecidable by reducing the Post's Correspondence Problem (PCP) to the problem. In the reduction, each interpretation I is regarded as a candidate for a solution to a PCP. If I is actually a solution, then execution of a term, say $m(o)$, is aborted under I. Otherwise, $m(o)$ is nonterminating. By slightly modifying the reduction in Reference [22], we can construct a schema as follows:

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

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

3.5.2 A Sufficient Condition

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

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

Intuitively, $Z(t)$ contains all the classes c such that the result of successful execution of $t[\vec{o}/\vec{c}]$ is an object o for some $\vec{o} \in \nu(\vec{c})$ and $o \in \nu(c)$. The smaller $Z(t)$ is, the better approximation we have.

$$
\begin{aligned}
Z(\mathit{leader}(\mathsf{employee})) &= \{\mathsf{employee}, \mathsf{staff}\} \\
Z(\mathit{leader}(\mathsf{staff})) &= \{\mathsf{staff}\} \\
Z(\mathit{hostname}(\mathsf{employee})) &= \{\mathsf{host}, \mathsf{server}\} \\
Z(\mathit{hostname}(\mathsf{staff})) &= \{\mathsf{host}, \mathsf{server}\} \\
Z(\mathit{service}(\mathsf{host})) &= \{\mathsf{use}\} \\
Z(\mathit{service}(\mathsf{server})) &= \{\mathsf{use}\} \\
Z(\mathit{boss}(\mathsf{employee})) &= \{\mathsf{staff}\} \\
Z(\mathit{boss}(\mathsf{staff})) &= \{\mathsf{staff}\} \\
Z(\mathit{admin}(\mathsf{employee})) &= \{\mathsf{use}\} \\
Z(\mathit{admin}(\mathsf{staff})) &= \{\mathsf{use}\} \\
Z(m(c)) &= \phi \text{ for any other combinations of } m \text{ and } c \\
Z(m(t)) &= \bigcup_{c \in Z(c)} Z(m(c))
\end{aligned}
$$

Figure 3.12 An example of Z for S_2.

Example 3.13: For S_2 in Figure 3.1, Z can be computed using the algorithm provided in Reference [30]. Figure 3.12 shows the result of the computation. For example, $Z(leader(\mathsf{employee})) = \{\mathsf{employee}, \mathsf{staff}\}$ means that executing $leader(o)$ for an object o of employee results in an object of either employee or staff. □

The following defines rewriting rules $\underset{A,S}{\triangleright}$ on $T_M(C)$ which approximate $\underset{A,I}{\triangleright}$.

Definition 3.13: Define P_S as the minimum set of rewriting rules $\underset{A,S}{\triangleright}$ on $T_M(C)$ satisfying the following three conditions:

(1) If $(m, \vec{c}) \in A$, then P_S contains

$$m(\vec{c}) \underset{A,S}{\triangleright} c$$

for each $c \in Z(m(\vec{c}))$. This rule approximates Definition 3.10(1).

(2) If $(m_u, \vec{c}) \in A$, $m_u \in M_{u,n}$, and $Res(m_u, \vec{c}) = t \in T_M(\{\vec{x}\})$, then P_S contains

$$t[\vec{c}/\vec{x}] \underset{A,S}{\triangleright} c$$

for each $c \in Z(t[\vec{c}/\vec{x}])$. This rule approximates Definition 3.10(2).

(3) If P_S contains $t \underset{A,S}{\triangleright} c$ and $t'' \underset{A,S}{\triangleright} c''$ such that $t, t'' \in T_M(C)$, $c, c'' \in C$, and t'' is a proper subterm of t at r'', then P_S contains

$$t[r'' \leftarrow c''] \underset{A,S}{\triangleright} c'$$

for each $c' \in Z(t[r'' \leftarrow c''])$. This rule approximates Definition 3.10(3).

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

Example 3.14: For S_2 in Figure 3.1, A_2 in Example 3.6, and Z in Figure 3.12, P_{S_2} shown in Figure 3.13 can be constructed. Rules (1)–(6) and (7)–(10) are obtained by Definitions 3.13(1) and (2) respectively. Rules (11) and (12) are obtained by Definition 3.13(3). □

(1) $boss(\text{employee}) \underset{A_2,S_2}{\rhd} \text{staff}$

(2) $boss(\text{staff}) \underset{A_2,S_2}{\rhd} \text{staff}$

(3) $hostname(\text{employee}) \underset{A_2,S_2}{\rhd} \text{staff}$

(4) $hostname(\text{employee}) \underset{A_2,S_2}{\rhd} \text{server}$

(5) $admin(\text{employee}) \underset{A_2,S_2}{\rhd} \text{use}$

(6) $admin(\text{White}) \underset{A_2,S_2}{\rhd} \text{Web}$

(7) $boss(leader(\text{employee})) \underset{A_2,S_2}{\rhd} \text{staff}$

(8) $leader(\text{staff}) \underset{A_2,S_2}{\rhd} \text{staff}$

(9) $service(hostname(\text{employee})) \underset{A_2,S_2}{\rhd} \text{use}$

(10) $service(hostname(\text{staff})) \underset{A_2,S_2}{\rhd} \text{use}$

(11) $service(\text{host}) \underset{A_2,S_2}{\rhd} \text{use}$

(12) $service(\text{server}) \underset{A_2,S_2}{\rhd} \text{use}$

Figure 3.13 Contents of P_{S_2}.

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

Lemma 3.3: If there exists an interpretation $I = (\nu, \mu)$ such that $t[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o \in P_I$ for some $\vec{o} \in \nu(\vec{c})$ and $o \in \nu(c)$, then $t[\vec{c}/\vec{x}] \underset{A,S}{\triangleright} c \in P_S$.

Proof: We use induction on the number of the repetition of a procedure which computes the least fixed point satisfying the three conditions in Definition 3.10.

Basis: Consider the following two cases:

1. The case that $m(\vec{o}) \underset{A,I}{\triangleright} o$ is obtained from Definition 3.10 (1): It holds that $(m, \vec{c}) \in A$, $\vec{o} \in \nu(\vec{c})$, and $m(\vec{o}) \xrightarrow{*} o$. From the property of Z, there exists a class c such that $c \in Z(m(\vec{c}))$ and $o \in \nu(c)$. From Definition 3.13 (1), P_S contains $m(\vec{c}) \underset{A,S}{\triangleright} c$ since $(m, \vec{c}) \in A$ and $c \in Z(m(\vec{c}))$.

2. The case that $Res(m_u, \vec{c})[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o$ is obtained from Definition 3.10 (2): In the same way as the above 1, it holds that $(m_u, \vec{c}) \in A$, $Res(m_u, \vec{c}) = t$, and $c \in Z(t[\vec{o}/\vec{x}])$. From Definition 3.13 (2), P_S contains $t[\vec{c}/\vec{x}] \underset{A,S}{\triangleright} c$.

Induction: Suppose that $t''[\vec{o''}/\vec{x''}]$ is a proper subterm of $t[\vec{o}/\vec{x}]$ at r'' and that $t[\vec{o}/\vec{x}] \underset{A,I}{\triangleright} o$ and $t''[\vec{o''}/\vec{x''}] \underset{A,I}{\triangleright} o''$ have been obtained. Let $t'[\vec{o'}/\vec{x'}] = t[\vec{o}/\vec{x}][r'' \leftarrow o'']$ $(\vec{o'} \in \nu(\vec{c'}))$, and suppose that $t'[\vec{o'}/\vec{x'}] \underset{A,I}{\triangleright} o$ is obtained from Definition 3.10 (3). By the inductive hypothesis, it holds that

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

Since $t'[\vec{o'}/\vec{x'}] = t[\vec{o}/\vec{x}][r'' \leftarrow o'']$, we have $t'[\vec{c'}/\vec{x'}] = t[\vec{c}/\vec{x}][r'' \leftarrow c'']$. Since $t'[\vec{o'}/\vec{x'}] \underset{A,I}{\triangleright} o \in P_I$ implies $t'[\vec{o'}/\vec{x'}] \xrightarrow{*} o$, it holds that $c \in Z(t'[\vec{c'}/\vec{x'}])$. From the above inductive hypothesis and Definition 3.13 (3), we have $t'[\vec{c'}/\vec{x'}] \underset{A,S}{\triangleright} c \in P_S$. □

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

Lemma 3.4: If $t[\vec{o}/\vec{x}] \underset{A,I}{\overset{*}{\Rightarrow}} t'[\vec{o'}/\vec{x'}]$ for some $\vec{o} \in \nu(\vec{c})$ and $\vec{o'} \in \nu(\vec{c'})$, then $t[\vec{c}/\vec{x}] \underset{A,S}{\overset{*}{\Rightarrow}} t'[\vec{c'}/\vec{x'}]$.

Proof: Consider the i-th step $t_{i-1}[\vec{o}_{i-1}/\vec{x}_{i-1}] \underset{A,I}{\Rightarrow} t_i[\vec{o}_i/\vec{x}_i]$ of the reduction $t[\vec{o}/\vec{x}] \underset{A,I}{\overset{*}{\Rightarrow}} t'[\vec{o'}/\vec{x'}]$. By Lemma 3.3, it is easily shown that $t_{i-1}[\vec{c}_{i-1}/\vec{x}_{i-1}] \underset{A,S}{\Rightarrow} t_i[\vec{c}_i/\vec{x}_i]$, where $\vec{o}_{i-1} \in \nu(\vec{c}_{i-1})$ and $\vec{o}_i \in \nu(\vec{c}_i)$. Since this reduction holds at an arbitrary step, the lemma holds. □

By Lemma 3.4, the following theorem is immediately shown.

Theorem 3.4: Let τ be a term in $T_M(C)$ to be verified. If there exists no class c such that $\tau \underset{A,S}{\overset{*}{\Rightarrow}} c$, then no security flaw on τ occurs, i.e., there exist no interpretation I and $\vec{O}$ such that $\tau[\vec{o}/\vec{c}] \underset{A,I}{\overset{*}{\Rightarrow}} o$ for any $\vec{o} \in \nu(\vec{c})$ and $o \in O_{S,I}$. □

Example 3.15: For S_2 in Figure 3.1, let $\tau'_0 = admin(leader(\textsf{employee}))$ be a term in $T_M(C)$ to be verified. Since no subterm of τ'_0 can be rewritten by any rule in Figure 3.13, no security flaw on τ'_o occurs. □

3.5.3 The Case of Monadic Method Schemas

This section shows that the sufficient condition in Theorem 3.4 is also a necessary one if a given method schema is monadic and Z satisfies that, for each $t \in T_M(C)$,

$$\begin{aligned} Z(t) = \{c' | &\text{there exists an interpretation such that} \\ &t[o/c] \xrightarrow{*} o' \text{ for } o \in \nu(c) \text{ and } o' \in \nu(c')\}. \end{aligned} \tag{3.1}$$

Before proving the above statement, the following interpretation is introduced.

Definition 3.14: Let N be a positive integer. Define a *syntactic interpretation* $I_S = (\nu_{I_S}, \mu_{I_S})$ of S as follows:

1. For each $c \in C$,

$$\nu_{I_S}(c) = \{c \cdot \alpha | \alpha \in C^* \text{ and the length of } c \cdot \alpha \text{ is at most } N\},$$

where C^* denotes the Kleene closure of C.

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

2.1 If $Res(m_b, c_0) = c'$, then $\mu_{I_S}(m_b)(c_0) = c'$, and, for $w \geq 1$,

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

2.2 If $Res(m_b, c_0) = \perp$, then $\mu_{I_S}(m_b)(c_0 \cdot c_1 \cdot c_2 \cdots \cdot c_w)$ $(w \geq 0)$ is undefined. □

Hereafter consider a syntactic interpretation with sufficiently large N. If Z satisfies Equation (3.1), then it also satisfies the following Lemmas 3.5 through 3.7.

Lemma 3.5: Let $t \in T_M(\{x\})$, and suppose that $c' \in Z(t[c/x])$. There exists $\beta \in C^*$ such that

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

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

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

Proof: Suppose that $c' \in Z(t[c/x])$. By Equation (3.1), there exists an interpretation $I = (\nu, \mu)$ such that $t[o/x] \xrightarrow{*} o'$ for some $o \in \nu(c)$ and $o' \in \nu(c')$. Consider the i-th step (counting from zero) $t_i[o_i/x] \rightarrow t_{i+1}[o_{i+1}/x]$ of the reduction $t[o/x] \xrightarrow{*} o'$, where $t_0[o_0/x] = t[o/x]$ and $t_n[o_n/x] = o'$. Let c_i $(0 \leq i \leq n-1)$ be the class such that $o_i \in \nu(c_i)$, and $m_i(o_i)$ the innermost term of $t_i[o_i/x]$. Define β_i $(0 \leq i \leq n-1)$ as follows:

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

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

It is easily verified that β satisfies condition 1 since

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

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

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

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

Lemma 3.6: Let $t, t', t'' \in T_M(\{x\})$ such that t'' is a subterm of t at r'' and $t' = t[r'' \leftarrow x]$. If both $c'' \in Z(t''[c/x])$ and $c' \in Z(t'[c''/x])$, then $c' \in Z(t[c/x])$.
Proof: By Lemma 3.5, there exist reduction strings β'' of $(t''[c/x], c'')$ and β' of $(t'[c''/x], c')$, i.e., for any $\alpha'', \alpha' \in C^*$,

- $t''[\beta'' \cdot c'' \cdot \alpha''/x] \xrightarrow{*} c'' \cdot \alpha''$ and the first symbol of $\beta'' \cdot c''$ is c; and
- $t'[\beta' \cdot c' \cdot \alpha'/x] \xrightarrow{*} c' \cdot \alpha'$ and the first symbol of $\beta' \cdot c'$ is c''.

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

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

Since Z satisfies Equation (3.1), c' must be in $Z(t[c/x])$. □

Lemma 3.7: Let $t, t', t'' \in T_M(\{x\})$ such that t'' is a subterm of t at r'' and $t' = t[r'' \leftarrow x]$. Suppose that $c'' \in Z(t''[c/x])$ and $c' \in Z(t'[c''/x])$. Let β' be an arbitrary reduction string of $(t'[c''/x], c')$. Then, there exist reduction strings β of $(t[c/x], c')$ and β'' of $(t''[c/x], c'')$ such that $\beta = \beta'' \cdot \beta'$.
Proof: Let β'' and β' be arbitrary reduction strings of $(t''[c/x], c'')$ and $(t'[c''/x], c')$ respectively. By the proof of Lemma 3.6, $\beta'' \cdot \beta'$ is a reduction string of $(t[c/x], c')$. This fact implies the lemma. □

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

Lemma 3.8: Let $c, c' \in C$, $t \in T_M(\{x\})$, and $t' \in T_M(\{x'\})$. If $t[c/x] \underset{A,S}{\triangleright} c' \in P_S$, then, for an arbitrary reduction string β of $(t[c/x], c')$ and for any $\alpha \in C^*$,

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

Proof: We use induction on the number of the repetition of a procedure which computes the least fixed point satisfying the three conditions in Definition 3.13.
Basis: Consider the following two cases:

1. The case that $m(c) \underset{A,S}{\triangleright} c'$ is obtained from Definition 3.13 (1): Let β be an arbitrary reduction string of $(m(c), c')$. Since $(m, c) \in A$ and $m(\beta \cdot c' \cdot \alpha) \xrightarrow{*} c' \cdot \alpha$, we obtain $m(\beta \cdot c' \cdot \alpha) \underset{A,I_S}{\triangleright} c' \cdot \alpha$ from Definition 3.10 (1).
2. The case that $Res(m_u, c)[c/x] \underset{A,S}{\triangleright} c'$ is obtained from Definition 3.13 (2): It can be proved in the same way as the above case.

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

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

where $t' = t[r'' \leftarrow x]$. Also, suppose that $t'[c''/x] \underset{A,S}{\triangleright} c'$ is obtained from Definition 3.13 (3). Since $t''[c/x] \underset{A,S}{\triangleright} c'' \in P_S$, it holds that $c'' \in Z(t''[c/x])$ by Definition 3.13. By Lemma 3.6, it holds that $c' \in Z(t[c/x])$. Then, by Definition 3.13 again, $t[c/x] \underset{A,S}{\triangleright} c'$ must be in P_S. Let β' be an arbitrary reduction string of $(t'[c''/x], c')$. That is, the first symbol of $\beta' \cdot c'$ is c'' and $t'[\beta' \cdot c' \cdot \alpha'/x] \xrightarrow{*} c' \cdot \alpha'$ for any $\alpha' \in C^*$. By Lemma 3.7 and the inductive hypothesis for $t[c/x] \underset{A,S}{\triangleright} c'$ and $t''[c/x] \underset{A,S}{\triangleright} c''$, there exist β and β'' such that, for any α and α'',

- $t[\beta \cdot c' \cdot \alpha/x] \underset{A,I_S}{\triangleright} c' \cdot \alpha \in P_{I_S}$ and the first symbol of $\beta \cdot c'$ is c;
- $t''[\beta'' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\triangleright} c'' \cdot \alpha'' \in P_{I_S}$ and the first symbol of $\beta'' \cdot c''$ is c; and
- $\beta = \beta'' \cdot \beta'$.

When we choose α and α'' so that $\beta' \cdot c' \cdot \alpha = c'' \cdot \alpha''$, it holds that $\beta \cdot c' \cdot \alpha = \beta'' \cdot c'' \cdot \alpha''$. By Definition 3.10 (3),

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

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

Lemma 3.9: Let $c, c' \in C$ and $t, t' \in T_M(\{x\})$. If $t[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} t'[c''/x]$, then there exists a string β such that the first symbol $\beta \cdot c''$ is c and, for any $\alpha'' \in C^*$,

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

Proof: We use induction on the length of the reduction $t[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} t'[c''/x]$.
Basis: The lemma holds evidently when the length of the reduction is zero.
Induction: Consider the following reduction that $t[c/x] \underset{A,S}{\overset{*}{\Rightarrow}} t_i[c_i/x] \underset{A,S}{\Rightarrow} t'[c''/x]$. By the inductive hypothesis, there exists a string β_i such that the first symbol of $\beta_i \cdot c_i$ is c and, for any $\alpha_i \in C^*$,

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

By Definition 3.13, there exists a subterm t'' of t_i at r'' such that $t''[c_i/x] \underset{A,S}{\triangleright} c''$ and $t'[c''/x] = t_i[c_i/x][r'' \leftarrow c'']$. By Lemmas 3.5 and 3.8, there exists β' such that the first symbol of $\beta' \cdot c''$ is c_i and, for any $\alpha'' \in C^*$, rule $t''[\beta' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\triangleright} c'' \cdot \alpha''$ exists in P_{I_S}. By Definition 3.10, it follows that $t''[\beta' \cdot c'' \cdot \alpha''/x] \underset{A,I_S}{\Rightarrow} c'' \cdot \alpha''$. Hence,

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

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

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

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

Theorem 3.5: Let S be a monadic method schema and τ a term in $T_M(C)$ to be verified. Suppose that Z satisfies Equation (3.1). If there exists a class c' such that $\tau \overset{*}{\underset{A,S}{\Rightarrow}} c'$, then a security flaw on τ occurs, i.e., there exist an interpretation I and $\tilde{O}$ such that $\tau[o/c] \overset{*}{\underset{A,I}{\Rightarrow}} o'$ for some $o \in \nu(c)$ and $o' \in O_{S,I}$. □

Example 3.16: For S_2 in Figure 3.1, let $\tau_1' = admin(boss(\mathsf{employee}))$ and $\tau_2' = service(\mathsf{server})$ be terms in $T_M(C)$ to be verified. By applying the rewriting rules (1) and (6) in Figure 3.13 to τ_1' sequentially, τ_1' can be computed as follows:

$$\tau_1' = admin(boss(\mathsf{employee})) \underset{A_2,S_2}{\Rightarrow} admin(\mathsf{staff}) \underset{A_2,S_2}{\Rightarrow} \mathsf{use}.$$

Since S_2 is monadic, a security flaw on τ_1' occurs by Theorem 3.5. In the same way, a security flaw on τ_2' occurs since τ_2' can be computed as follows:

$$\tau_2' = service(\mathsf{server}) \underset{A_2,S_2}{\Rightarrow} \mathsf{use}.$$

Actually, in Example 3.12, security flaws on τ_1 and on τ_2 occur. □

3.5.4 The Algorithm and its Complexity

The algorithm for deciding the sufficient condition stated in Theorem 3.4 consists of the following three steps.

Step 1: Compute Z from S.

Step 2: Compute P_S from S, A, and Z.

Step 3: Determine whether there exists a class c such that $\tau \overset{*}{\underset{A,S}{\Rightarrow}} c$.

Using the type-checking algorithm in [30], Z which is fairly small can be computed, and, for a monadic schema, Z satisfying Equation (3.1) can be computed in polynomial time of $\|S\|$.

The following summarizes the time complexity of the algorithm. Let H be the maximum size of all t such that $(m, \vec{c}, t) \in \Sigma_u$, and l the size of a given term τ to be verified.

Before explaining the above algorithm, define a finite set T_A as $\{t | t \underset{A,S}{\triangleright} c \in P_S\}$, and consider the size of T_A. Let h be the height of a tree representing a term

$t \in T_M(C)$. Also, let X_h be a finite set of trees obtained by replacing subterms of t with classes, where X_h contains both t itself and c such that $c \in Z(t)$. For example, let $t = m_1(m_2(c_1, c_2), c_3)$, which is represented as a tree with height two. If $Z(m_2(c_1, c_2)) = \{c_1, c_2\}$, $Z(m_1(c_1, c_3)) = \{c_1\}$, and $Z(m_1(c_2, c_3)) = \{c_2\}$, then, for such t,

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

By solving the following recurrence formula

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

the following result is obtained

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

Since X_h contains T_A for all terms $t \in T_M(C)$ such that $Res(m, \vec{c}) = t$ for $m \in M_u$, the size of T_A is

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

After all, for $k \geq 1$,

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

Before computing Z in Step 1, define a finite subset $Z_0(m(\vec{c}))$ of C such that, for $m \in M_n$ and $\vec{c} \in C^n$,

$$\begin{aligned} Z_0(m(\vec{c})) \;=\; \{c | &\text{there exists an interpretation such that} \\ &m(\vec{o}) \xrightarrow{*} o \text{ for } \vec{o} \in \nu(\vec{c}) \text{ and } o \in \nu(c)\}. \end{aligned}$$

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

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

which is given in [30]. The time complexity t_{Z_0} to retrieve an element from $Z_0(m(\vec{c}))$ is

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

if Z_0 is implemented as a 2-3 tree. Next, compute $Z(t)$ for $t \in T_M(C)$ as follows. If $m(\vec{c})$ is the innermost term of t, then replace $m(\vec{c})$ with c for each $c \in Z_0(m(\vec{c}))$. Repeat this replacement until t is rewritten to a class. The time complexity to compute $Z(t)$ is

$$\mathcal{O}(t_{Z_0}|C|^k\|t\|) = \mathcal{O}(kH|C|^k \log \|S\|).$$

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

$$\mathcal{O}(k|C|^{2k+1}\|S\| + |T_A|kH|C|^k \log \|S\|) = \mathcal{O}(kH|C|^{k^H+k}\|S\| \log \|S\|). \quad (3.2)$$

The time complexity t_Z to retrieve an element from $Z(t)$ is

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

if Z is implemented as a 2-3 tree.

In order to execute Step 2, it is sufficient to compute T_A. Suppose that A, Res, and T_A are implemented as 2-3 trees. Figure 3.14 shows a procedure CONSTRUCT-REWRITING-RULES to compute T_A. Retrieving an element from A and Res takes $t_A = \mathcal{O}(k \log \|S\|)$ time and $t_{Res} = \mathcal{O}(k \log \|S\|)$ time respectively. Retrieving an element from T_A and inserting an element into T_A takes

$$t_{T_A} = \mathcal{O}(\log |T_A|) = \mathcal{O}(k^H \log \|S\|)$$

time since $|T_A| = \mathcal{O}(|C|^{k^H}\|S\|)$, Therefore, executing lines (1)–(5) takes

$$\mathcal{O}(|A|(t_A + t_{T_A} + \log |M_u| + t_{Res} + t_{T_A})) = \mathcal{O}(k^H|C|^k\|S\| \log \|S\|)$$

time, and executing lines (6)–(11) takes

$$\mathcal{O}(|T_A||T_A|(t_{T_A} + H^2 + t_Z + |C|t_{T_A})) = \mathcal{O}(|C|^{2k^H}\|S\|^2(H^2 + k^H|C| \log \|S\|))$$

time. Thus, the total time of Step 2 is

$$\mathcal{O}(|C|^{2k^H}\|S\|^2(H^2 + k^H|C| \log \|S\|)). \quad (3.3)$$

Lastly, consider Step 3. For a given term $\tau \in T_M(C)$, define a finite set D_i as follows:

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

procedure CONSTRUCT-REWRITING-RULES

Input : a method schema S, an authorization A, and Z

Output: a finite set $T_A = \{t | t \underset{A,S}{\triangleright} c \in P_S\}$

begin

1: $T_A \leftarrow \phi$;
2: **for each** $(m, \vec{c})$ in A
3: $T_A \leftarrow T_A \cup m(\vec{c})$;
4: **if** $m \in M_u$ **then**
5: $T_A \leftarrow T_A \cup Res(m, \vec{c})[\vec{c}/\vec{x}]$;
6: **repeat**
7: **for each** t, t' in T_A
8: **if** t'' is a subterm of t at r'' **then**
9: **for each** c'' in $Z(t'')$
10: $T_A \leftarrow T_A \cup t[r'' \leftarrow c'']$;
11: **until** T_A does not change

end

Figure 3.14 Procedure to construct rewriting rules $\underset{A,S}{\triangleright}$.

procedure REWRITE-A-TERM

Input : a method schema S, Z, T_A, and a term t in $T_M(C)$

Output: a finite set D satisfying $t' \in D$ such that $t \underset{A,S}{\Rightarrow} t'$

begin

1: $D \leftarrow \phi$;
2: **for each** t'' in T_A
3: **if** t'' is a subterm of t at r'' **then**
4: **for each** c'' in $Z(t'')$
5: $D \leftarrow D \cup t[r'' \leftarrow c'']$;

end

Figure 3.15 Procedure to execute rewriting $\underset{A,S}{\Rightarrow}$.

First compute D_1 and then $D_2, D_3, \ldots$ until $D_j = D_{j+1}$. It holds that $j \leq l$ since $\|t\| > \|t'\|$ for each pair t and t' such that $t \underset{A,S}{\triangleright} t' \in P_S$. The size of each D_i is at most $|X_l|$. Figure 3.15 shows a procedure REWRITE-A-TERM to compute all t' such that $t \underset{A,S}{\Rightarrow} t'$ for a given $t \in T_M(C)$. Suppose that executing line (5) is in constant time. Executing lines (1)–(5) takes

$$\mathcal{O}(|T_A|(t_{T_A} + lH + t_Z + |C|)) = \mathcal{O}(|C|^{k^H}\|S\|(k^H \log \|S\| + lH + |C|))$$

time. Since the procedure in Figure 3.15 is executed for each term in D_i $(1 \leq i \leq j)$, the total time complexity of Step 3 is

$$\begin{aligned} &\mathcal{O}(j|X_l||C|^{k^H}\|S\|(k^H \log \|S\| + lH + |C|)) \\ &= \begin{cases} \mathcal{O}(l|C|^{k^l+k^H}\|S\|(k^H \log \|S\| + lH + |C|)) & \text{if } k \geq 2, \\ \mathcal{O}(l^2|C|^2\|S\|(\log \|S\| + lH + |C|)) & \text{if } k = 1. \end{cases} \end{aligned} \tag{3.4}$$

After all, the time complexity of the algorithm is obtained by Equations(3.2)–(3.4), By letting $L = \max\{l, H\}$, the time complexity is

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

Moreover, by assuming $L \leq \|S\|$, the time complexity is

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

This result leads to the following theorem.

Theorem 3.6: The sufficient condition stated in Theorem 3.4 is decidable. □

Corollary 3.2: For a monadic method schema, the detection problem of security flaws is solvable in polynomial time of the size of the schema. □

3.6 Variations of the Detection Problem of Security Flaws

Section 3.4 has provided the method to decide whether or not, for given S, I, A, and $\tau \in T_M(O_{S,I})$, there exists $o \in O_{S,I}$ such that $\tau \underset{A,I}{\approx} o$. An authorization A

is called *safe* on τ if there exists no such object o. That A is unsafe on τ means that a prohibited data can be obtained under A. Consider the following problem of finding A which is safe on τ.

Definition 3.15: Let S be a method schema, I an interpretation, A an authorization, and τ a term in $T_M(O_{S,I})$ to be verified. The *finding problem of a safe authorization for OODB instances*, or simply the *finding problem for instances*, is to find a maximal safe subset $A' \subseteq A$ on τ for given S, I, A, and τ. □

Theorem 3.7: The finding problem of a safe authorization for OODB instances is solvable in polynomial time if $|O|^k = \mathcal{O}(|\mu|)$.

Proof: An algorithm to find a safe authorization is shown in Figure 3.16. The subset A' obtained from A by FIND-A-SAFE-AUTHORIZATION is obviously safe. Suppose that A' is not maximal. By this assumption, there exists $a' \in A - A'$ such that $A' \cup \{a'\}$ is safe, that is, $\tau \underset{A' \cup \{a'\}, I}{\approx} o$ does not hold for any $o \in O_{S,I}$. On the other hand, by the definition of the algorithm, $a' \notin A'$ implies that there exists some $A'' \subseteq A'$ such that $A'' \cup \{a'\}$ is not safe on τ, that is, $\tau \underset{A'' \cup \{a'\}, I}{\approx} o$ holds for some $o \in O_{S,I}$. It follows that $\tau \underset{A' \cup \{a'\}, I}{\approx} o$ also holds. This leads to contradiction. Therefore, A' is a maximal safe authorization.

Since the detection problem for instances at line (3) in Figure 3.16 is tested $|A|$ times and is solvable in polynomial time from Corollary 3.1 if $|O|^k = \mathcal{O}(|\mu|)$, the finding problem for instances is also solvable in polynomial time. □

It often happens that a priority is given for each element of an authorization and computing a maximum safe authorization according to the priority is expected. Another variation of the finding problem for instances is to find a safe authorization $A' \subseteq A$ such that, when a priority is given for each element of A, the sum of the priorities of the elements of A' is maximum. The problem is obviously solvable in PSPACE, but tighter upper/lower bounds are open. As a special case of the problem, when the priority of a_i $(1 \leq i \leq n)$ is 2^{n-i} for each $a_i \in A$, the problem is obviously solvable in polynomial time by the algorithm in Figure 3.16, where elements in A are checked in the order $a_1, a_2, \ldots, a_n$.

For $\tau \in T_M(C)$, the concept of a safe authorization on τ can be also introduced. Consider the following problem.

procedure FIND-A-SAFE-AUTHORIZATION
Input : a method schema S, an interpretation I, an authorization A, and a term τ in $T_M(O_{S,I})$ to be verified
Output: a maximal safe subset A' of A on τ
begin
1: $A' \leftarrow \phi$;
2: **foreach** a **in** A
3: **if** $A' \cup \{a\}$ is safe on τ **then**
4: $A' \leftarrow A' \cup \{a\}$;
end

Figure 3.16 Procedure to find a safe authorization.

Definition 3.16: Let S be a method schema, A an authorization, and τ a term in $T_M(C)$ to be verified. The *finding problem of a safe authorization for OODB schemas* is to find a maximal safe subset $A' \subseteq A$ on τ for given S, A, and τ. □

If S is monadic, then the following theorem obviously holds in the same way as Theorem 3.7.

Theorem 3.8: The finding problem of a safe authorization for OODB schemas is solvable in polynomial time. □

3.7 Conclusions

In this chapter, the detection problems of security flaws for OODB instances and for OODB schemas have been discussed. Variations of the detection problems have also been mentioned.

As mentioned in Section 3.2, method schemas were adopted as a formal model of OODBs. Method schemas do not support sets of objects. That is, method schemas cannot define queries on sets of objects. Therefore, the method proposed in this section cannot be directly applied to other models of OODBs which support sets of objects (e.g., object-relational models, deductive object-oriented models). In order to support sets of objects, the definition of $\underset{A,I}{\approx}$ in Section 3.3.2 need to be modified. That is, inference rules on operations on sets have to be considered.

Also, method schemas do not support updates of a database interpretation. Therefore, this thesis does not discuss problems on updates of a database interpretation. For example, updating a database interpretation may change the safety of authorizations. By this change, a user may be able to obtain information on a database interpretation. Moreover, when integrity constraints are taken into account, the notion of the safety of authorizations may be extended so that every method in an authorization causes neither security flaws nor violation of integrity constraints. This makes the situation more complicated. Many other problems involving updates are left as future work.

The author intends to discuss the detection problem when, in the discussion of Section 3.3.2, a user knows either what $O_{S,I}$ is or what C is, or the user

knows both of them. Moreover, for a general schema, if Z satisfies that, for each $t \in T_M(C)$,

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

then whether or not the sufficient condition in Theorem 3.4 is also a necessary one is open.

Chapter 4

Conclusions

In Chapter 2, we have presented an authorization model which is independent of OODB schemas and authorization policies, and have defined an authorization specification language which is powerful enough to specify authorization policies proposed in the literature. Furthermore, an efficient access control method has been proposed. The proposed method partially computes inferences of authorizations from S and R in compile-time and decides whether a given access request is permitted or prohibited by using the results of compile-time. The time complexities are $\mathcal{O}(N^2(f_c(N) + N \log N))$ for compile-time and $\mathcal{O}(N^4(f_i(N, |O|) + N^3 \log N))$ for run-time. The simulation results has concluded that the proposed method makes the access control more efficient than conventional methods.

In Chapter 3, we have formally defined the concept of security flaws in OODBs and have analyzed user's inference under an authorization. The following problems on security flaws in OODBs have been discussed:

(1) The detection problem of security flaws for OODB instances.

(2) The detection problem of security flaws for OODB schemas.

(3) The finding problem of a safe authorization.

First, it has been shown in this thesis that the problem (1) is solvable in polynomial time in practical cases by reducing to the congruence closure problem. Next, this thesis has shown that the problem (2) is undecidable for general method

schemas and has provided a decidable sufficient condition for a given method schema to have no security flaw on a given term. Also, it has been shown in this thesis that the problem (2) is decidable in polynomial time of the size of the schema for a monadic schema. Lastly, this thesis has shown that the problem (3) is solvable in polynomial time in practical cases.

An authorization model provided in Chapter 2 can specify an authorization model flexibly and generally and contain authorization models proposed in the literature as special cases. By using the proposed model, database administrators can specify an appropriate authorization policy according to a database management policy. Furthermore, by the proposed access control method, the DBMS can achieve an access control to OODBs efficiently. Many papers have scarcely discussed the complexity of access control although they have proposed various authorization models. Some papers have roughly estimated the complexity and given methods under a restricted authorization model, e.g., one which cannot specify inference rules dependent on the contents of objects in OODBs.

Even though the DBMS enforces access control under a given authorization, malicious users may infer prohibited information from permitted information, i.e., a security flaw may occur. Many papers have scarcely provided algorithms to formally detect a security flaw and estimated the complexity. In Chapter 3, we have formally defined user's inferences and proposed algorithms to detect a security flaw for the problems (1)–(3). By detecting the possibilities of a security flaw, database administrators can eliminate prohibited inference by users' attack in advance and can keep the database more secure.

References

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

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

[3] R. Ahad, J. Davis, S. Gower, P. Lyngbaek, A. Marynowski, and E. Onuegbe, "Supporting access control in an object-oriented database language," Proc. 3rd Int'l Conf. on Extending Database Technology, LNCS 580, pp.184–200, 1992.

[4] A. Baraani-Dastjerdi, J. Pieprzyk, R. Safavi-Naini, and J.R. Getta, "A model of authorization for object-oriented databases based on object views," Proc. 4th Int'l Conf. on Deductive and Object-Oriented Databases, LNCS 1013, pp.503–520, 1995.

[5] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "A temporal access control mechanism for database systems," IEEE Transactions on Knowledge and Data Engineering, Vol.8, No.1, pp.67–80, Feb. 1996.

[6] E. Bertino, S. Jajodia, and P. Samarati, "Database security: research and practice," Information Systems, Vol.20, No.7, pp.537–556, Nov. 1995.

[7] E. Bertino, F. Origgi, and P. Samarati, "A new authorization model for object-oriented databases," Database Security, VIII(A-60): Status and Prospects, Elsevier Science Publishers, pp.199–222, 1994.

[8] E. Bertino and P. Samarati, "Research issues in discretionary authorizations for object bases," Proc. OOPSLA-93 Conference Workshop on Security for Object-Oriented Systems, pp.183–199, 1993.

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

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

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

[12] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, "Introduction to algorithms," The MIT Electrical Engineering and Computer Science Series, pp.446–450, The MIT Press, 1990.

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

[14] P.J. Downey, R. Sethi, and R.E. Tarjan, "Variations on the common subexpression problem," J. ACM, Vol.27, No.4, pp.758–771, 1980.

[15] E.B. Fernandez, R.B. France, and D. Wei, "A formal specification of an authorization model for object-oriented databases," Database Security, IX: Status and Prospects, Chapman & Hall, pp.95–110, 1996.

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

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

[18] E. Gudes, H. Song, and E.B. Fernandez, "Evaluation of negative, predicate, and instance-based authorization in object-oriented databases," Database Security, IV: Status and Prospects, Elsevier Science Publishers, pp.85–98, 1991.

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

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

[21] R. Hull, K. Tanaka, and M. Yoshikawa, "Behavior analysis of object-oriented databases: method structure, execution trees, and reachability," Proc. 3rd Int'l Conf. on Foundations of Data Organization and Algorithms, pp.372–388, June 1989.

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

[23] W. Kim, "Introduction to object-oriented databases," The MIT Press, Cambridge, 1990.

[24] G. Nelson and D.C. Oppen, "Fast decision procedures based on congruence closure," J. ACM, Vol.27, No.2, pp.356–364, 1980.

[25] G. O'Shea, "On the specification, validation and verification of security in access control systems," The Computer Journal, Vol.37, No.5, pp.437–448, 1994.

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

[27] K. Sakaguchi, T. Morita, Y. Ishihara, H. Seki, and M. Ito, "A content-based authorization model for object-oriented databases," IEICE Technical Report, DE96-80, pp.37-42, Jan. 1997.

[28] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-based access control models," IEEE Computer, Vol.29, No.2, pp.38-47, Feb. 1996.

[29] R.S. Sandhu and P. Samarati, "Access control: principles and practice," IEEE Communications Magazine, Vol.32, No.9, pp.40-48, Sep. 1994.

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

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

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

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