PAPER *Special Section on Cryptography and Information Security*

The Dynamic-Typed Access Matrix Model and Decidability of the Safety Problem

Masakazu SOSHI[†a)], **Mamoru MAEKAWA**[††b)], ***Nonmembers***, ***and*** **Eiji OKAMOTO**[†††c)], ***Member***

SUMMARY The safety problem in access matrix models determines whether a given subject can eventually obtain access privilege to a given object. Generally speaking, the safety problem is, unfortunately undecidable. Not much is known about protection systems for which the safety problem is decidable, except for strongly constrained systems (e.g., monotonic systems).

Therefore, we propose the Dynamic-Typed Access Matrix (DTAM) Model, which extends the Typed Access Matrix model of Sandhu by allowing the type of an object to change dynamically. The DTAM model has an advantage that it can describe non-monotonic protection systems for which the safety problem is decidable. In particular, with further restrictions, we can show that the problem becomes NP-hard. In this paper, we formally define the DTAM model and then discuss various aspects of it thoroughly.

key words: *access control, access matrix model, safety problem, computational complexity, decidability*

1. Introduction

Today a huge amount of valuable information is being processed and stored by computers and it is of great importance to establish security in such environments. A security model gives us a framework that specifies computer systems (or protection systems) precisely from a security point of view.

One of the most widely accepted security models is an *access matrix model.* With an access matrix model, a protection system is described in terms of *subjects* (e.g., users or processes) and *objects* (e.g., files or I/O devices). Access control is enforced according to an *access matrix* A, which has a row for each subject and a column for each object, and $A[s, o]$ maintains the set of access modes that subject s is authorized to perform on object o.

Harrison et al. first formalized the security property of protection systems in the access matrix model (HRU model) as the *safety problem* [8]. The safety problem is the one to determine whether or not a given subject can eventually obtain an access privilege to a given object. Generally speaking, unfortunately, the safety problem is undecidable [8], [19]. This is primarily due to the fact that the access matrix model has broad expressive power and that the number of newly created objects can be infinite. Little is known about protection systems for which the safety problem is decidable, except for strongly constrained systems (e.g., *monotonic* systems, where no new entities can be created and no revocation of privileges is allowed) [7], [8], [16], [18], [22]. For example, Sandhu developed the Typed Access Matrix (TAM) Model [18], which has a wide variety of decidable safety cases, but most of which are limited to monotonic systems. However, since security policies in existent computer systems are not monotonic, it would be difficult to apply the safety analysis of monotonic systems to real systems.

Therefore, we propose the Dynamic-Typed Access Matrix (DTAM) model[*], which extends the TAM model by allowing the type of an object to change dynamically. The DTAM model has an advantage that it can describe nonmonotonic protection systems for which the safety problem is decidable. In order to show this, first we introduce a *type relationship (TR) graph.* Then we show that the safety problem for nonmonotonic systems becomes decidable if, roughly speaking, the TR graphs of the systems have no cycle with respect to the parent-child relationship between objects.[**] Moreover, if we impose on this situation additional restrictions that no new objects are permitted to be created, the safety problem becomes NP-hard. The decidable safety cases discussed in this paper fall outside the known decidable ones in previous work [7], [8], [12], [14], [16], [18], [22].

The remainder of the paper is structured as follows. First, we present the background of our work in Sect. 2 and point out the problems in previous work. In order to solve them, we propose and formalize the DTAM model in Sect. 3. Next, we undertake a thorough investigation of the safety property of the DTAM model in Sect. 4. In Sect. 5, we discuss various topics

[†]The author is with the Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Tatsunokuchi-machi, Nomi-gun, Ishikawa 923-1292, JAPAN.

[††]The author is with the Graduate School of Information Systems, University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, JAPAN.

[†††]The author is with the Institute of Information Sciences and Electronics, University of Tsukuba, 1-1-1 Tennodai, Tsukuba-shi, Ibaraki-ken 305, JAPAN.

a) E-mail: soshi@jaist.ac.jp

b) E-mail: maekawa@is.uec.ac.jp

c) E-mail: okamoto@is.tsukuba.ac.jp

[*]This work is extension of [24], [25].

[**]The precise condition for this case will be given in Sect. 4.

on the DTAM model, and then finally, we conclude this paper in Sect. 6.

2. Related Work

This section discusses previous work related to ours.

Harrison et al. formulated and analyzed the safety problem of access matrix models for the first time [8]. Their model is a state transition model and is well known as the 'HRU' model. The state of a protection system in the HRU model is a tuple of subjects, objects, and authorization represented by an access matrix. The state is updated by *commands*, each of which consists of a *condition* and a *body*. The condition part of a command is a conjunction of conditional expressions, each of which tests authorization in the access matrix of concern. Furthermore, the body of a command is a sequence of *primitive operations*, i.e., enter/delete access rights, and create/destroy objects. The primitive operations in the body can be executed in order if and only if all the conditions in the condition part are satisfied in the state. Now we can define the *safety problem* in the HRU model as the one to determine whether or not a given subject can eventually obtain an access privilege to a given object. Namely, informally speaking, we can say that the safety problem models confidentiality and integrity in computer security.

Generally speaking, unfortunately, the safety problem is undecidable. Harrison et al. proved it by reducing the Halting Problem of Turing Machines to the safety problem [8]. Such undecidability is due to the fact that the HRU model has rich expressive power and can have infinitely many objects to be created. Even in protection systems where objects are prohibited from being newly created, the safety problem is P-space complete. Harrison et al. also showed that the safety problem in mono-operational protection systems, where the body of every command is allowed to have only one primitive operation, is decidable but NP-complete [8]. Harrison and Ruzzo also explored other decidable cases, where protection systems are monotonic and mono-conditional, that is, the systems that contain no commands with destroy and delete and at the same time no commands other than those with at most one condition expression in their condition parts [7].

Therefore, many researchers have so far investigated decidable cases for the safety problem [7], [8], [12], [14], [16], [18], [22]. In particular, Sandhu has achieved great success in this area [16]–[19], [21]. For example, Sandhu developed the Typed Access Matrix (TAM) Model [18]. The TAM model bears a close resemblance to the HRU model, but the former has strong typing in objects. As a result, the TAM model has a wide variety of decidable safety cases. However, unfortunately, most of the cases are limited to monotonic systems. Since security policies in existent computer systems are not monotonic, it would be difficult to apply the safety analysis of monotonic systems to real systems. To sum up, we can say that not much is known about protection systems for which the safety problem is decidable, except for strongly constrained systems. Readers can find a comprehensive survey of the topic in [12], [14], [18].

Therefore, we propose the Dynamic-Typed Access Matrix (DTAM) model, which extends the TAM model by allowing the type of an object to change dynamically. The DTAM model has an advantage that it can describe nonmonotonic protection systems for which the safety problem is decidable. The decidable safety cases discussed in this paper fall outside the known decidable ones in previous works.

3. Dynamic-Typed Access Matrix Model

In this section, we give a formal description of the Dynamic-Typed Access Matrix (DTAM) model.

3.1 Basic Concepts

Definition 1: *Objects* are defined as passive entities, e.g., files or I/O devices, which are protected by the security control mechanism of a computer system, and *subjects* as active entities, e.g., users or processes, which access objects. The current set of subjects and objects are denoted by S and O, respectively. We assume $S \subseteq O$. Each member of the set $O - S$ is called a *pure object* [18]. ■

Every object has its own inherent *identity*. Hence, for instance, no object can be created repeatedly as an identical one. In this paper, the identities of a subject and an object are represented by s and o, respectively ($s \in S$, $o \in O$).

Definition 2: Every object has a *type*, which can be changed dynamically. L is a finite set of all types. In particular, we denote a set of types of subjects by L_S ($L_S \subseteq L$). We assume $1 \leq |L_S| \leq |L|$. ■

For example, L_S may consist of three user types, *programmer*, *system-engineer*, and *project-manager*. In addition, we can take $\mathit{file}_{trade-secrets}$ and file_{public} as examples of types of pure objects.

Next, we define the type function as follows:

Definition 3: First, we define the function which returns the type of a subject as $f_S : S \to L_S$. Next, we define the function which returns the type of a pure object as $f_O : (O - S) \to (L - L_S)$. Now we can define the *type function* $f_L : O \to L$, which associates a type with every object, as follows:

$$f_L(o) = \begin{cases} f_S(o) & \text{if } o \in S, \\ f_O(o) & \text{otherwise.} \end{cases}$$

■

Note that mapping from objects to their types expressed by f_L may vary as time elapses because object types can be dynamically changed.

Definition 4: *Access modes* are kinds of access that subjects can execute on objects (e.g., *read*, *write*, *own*, and *execute*) and a finite set of access modes is denoted by R. ■

Using Def. 4, an access matrix can be defined as follows:

Definition 5: An *access matrix* A is a matrix which has a row for each subject and a column for each object. An element $A[s,o]$ of A stores the set of access modes ($A[s,o] \subseteq R$) that subject s is authorized to exercise on object o. ■

Now we can define a protection state (or state for short) of a system as follows:

Definition 6: A *protection state* is defined by (S, O, A, f_L) and denoted by Q. ■

3.2 Primitive Operations and Commands

The way in which a protection system evolves by activities of subjects is modeled by incremental changes of the protection state, which are made by executing a sequence of commands. In this section, we first define primitive operations in order to give the definition of commands.

Definition 7: The definition of *primitive operations* is given in Table 1, where the states just before and after a primitive operation executes are indicated by (S, O, A, f_L) and (S', O', A', $f_L{}'$), respectively. ■

Most notable primitive operations in the DTAM model are **change type of subject** s **to** $l_s{}'$ and **change type of object** o **to** $l_o{}'$. It is often desirable to change the type of an object dynamically to specify security policies in real computer systems [4], [15]. For the example in Sect. 3.1, if a user who is a programmer is promoted, first to the position of a system engineer, and next to a project manager, then such a situation is easily expressed by dynamically changing the user type accordingly. Dynamically changeable types are also advantageous in safety analysis (Section 4).

We shall define commands based on Def. 7.

Definition 8: A *command* is a computational unit which has the form:

```
command $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$
  if $r_1 \in A[x_{k_{s1}}, x_{k_{o1}}]$
    $\wedge\ r_2 \in A[x_{k_{s2}}, x_{k_{o2}}]$
    $\wedge \ldots$
    $\wedge\ r_m \in A[x_{k_{sm}}, x_{k_{om}}]$
  then
    $op_1$
    $op_2$
    $\ldots$
    $op_n$
end
```

Here, α is the name of the command, and x_1, x_2, …, x_k are formal parameters of α whose types are given by l_1, l_2, …, l_k, respectively. Furthermore, k_{s1}, k_{s2}, …, k_{sm}, k_{o1}, k_{o2}, …, k_{om} are integers between 1 and k. r_1, r_2, …, r_m are access modes and op_1, op_2, …, op_n are primitive operations. We assume that k, m, and n are finite. CM denotes a finite set of commands.

As defined above, a command consists of the condition and the body. *Condition* of a command is the predicate placed between **if** and **then** in the command, where we can specify the conjunction of multiple condition expressions. However, a command does not necessarily have the condition. A command with no condition is said to be an *unconditional command*. A condition expression in the condition of a command tests for the presence of an access mode in a cell of A. Finally, the *body* of a command is the sequence of the primitive operations contained in the command. ■

A command is invoked by replacing all formal parameters of the command with actual parameters (i.e., objects) of the appropriate types. After that, if the condition of the command and all of the conditions of the primitive operations in the body are evaluated to true in terms of the actual parameters, then the command (more precisely, the primitive operations in the body with actual parameters) can be executed. Otherwise, the command cannot be executed. Furthermore, we assume that every execution of commands is serial and atomic.

3.3 Examples of Commands

To demonstrate the expressive power of the DTAM model, we show some examples of commands in this section.

First, let us consider a security policy where a user who creates an object becomes the owner of the object and can control the transfer of access rights for the object [3], [8]. Such a policy or the derivatives are familiar to us and we can find them implemented on various computer systems.

Now the policy just described can be expressed in the following commands, 'create-file' and 'confer-read':

```
command create-file($x_1$:$user$, $x_2$:$file$)
  create object $x_2$ of type $file$
  enter own into $A[x_1, x_2]$
end

command confer-read($x_1$:$user$, $x_2$:$user$, $x_3$:$file$)
  if own $\in A[x_1, x_3]$
  then
    enter read into $A[x_2, x_3]$
end
```

Table 1 DTAM primitive operations

Primitive Operations	Conditions	New States
enter r **into** $A[s,o]$	$s \in S$ $o \in O$ $r \in R$	$S' = S$, $O' = O$ $A'[s,o] = A[s,o] \cup \{r\}$ $A'[s',o'] = A[s',o']$ if $(s',o') \neq (s,o)$, for all $s' \in S$, $o' \in O$ ${f_L}'(o') = f_L(o')$ for all $o' \in O$
delete r **from** $A[s,o]$	$s \in S$ $o \in O$ $r \in R$	$S' = S$, $O' = O$ $A'[s,o] = A[s,o] - \{r\}$ $A'[s',o'] = A[s',o']$ if $(s',o') \neq (s,o)$, for all $s' \in S$, $o' \in O$ ${f_L}'(o') = f_L(o')$ for all $o' \in O$
change type of subject s **to** ${l_s}'$	$s \in S$ ${l_s}' \in L_S$	$S' = S$, $O' = O$ $A'[s',o] = A[s',o]$ for all $s' \in S$, $o \in O$ ${f_L}'(s) = {l_s}'$ ${f_L}'(o) = f_L(o)$ if $o \neq s$, for all $o \in O$
change type of object o **to** ${l_o}'$	$o \in O - S$ ${l_o}' \in L - L_S$	$S' = S$, $O' = O$ $A'[s,o'] = A[s,o']$ for all $s \in S$, $o' \in O$ ${f_L}'(o) = {l_o}'$ ${f_L}'(o') = f_L(o')$ if $o' \neq o$, for all $o' \in O$
create subject s' **of type** l_s	$s' \notin O$ $l_s \in L_S$	$S' = S \cup \{s'\}$, $O' = O \cup \{s'\}$ $A'[s,o] = A[s,o]$ for all $s \in S$, $o \in O$ $A'[s',o] = \phi$ for all $o \in O$ $A'[s,s'] = \phi$ for all $s \in S'$ ${f_L}'(s') = l_s$ ${f_L}'(o) = f_L(o)$ for all $o \in O$
create object o' **of type** l_o	$o' \notin O$ $l_o \in L - L_S$	$S' = S$, $O' = O \cup \{o'\}$ $A'[s,o] = A[s,o]$ for all $s \in S$, $o \in O$ $A'[s,o'] = \phi$ for all $s \in S$ ${f_L}'(o') = l_o$ ${f_L}'(o) = f_L(o)$ for all $o \in O$
destroy subject s	$s \in S$	$S' = S - \{s\}$, $O' = O - \{s\}$ $A'[s',o] = A[s',o]$ for all $s' \in S'$, $o \in O'$ ${f_L}'(o) = f_L(o)$ for all $o \in O'$
destroy object o	$o \in O - S$	$S' = S$, $O' = O - \{o\}$ $A'[s,o'] = A[s,o']$ for all $s \in S'$, $o' \in O'$ ${f_L}'(o') = f_L(o')$ for all $o' \in O'$

As another example, consider a security policy where trusted subjects (e.g., security officers) can degrade the security level of information [15]. Then it is almost straightforward to give the command 'downgrade' below:

```
command downgrade($x_1$:$security\_officer$,
                  $x_2$:$file_{high}$)
  if own $\in A[x_1, x_2]$
  then
    change type of object $x_2$ to $file_{low}$
end
```

3.4 Authorization Schemes and Protection Systems

In this section, we define an authorization scheme and a protection system, which are abstractions of security policies and computer systems, respectively [18]:

Definition 9: An *authorization scheme* is defined by (L_S, L, R, CM). Furthermore, a *protection system* (or simply *system*) consists of an authorization scheme and an initial state (S_0, O_0, A_0, f_{L0}). ■

Next we consider monotonicity and nonmonotonicity of authorization schemes and protection systems.

Definition 10: An authorization scheme whose commands do not contain primitive operations **destroy**, **delete**, and **change type** is said to be *monotonic*. An authorization scheme which is not monotonic is said to be *nonmonotonic*. Furthermore, if the authorization scheme of a system is monotonic, the system is said to be monotonic, otherwise it is nonmonotonic. ■

This completes the formalization of the DTAM model.

4. Safety Analysis

In this section, we shall study the safety problems in the DTAM model thoroughly.

4.1 Preliminaries

First, in this section, we present some preliminaries that make the analysis easier.

Definition 11: *Normalization of command* $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ is to perform the following two transformations on α for every formal parameter x_i (1

$\leq i \leq k$). However, if α has no **change type of subject** (or **object**) x_i in its body, then the two transformations have no effect on it with respect to x_i. In the description below, we assume for simplicity that x_i is a subject. If x_i is a pure object, we transform α in a similar manner.

[**Transformation 1**] If α has only one **change type of subject** x_i, then the transformation 1 has no effect on it with respect to x_i. Otherwise, α includes in the body more than one **change type of subject** x_i. Now we extract from the body of α every **change type of subject** x_i but the last one.

[**Transformation 2**] In this stage, we assume that the transformation 1 has already been applied to α. Let us assume that with respect to x_i, the body of α now contains **create subject** x_i **of type** l_i and **change type of subject** x_i **to** l'_i (if it is not the case, Transformation 2 has no effect with respect to x_i). Now we extract **change type of subject** x_i **to** l'_i from the body and transform **create subject** x_i **of type** l_i into **create subject** x_i **of type** l'_i. Furthermore, we replace the type of formal parameter x_i of α with l'_i. As a result, we have $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_i : l'_i, \ldots, x_k : l_k)$ instead of the original α.

■

Transformations 1 and 2 *optimize* commands with respect to **change type**, i.e., take the net effects of the sequences of the primitive operations. So the following theorem is rather obvious:

Theorem 1: Given any command $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ and protection state Q, if α can be run on Q, and Q changes into a state Q' by executing α, then command $\alpha'(x_1 : l'_1, x_2 : l'_2, \ldots, x_k : l'_k)$, which is the normalization of α, can also be run on Q, and Q changes into Q' by executing α'.

[**Proof**] For the sake of brevity, we assume that every formal parameter x_i of α is a subject. If it is a pure object, we can prove the theorem in the same way.

Concerning Transformation 1, for each x_i, every **change type of subject** x_i in the body of α has no effect on the execution of other primitive operations in the body. Thus, only the last **change type of subject** x_i is significant and the results of execution of α and that of α' on Q are the same.

Now notice that for each x_i, there is at most one **create subject** x_i in the body of α because no subject can be created repeatedly as the identical one (see also Sect. 3.1). Furthermore, before **create subject** x_i, there must exist no primitive operation which accesses x_i, i.e., **enter/delete** for an element of A corresponding to x_i, **change type of subject** x_i, and **create/destroy subject** x_i. Therefore, Transformation 2 does not cause any difference between the results of execution of α and that of α', but possibly does between the formal parameters of x_i in α and in α'. However, the latter difference is not significant in type checking of formal and actual parameters in α and α' since the actual parameter subject corresponding to x_i does not exist until α (or α') is executed on Q.

Finally, recall that Transformations 1 and 2 do not cause any change in the condition part of α. As a result, if the condition of α holds true on Q, then so does the condition of α'. This completes the proof. □

By Theorem 1, we can easily show the next corollary:

Corollary 1: A set of reachable states from the initial state of a protection system does not change even if all commands in the command set of the system are normalized.

The most important result of Theorem 1 (or Corollary 1) is that we have only to consider commands each of which contains at most one **change type** operation with respect to each formal parameter. This makes the following safety analysis easier. Hence hereafter we assume that all commands are normalized unless otherwise explicitly stated.

Now we introduce a type relationship (TR) graph for safety analysis of the DTAM model. For that purpose, first, we define parent-type relationships between types.

Definition 12: If the body of $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ has **create subject** x_i **of type** l_i or **create object** x_i **of type** l_i ($1 \leq i \leq k$), then we define l_i as a *child type with respect to* ***create*** *in* α. If l_i is not a child type with respect to **create** in α, then l_i is said to be a *parent type with respect to* ***create*** *in* α. In particular, if every l_i ($1 \leq i \leq k$) is a child type with respect to **create**, all l_i are said to be *orphan types.* ■

Definition 13:
- If the body of $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ has **change type of subject** x_i **to** l'_i or **change type of object** x_i **to** l'_i ($1 \leq i \leq k$), then l'_i is said to be a *child type with respect to* ***change type*** *in* α and l_i is said to be a *parent type with respect to* ***change type*** *in* α.
- If the body of $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ has neither **change type of subject** x_i nor **change type of object** x_i and l_i is a parent type with respect to **create** in α ($1 \leq i \leq k$), then l_i is said to be a *parent type with respect to* ***change type*** *in* α as well as a *child type with respect to* ***change type*** *in* α. In this case, the types of the parent and the child are the same.

■

In order to demonstrate what parent-child relationships between types are like, let us consider the following three commands α_1, α_2, and α_3:

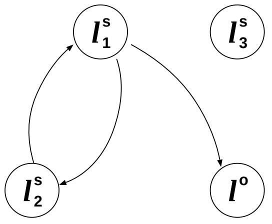

Fig. 1 Example of TR graph

```
command $\alpha_1(x_1 : l^s_1, x_2 : l^o)$
  create object $x_2$ of type $l^o$
  change type of subject $x_1$ to $l^s_2$
end

command $\alpha_2(x : l^s_2)$
  change type of subject $x$ to $l^s_1$
end

command $\alpha_3(x : l^s_3)$
  create subject $x$ of type $l^s_3$
end
```

First let us consider α_1. l^s_1 is a parent type with respect to **create** in α_1 and a parent type with respect to **change type** in α_1. Also l^s_2 is a child type with respect to **change type** in α_1. l^o is a child type with respect to **create** in α_1. In α_2, l^s_2 is a parent type with respect to **create** in α_2 and a parent type with respect to **change type** in α_2. Furthermore, l^s_1 is a child type with respect to **change type** in α_2. Concerning α_3, we see that l^s_3 is an orphan type. It is evident from α_3 that any command that has an orphan type must be unconditional and we can execute the command on any protection state. In consequence, we can create objects of an orphan type infinitely.

Now we are ready to define a type relationship (TR).

Definition 14: A *type relationship (TR) graph* $RG = (V_R, E_R)$ is a directed graph defined as follows:

- V_R is a set of vertices and $V_R = L$.
- E_R ($\subseteq V_R \times V_R$) is a set of edges and for each pair of v_1, $v_2 \in V_R$, an edge from v_1 to v_2 exists in E_R if and only if either of the following two conditions holds:
  - for some command α, v_1 is a parent type with respect to **create** in α, and v_2 is a child type with respect to **create** in α, or
  - for some command α, v_1 is a parent type with respect to **change type** in α, and v_2 is a child type with respect to **change type** in α.

■

For example, we show in Fig. 1 the TR graph for the three commands α_1, α_2, α_3 in this section .

4.2 Safety Analysis of Nonmonotonic Protection Systems (I)

Let us again consider the TR graph depicted in Fig. 1. Furthermore, we assume that a subject s of type l^s_1 exists in a state. Now we can create from the state an infinite number of pure objects $o_1, o_2, \ldots$, by executing $\alpha_1(s, o_1)$, $\alpha_2(s)$, $\alpha_1(s, o_2)$, $\alpha_2(s)$, ... In addition, as stated in Sect. 4.1, we can create objects of an orphan type infinitely. In summary, the existence of cycles[†] and orphan types in a TR graph is closely related to whether or not the number of objects in a protection system is finite, which in turn heavily influences the decidability of the safety problem as mentioned in Sect. 1.

In this section, we shall show that the DTAM model can describe nonmonotonic systems for which the safety problem is decidable.

First of all, we define creating commands [18] and parent-child relationships between objects.

Definition 15: If command α contains **create subject** or **create object** operations in its body, we say that α is a *creating command*, otherwise, it is a *noncreating command.* ■

Definition 16: If command $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ can be executed by substituting $o_1, o_2, \ldots, o_k$ for $x_1, x_2, \ldots, x_k$ and the execution creates some new objects, then we say that o_i $(1 \le i \le k)$ is a *parent* if l_i is a parent type with respect to **create** in α, otherwise that o_i is a *child.* A *descendant* of object o is recursively defined as o itself or a child of a descendant of o. If object o_1 is a descendant of object o_2, o_2 is said to be an *ancestor* of o_1. ■

Note that even a pure object can be a parent of other objects by definition.

Now we can prove the following lemma:

Lemma 1: Suppose a TR graph has no cycle that contains parent types with respect to **create** in creating commands. In such a case, given any creating command $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$, if l_i $(1 \le i \le k)$ is a parent type with respect to **create** in α, then α must have **change type of subject** x_i **to** $l_i{}'$ or **change type of object** x_i **to** $l_i{}'$ in its body such that $l_i \ne l_i{}'$.

[Proof] Suppose that for some creating command $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ and some i $(1 \le i \le k)$, l_i is a parent type with respect to **create** in α and α does not have **change type of subject** x_i **to** $l_i{}'$ or **change type of object** x_i **to** $l_i{}'$ in its body such that $l_i \ne l_i{}'$. In that case, l_i is a parent type as well as a child type with respect to **change type** by Def. 13. Consequently, the TR graph must contain at least one self-loop with vertex l_i by Def. 14. This is a contradiction. □

[†]Throughout this paper, we regard a (self-)loop as a special case of cycles, i.e., a cycle of length one.

Lemma 1 means that the execution of a creating command α must change the type of every actual parameter (object) into another type if the type of the corresponding formal parameter is a parent type with respect to **create** in α. However, the converse of Lemma 1 is not true.

Using Lemma 1, we can prove Lemma 2:

Lemma 2: The number of objects in arbitrary protection state of a protection system has an upper bound, provided that:

1. the authorization scheme of the system has no orphan type, and
2. the TR graph of the system has no cycle that contains parent types with respect to **create** in creating commands.

[Proof] Since there are no orphan types, every child type with respect to **create** has the corresponding parent type with respect to **create**. Therefore, every object in the system is a descendant of an object in the initial state $Q_0 = (S_0, O_0, A_0, f_{L0})$.

For command α, let $CR(\alpha)$ be the total number of **create subject** and **create object** operations in the body of α. Furthermore, let CR_{max} be the maximum value of $CR(\alpha)$ ($\alpha \in CM$). By Def. 8, CR_{max} is finite.

Now, with respect to some object o in a protection state, let us consider the number of descendants of o (see Fig. 2).[†]

First, we consider the maximum number of direct children of o. We see that only at most $|L|-1$ times we can execute creating commands with o as input. The reason for this is as follows. If we can execute some creating command α with o as its actual parameter, then the type of the corresponding formal parameter must always be a parent type with respect to **create** in α since o is already existent. Therefore, by Lemma 1, a type of o must be changed to another type after α with o is executed. So if we can execute such creating commands more than $|L|-1$ times, then in the execution sequence of the commands, at least two types assigned to o must be the same. However, this implies that in the TR graph, there exists a cycle that contains several types assigned to o, all of which are parent types with respect to **create** in creating commands. This contradicts the assumption 2 of Lemma 2. Therefore, the number that creating commands with o as input can be executed is at most $|L|-1$ and as a consequence, the number of direct children of o during the lifetime of the system is given by at most $CR_{max} \times (|L|-1)$.

Next, we discuss the maximum number of generations of descendants of o. The number of the generations is less than or equal to $|L|$. The reason is that, if it is greater than $|L|$, in descendants of o there exist two objects that are of the same type. This also implies that the TR graph has a cycle with parent types with respect to **create** and causes a contradiction.

[†]Note that Fig. 2 is *not* a TR graph. Please do not be confused in the following discussion.

From the discussions above, an upper bound of the number of descendants of an object is given by:

$$\begin{aligned}
&1 + (CR_{max}(|L|-1)) + (CR_{max}(|L|-1))^2 \\
&\quad + \cdots + (CR_{max}(|L|-1))^{|L|-1} \\
&= \begin{cases} \dfrac{(CR_{max}(|L|-1))^{|L|}-1}{CR_{max}(|L|-1)-1} & \cdots \text{where } CR_{max}(|L|-1) > 1 \\[2ex] |L| & \cdots \text{ where } CR_{max}(|L|-1) = 1 \\[2ex] 1 & \cdots \text{ where } CR_{max}(|L|-1) = 0. \end{cases}
\end{aligned}$$

Consequently, the number of objects in arbitrary protection state of the protection system has an upper bound O_{max}, which is given by:

$$O_{max} = \begin{cases} |O_0| \dfrac{(CR_{max}(|L|-1))^{|L|}-1}{CR_{max}(|L|-1)-1} \\ \qquad \cdots \text{ where } CR_{max}(|L|-1) > 1 \\[2ex] |O_0||L| \\ \qquad \cdots \text{ where } CR_{max}(|L|-1) = 1 \\[2ex] |O_0| \\ \qquad \cdots \text{ where } CR_{max}(|L|-1) = 0. \end{cases}$$

□

From Lemma 2, we can derive Theorem 2:

Theorem 2: The safety problem for protection systems is decidable, provided that:

1. the authorization schemes of the systems have no orphan type, and
2. the TR graphs of the systems have no cycle that contains parent types with respect to **create** in creating commands.

[Proof] By Lemma 2, the number of objects in arbitrary protection states of the systems in Theorem 2 is finite. This implies that the number of distinct protection states of such a system is also finite, which is proved as follows.

Let n_s and n_o denote the numbers of subjects and objects, respectively. Then the access matrix A has n_s rows and n_o columns and can express at most $(2^{|R|})^{n_s n_o}$ distinct states of authorization since each element of A can have at most $2^{|R|}$ distinct states. In regard to f_L, the maximum number of ways in which f_L maps objects to object types is given by:

$$\begin{cases} |L_S|^{n_s}(|L|-|L_S|)^{n_o-n_s} & \text{if } |L_S| < |L| \\[2ex] |L_S|^{n_s} & \text{otherwise.} \\ & \text{(i.e., } |L| = |L_S|) \end{cases}$$

From the discussions above, an upper bound of the number of distinct protection states of the system is

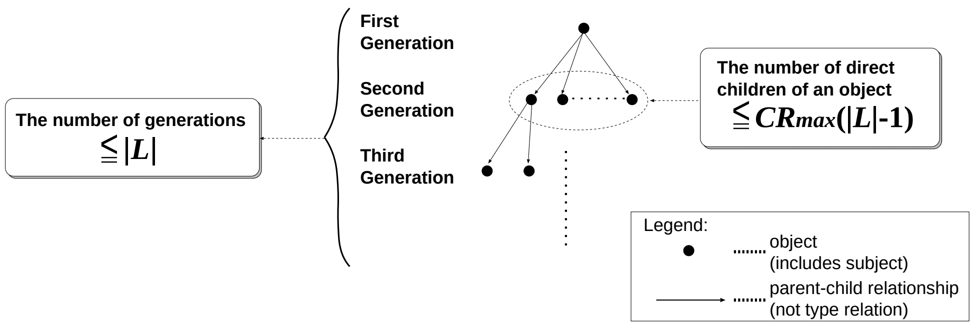
First Generation
Second Generation
Third Generation
The number of generations ≦|L|
The number of direct children of an object ≦CRmax(|L|-1)
Legend:
object (includes subject)
parent-child relationship (not type relation)

Fig. 2 Descendants of an object

given by (recall that a protection state is defined by a four-tuple (S, O, A, f_L)):

$$\begin{cases} \displaystyle\sum_{n_o=0}^{O_{max}} \sum_{n_s=0}^{n_o} \left\{ \binom{O_{max}}{n_s} \binom{O_{max}-n_s}{n_o-n_s} \times 2^{|R|n_s n_o} |L_S|^{n_s} (|L|-|L_S|)^{n_o-n_s} \right\} & \cdots \text{ if } |L_S| < |L| \\ \displaystyle\sum_{n_s=0}^{O_{max}} \left\{ \binom{O_{max}}{n_s} 2^{|R|{n_s}^2} |L_S|^{n_s} \right\} & \cdots \text{ otherwise.} \end{cases}$$

In other words, the number of different states is finite. Therefore, we can check whether or not a particular subject has a particular right for a particular object in every reachable state from the initial state by using, say, depth-first search. □

By the proof above, we see that whenever the conditions given in Theorem 2 are satisfied, the safety problem is decidable regardless of the kinds of primitive operations in command bodies. Namely, Theorem 2 shows the existence of new nonmonotonic systems where the safety problem is decidable.

4.3 Safety Analysis of Nonmonotonic Protection Systems (II)

In this section, we again discuss the safety problem for nonmonotonic systems in Theorem 2, but with further restriction that they have no creating commands.

Theorem 3: The safety problem is NP-hard for protection systems, provided that:

1. the authorization schemes of the systems have no creating command,[†] and
2. the TR graphs of the systems have no cycle.

[**Proof**] First we present the subset sum problem [6]:

Given a finite set M, a size function $w(m) \in Z^+$ for each $m \in M$, positive integer N. Is there a subset $M' \subseteq M$ such that the sum of the sizes of the elements in M' is exactly N?

The subset sum problem is known to be NP-complete. Hereafter, we assume that $M = \{m_1, m_2, \ldots, m_n\}$ and $\sum_{i=1}^{n} w(m_i) = I$. Furthermore, let $w_1, w_2, \ldots, w_k$ be the set of distinct values of $w(m_1), w(m_2), \ldots, w(m_n)$. Without loss of generality, we assume that $w(m_1) \leq w(m_2) \leq \ldots \leq w(m_n)$ and $w_1 < w_2 < \ldots < w_k$. This implies that $1 \leq w(m_1) = w_1$.

Given this subset sum problem and a protection system that satisfies the conditions in Theorem 3, we run the *authorization scheme construction algorithm* (AC algorithm for short), which is given in Fig. 3. Two **while** statements in the figure ((1) and (6)) compute L_S and CM, respectively, and the commands $\alpha_{N,end}$ and $\alpha_{i,j}$ (i and j are variables) are defined as follows:

```
command $\alpha_{N,end}(x : l^s_N)$
  enter $r$ into $A[x,x]$
  change type of subject $x$ to $l^s_{end}$
end

command $\alpha_{i,j}(x_1 : l^s_i, x_2 : l^o_j)$
  change type of subject $x_1$ to $l^s_{i+w(m_j)}$
  change type of object $x_2$ to $l^o_{end}$
end
```

For example, let us consider the case that $M = \{m_1, m_2, m_3\}$, $w(m_1) = 2$, $w(m_2) = 3$, $w(m_3) = 3$. Shown in Fig. 4 is the TR graph of the authorization scheme generated by the AC algorithm. For the sake of simplicity, the parts of the graph for types of pure objects and parent-child relationships in command $\alpha_{N,end}$ are not drawn in Fig. 4.

Now we consider whether the AC algorithm eventually stops or not. Regarding **while** statement (1) of the AC algorithm, l^s_i whose subscript i is the smallest is removed from C in step (2) and $l^s_{i+w_j}$ is added to C in step (5). However, since the condition in step (4) ensures that a subscript of each element of C is not greater than I, C becomes empty eventually and **while** statement (1) surely terminates. With respect to **while** statement (6), it also stops in the end. The computational complexity of AC algorithm is $O(In)$.

[†]Such protection systems do not have orphan types.

```
$L_S \leftarrow \{l_0^s\}$; $C \leftarrow \{l_0^s\}$;
while $C \neq \phi$ do                                          /* (1) */
  begin
    From $C = \{l_{i_1}^s, l_{i_2}^s, \ldots, l_{i_l}^s\}$, choose $l_{i_a}^s$ whose
          subscript $i_a$ is the smallest of $\{i_1, i_2, \ldots, i_l\}$
          and set $i \leftarrow i_a$.
    $C \leftarrow C - \{l_i^s\}$;                                /* (2) */
    for $j \leftarrow 1$ to $k$ do                              /* (3) */
      begin
        if $i + w_j \leq I$ then                                /* (4) */
          begin
            $L_S \leftarrow L_S \cup \{l_{i+w_j}^s\}$;
            $C \leftarrow C \cup \{l_{i+w_j}^s\}$;              /* (5) */
          end
        else
          goto END1
      end;
END1:
  end;

$CM \leftarrow \{\alpha_{N,end}\}$; $C \leftarrow L_S$;
while $C \neq \phi$ do                                          /* (6) */
  begin
    From $C = \{l_{i_1}^s, l_{i_2}^s, \ldots, l_{i_l}^s\}$, choose $l_{i_a}^s$ whose
          subscript $i_a$ is the smallest of $\{i_1, i_2, \ldots, i_l\}$
          and set $i \leftarrow i_a$.
    $C \leftarrow C - \{l_i^s\}$;
    for $j \leftarrow 1$ to $n$ do
      begin
        if $i + w(m_j) \leq I$ then
          $CM \leftarrow CM \cup \{\alpha_{i,j}\}$
        else
          goto END2
      end;
END2:
  end;
$R \leftarrow \{r\}$; $L_S \leftarrow L_S \cup \{l_N^s, l_{end}^s\}$;
$L \leftarrow L_S \cup \{l_1^o, l_2^o, \ldots, l_n^o, l_{end}^o\}$;
```

Fig. 3 Algorithm for authorization scheme construction

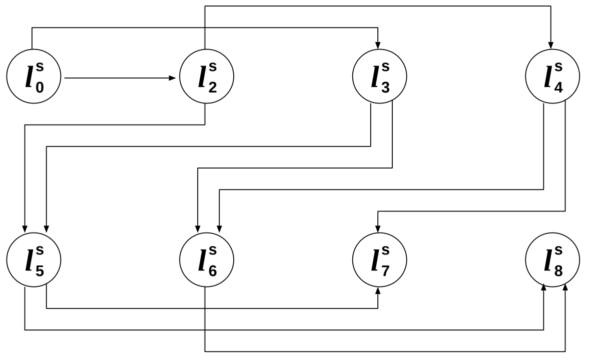

Fig. 4 TR graph generated by AC algorithm

We are now in a position to consider the polynomial-time reducibility from the subset sum problem to the safety problem in Theorem 3. For that purpose, we consider the protection system $\mathcal{P}$, which has the authorization scheme generated by the AC algorithm. The initial state of $\mathcal{P}$, (S_0, O_0, A_0, f_{L0}), is given in Fig. 5. In the rest of this section we reduce the subset sum problem to the safety problem for $\mathcal{P}$, which

- $S_0 = \{s\}$
- $O_0 = S_0 \cup \{o_1, o_2, \ldots, o_n\}$
- For every pair of $s \in S_0$ and $o \in O_0$, $A_0[s, o] = \phi$
- $f_{L0}(s) = l_0^s$, $f_{L0}(o_i) = l_i^o$ $(1 \leq i \leq n)$

Fig. 5 Initial state of $\mathcal{P}$

is a restricted case of the safety problem in Theorem 3.

Let us assume that the subset sum problem has a solution $M' = \{m_{j_1}, m_{j_2}, \ldots, m_{j_l}\}$. That is, $M' \subseteq M$ and $w(m_{j_1}) + w(m_{j_2}) + \ldots + w(m_{j_l}) = N$. In such a case, for subject s and pure objects o_{j_1}, o_{j_1}, $\ldots$, o_{j_l}, we can execute commands $\alpha_{0,j_1}(s, o_{j_1})$, $\alpha_{w(m_{j_1}),j_2}(s, o_{j_2})$, $\alpha_{w(m_{j_1})+w(m_{j_2}),j_3}(s, o_{j_3})$, $\ldots$, and finally $\alpha_{w(m_{j_1})+w(m_{j_2})+\ldots+w(m_{j_{l-1}}),j_l}(s, o_{j_l})$ one by one. According to the execution, the type of s changes from l_0^s to $l_{w(m_{j_1})}^s$, $l_{w(m_{j_1})+w(m_{j_2})}^s$, $\ldots$, $l_{w(m_{j_1})+w(m_{j_2})+\ldots+w(m_{j_l})}^s = l_N^s$ in turn. Finally, for subject s, we can invoke command $\alpha_{N,end}(s)$, and s acquires the privilege r. If the subset sum problem has no solution, the type of s can never be l_N^s and r is not granted to s.

On the other hand, suppose that s can possess the privilege r in $\mathcal{P}$. Namely, for s, o_{j_1}, o_{j_2}, $\ldots$, o_{j_l}, we can run commands $\alpha_{a_0,j_1}(s, o_{j_1})$, $\alpha_{a_1,j_2}(s, o_{j_2})$, $\ldots$, $\alpha_{a_{l-1},j_l}(s, o_{j_l})$ in this order and the type of s is changed into $l_{a_l}^s = l_N^s$. Finally, we can execute $\alpha_{N,end}(s)$ and s gets r. Note that $0 = a_0 < a_1 < \ldots < a_l = N$ holds.

At this time, let us assume that $\alpha_{0,i}$ $(1 \leq i \leq n)$ takes the following form:

command $\alpha_{0,i}(x_1 : l_0^s, x_2 : l_i^o)$
  change type of subject x_1 **to** $l_{z_i}^s$
  change type of object x_2 **to** l_{end}^o
end.

Thus we have $z_1, z_2, \ldots, z_n$ and for every i $(1 \leq i \leq n)$, we set $w(m_i)$ to z_i. Consequently, we have $M = \{m_1, m_2, \ldots, m_n\}$, $w(m_1)$, $w(m_2)$, $\ldots$, $w(m_n)$.

Now, by the authorization scheme of $\mathcal{P}$, we see that for every b $(1 \leq b \leq l)$, $a_b - a_{b-1} = w(m_{j_b})$ holds and by adding each side of these equations we obtain:

$$N = w(m_{j_1}) + w(m_{j_2}) + \ldots + w(m_{j_l}).$$

Then $M' = \{m_{j_1}, m_{j_2}, \ldots, m_{j_l}\}$ is exactly a solution of the subset sum problem. □

5. Discussion

The reason for the decidability of Theorem 2 can be informally summarized as follows: recall that the TR graphs have no cycle that contains parent types with respect to **create** in creating commands in the theorem. So if the current type of o is a parent type with respect to **create** in α, the execution of creating command α must change the type of every actual parameter object o of α into a type that o has never experienced. To put it in another way, creating commands make 'irreversible' changes on types of parent objects. It is this irreversibility in creating objects that makes the safety analysis decidable. The type change in creating objects is irreversible, so that the number of times such type changes occurs is finite since the total number of types is finite by assumption. In consequence, the number of objects is finite (Lemma 2) and the safety problem becomes decidable.

In nonmonotonic systems in Theorem 2, it is possible that the systems reach a state where we can no longer create new objects. However, generally speaking, it is a good practice to reevaluate security policies continuously [1], so the state could be a possible candidate point of time for such reevaluation.

Although the number of objects in the systems of Theorem 2 is finite, because the TR graphs can have cycles as long as the cycles do not contain parent types with respect to **create** in creating commands, it is possible to execute commands infinite times in such systems. In that case, the lifetimes of the systems are infinite.

Finally, it should be noted again that the safety problem is generally undecidable and most decidable safety cases in previous work are for monotonic systems. On the other hand, the decidable cases in Theorem 2 and in Theorem 3 are for nonmonotonic systems and fall outside the known decidable ones. In particular, we have shown that the safety problem in Theorem 3 belongs to a well-known complexity class, namely, NP-hard. However, in practice, safety analysis is intractable unless it has polynomial time complexity. Thus, further research is needed for nonmonotonic systems where the safety analysis is decidable in polynomial time.

6. Conclusion

In this paper, we have proposed the Dynamic-Typed Access Matrix (DTAM) Model, which extends the TAM model by allowing the type of an object to change dynamically. The DTAM model has an advantage that it can describe non-monotonic protection systems where the safety problem is decidable. In order to show this, first we have introduced a type relationship (TR) graph, with which we express both parent-child and transition relationships among types. Next we have shown that the safety problem becomes decidable in a non-monotonic system, provided that some restrictions are imposed on it. Moreover, we have shown that the safety problem becomes NP-hard when no new entities are permitted to be created. The decidable safety cases discussed in this paper fall outside the known decidable ones in previous work.

References

[1] D. Bailey. A philosophy of security management. In M. D. Abrams, S. Jajodia, and H. J. Podell eds., *Information Security: An Integrated Collection of Essays*, pp. 98–110. IEEE Computer Society Press, 1995.

[2] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report ESD-TR-73-278-I, MITRE Corp., Bedford, MA, Mar. 1973.

[3] D. E. R. Denning. *Cryptography and Data Security.* Addison-Wesley Publishing Co., Reading, MA, 1982.

[4] S. N. Foley, L. Gong, and X. Qian. A security model of dynamic labeling providing a tiered approach to verification. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 142–153, 1996.

[5] S. Ganta. *Expressive Power of Access Control Models Based on Propagation of Rights.* PhD thesis, Laboratory for Information Security Technology (LIST), George Mason University, 1996.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-completeness.* W. H. Freeman and Co., 1979.

[7] M. A. Harrison and W. L. Ruzzo. Monotonic protection systems. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton eds., *Foundations of Secure Computations*, pp. 337–365. Academic Press, 1978.

[8] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, Aug. 1976.

[9] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Inc., 1991.

[10] International Organization of Standardization (ISO). International standard ISO/IEC 15408, 1999. Technically identical to Common Criteria version 2.1.

[11] A. K. Jones, R. J. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *17th Annual Symposium on Foundations of Computer Science*, pp. 33–41, Houston, Texas, Oct. 1976.

[12] C. E. Landwehr. Protection (security) models and policy. In A. B. Tucker, Jr. ed., *The Computer Science and Engineering Handbook*, chapter 90, pp. 1914–1928. CRC Press, Boca Raton, FL, 1997.

[13] M. Maekawa, A. E. Oldehoeft, and R. R. Oldehoeft. *Operating Systems – Advanced Concepts.* The Benjamin/Cummings Publishing Company, Inc., 1987.

[14] J. McLean. Security models. In J. J. Marciniak ed., *Encyclopedia of Software Engineering*, Vol. 2, pp. 1136–1145. John Wiley & Sons, Inc., 1994.

[15] C. Meadows. Policies for dynamic upgrading. In S. Jajodia and C. E. Landwehr eds., *Database Security, IV: Status and Prospects*, pp. 241–250. Elsevier Science Publishers B. V (North-Holland), 1991.

[16] R. S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating schemes. *J. ACM*, 35(2):404–432, Apr. 1988.

[17] R. S. Sandhu. Expressive power of the schematic protection model. *J. Comput. Security*, 1(1):59–98, 1992.

[18] R. S. Sandhu. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 122–136, May 1992.

[19] R. S. Sandhu. Undecidability of the safety problem for the schematic protection model with cyclic creates. *J. Comput. Syst. Sci.*, 44(1):141–159, Feb. 1992.

[20] R. S. Sandhu and S. Ganta. Expressive power of the Single-Object Typed Access Matrix Model. In *Computer Security Applications Conference*, pp. 184–194, Dec. 1993.

[21] R. S. Sandhu and G. S. Suri. Non-monotonic transformation of access rights. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 148–161, May 1992.

[22] L. Snyder. Formal models of capability-based protection systems. *IEEE Trans. Comput.*, C-30(3):172–181, Mar. 1981.

[23] M. Soshi. *Design, Safety Analysis, and Applications of the Access Control Model based on Dual Labels.* PhD thesis, Graduate School of Information Systems, University of Electro-Communications, Mar. 1999. (In Japanese).

[24] M. Soshi. Safety analysis of the dynamic-typed access matrix model. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner eds., *Computer Security – ESORICS 2000: 6th European Symposium on Research in Computer Security*, Vol. 1895 of *Lecture Notes in Computer Science*, pp. 106–121. Springer-Verlag, Oct. 2000.

[25] M. Soshi, T. Kato, and M. Maekawa. An access control model based on dual labels. *Transactions of Information Processing Society of Japan*, 40(3):1305–1314, Mar. 1999. (In Japanese).

Appendix A: Example: a Simplified Multilevel Policy

In this appendix, we express a practical security policy example for the DTAM model and show how our proposed scheme in this paper is applicable to it. Due to space constraints, unfortunately, the example we give in this appendix cannot but be relatively small. For more practical security policies and thorough discussion on them, consult [23], [25][†].

Note that the proposed scheme in this paper is one of the most general ones with decidable safety properties and can describe nonmonotonic systems. The restriction imposed on the proposed scheme is, informally speaking, that it has an upper bound of the number of objects. However, this is a very natural restriction found in almost all practical security systems. For example, ISO defines the international standard for security system evaluation, i.e., ISO 15408 [10]. The standard prescribes the maximum quota enforcement of several resources. Hence, the proposed scheme could express many, or almost all of, practical security policies[††], or on the contrary, it would be easy to modify existing practical policies such that they conform to the proposed scheme.

Now we present the example scheme. In short, it is a simplified multilevel security policy, but at the same time, it contains various factors that are commonly found in many security policies [2]–[4], [8], [11]–[15]. Furthermore, by using the techniques demonstrated here, it is also easy to express other practical security policies. The factors of the example policy are listed below:

- owner-based policy,
- object creation,
- transfer of privileges between subjects,
- security levels of subjects and objects,
- a simplified multilevel security policy,
- downgrading and sanitization, which involve **change type** operations in an essential sense and will be discussed in more details later, and
- TR graph according to the proposed scheme.

The details of the scheme are given below.

First, we define the authorization scheme (L_S, L, R, CM), each element of which is given as follows:

- $L_S = \{l^{s,init}_{high}, l^s_{high}, l^{s,init}_{low}, l^s_{low}, l^s_{security\_officer}, l^f_{high}, l^f_{low}, l^f_{high\_to\_low}\}$,
- $L = L_S$,
- $R = \{$own, read, write, seek_sanitize, sanitized$\}$,
- $CM = \{$create_file_high, create_file_low, confer_write_high, confer_write_low, confer_read_high, confer_read_low, confer_read_high_low, downgrade, sanitize, finish_sanitize, confer_read_high_to_low, confer_read_sanitized $\}$

Note that all types of this authorization scheme belong to L_S, i.e., subject types, for simplicity and convenience. The reason will be clarified when we discuss the commands finish_sanitize and confer_read_sanitized later.

Below we explain each command in CM. First, we introduce two creating commands, i.e., create_file_high and create_file_low as follows:

```
command create_file_high($user{:}l^{s,init}_{high}$, $file{:}l^f_{high}$)
    create subject $file$ of type $l^f_{high}$
    enter own into $A[user, file]$
    change type of subject $user$ to $l^s_{high}$
end
```

```
command create_file_low($user : l^{s,init}_{low}$, $file : l^f_{low}$)
    create subject $file$ of type $l^f_{low}$
    enter own into $A[user, file]$
    change type of subject $user$ to $l^s_{low}$
end
```

create_file_high and create_file_low are unconditional creating commands. In create_file_high, a user of type $l^{s,init}_{high}$ can create a file of type l^f_{high} and becomes the owner of the file. Furthermore, the type of the user

[†]These practical policies are expressed by the *dual label model* [23], but it is really straightforward to rewrite them on the DTAM model. [23] also includes examples of HRU, TAM, take-grant model, Bell-LaPadula Model, and so on, described by the dual label model.

[††]The fact that the DTAM model has a rich expressive power can be illustrated as follows. First recall that HRU can perfectly simulate any Turing machine. This is shown by the way of the proof in [8]. Hence, HRU is supposed to express any realistic security policies enforced on current computer systems. TAM and DTAM models are HRU variants and supersets of HRU. Furthermore, note that DTAM is a (proper) superset of TAM. This fact is evidence of the rich expressive power of the DTAM model. For more details of the expressive power of the TAM model, see [5], [17], [18], [20], and for DTAM, see [23], [25].

changes to l^s_{high}. This ensures that the scheme here conforms to the proposed scheme, namely, the TR graph has no cycle that contains parent types with respect to **create** in creating commands. The scheme also has no orphan type, which will become clear later.

With respect to create_file_low, note that a similar discussion also holds.

Next, we discuss authorization of write access for files. This is represented by the following commands:

```
command confer_write_high($user1 : l^s_{high}$,
        $user2 : l^s_{high}, file : l^f_{high}$)
  if own $\in A[user1, file]$
  then
    enter write into $A[user2, file]$
end
```

```
command confer_write_low($user1 : l^s_{low}, user2 : l^s_{low}$,
        $file : l^f_{low}$)
  if own $\in A[user1, file]$
  then
    enter write into $A[user2, file]$
end
```

If a user of type l^s_{high} (or l^s_{low}) is the owner of a file of type l^s_{high} (or l^s_{low}), then he can grant write access for the file to another user of type l^s_{high} (or l^s_{low}). Needless to say, these two users may be the same person.

Now we consider *downgrading* and *sanitization* in our policy[†]. In practical protection systems, it is often necessary that the security levels of objects are lowered (downgraded) because it is convenient or the objects are outdated. However, such level changes violate the multilevel security policy. To overcome the problem, usually security officers edit the sensitive parts of the objects (i.e., sanitization) and release them.

Command downgrade is now given as follows.

```
command downgrade($user1 : l^s_{high}$,
        $user2 : l^s_{security\_officer}, file : l^f_{high}$)
  if own $\in A[user1, file]$
    $\wedge$ write $\in A[user1, file]$
  then
    delete own from $A[user1, file]$
    delete write from $A[user1, file]$
    enter seek_sanitize into $A[user2, file]$
    change type of object $file$ to $l^f_{high\_to\_low}$
end
```

If a user of type l^s_{high} is the owner of a file of type l^f_{high} and is authorized to exercise write rights on the file, then he can lower the security level of the file. But since this violates the policy, a security officer must sanitize the file before a user of type l^s_{low} can read it (read commands will be discussed later). For that purpose, first seek_sanitize right on the file is granted to the officer and the own and write rights of the original owner are revoked.

[†]For more details of downgrading and sanitization, refer to [3], [21], [23], [25].

Sanitization operation is represented by the sanitize command below.

```
command sanitize($user : l^s_{security\_officer}$,
        $file : l^f_{high\_to\_low}$)
  if seek_sanitize $\in A[user, file]$
  then
    enter read into $A[user, file]$
    enter write into $A[user, file]$
end
```

If a security officer has the seek_sanitize right to a file of type $l^f_{high\_to\_low}$, then by the discussion above it implies that he is asked to sanitize the file. This situation is described by the two **enter** statements in the sanitize command.

Completion of sanitization is expressed by the finish_sanitize command given below.

```
command finish_sanitize($user : l^s_{security\_officer}$,
        $file : l^f_{high\_to\_low}$)
  if seek_sanitize $\in A[user, file]$
  then
    delete seek_sanitize from $A[user, file]$
    delete write from $A[user, file]$
    enter sanitized into $A[file, file]$
end
```

Note that since we treat files as subjects (see also the authorization scheme), we can store their states in the access matrix in terms of access modes. In finish_sanitize above, this is exemplified by the primitive operation '**enter** sanitized **into** $A[file, file]$'. This technique was first introduced in [8].

Now we are in a position to show read commands below.

```
command confer_read_high($user : l^s_{high}, file : l^f_{high}$)
    enter read into $A[user, file]$
end
```

```
command confer_read_low($user : l^s_{low}, file : l^f_{low}$)
    enter read into $A[user, file]$
end
```

confer_read_high and confer_read_low are simple and intuitive. A user of type l^s_{high} (or l^s_{low}) can read a file of type l^f_{high} (or l^f_{low}).

```
command confer_read_high_low($user : l^s_{high}, file : l^f_{low}$)
    enter read into $A[user, file]$
end
```

Because we are considering a (simplified) multi-level policy, the objective of confer_read_high_low command is also obvious. That is, a user of type l^s_{high} is allowed to read a file of type l^f_{low}.

```
command confer_read_high_to_low($user : l^s_{high}$,
        $file : l^f_{high\_to\_low}$)
    enter read into $A[user, file]$
end
```

```
command confer_read_sanitized($user : l^s_{low}$,
        $file : l^f_{high\_to\_low}$)
  if sanitized $\in A[file, file]$
  then
    enter read into $A[user, file]$
end
```

As we have discussed downgrading and sanitization, confer_read_high_to_low and confer_read_sanitized commands would also need a little explanation. If a file is of type $l^f_{high\_to\_low}$ and marked as sanitized, then it means that the file was of type l^f_{high} before, and that it has already been downgraded and sanitized by a security officer. Therefore it is assumed that it has now no sensitive information and a user of type l^s_{low} can read it. Needless to say, a user of type l^s_{high} can read it by command confer_read_high_to_low, even if it has not been sanitized yet.

The TR graph of the scheme reported here is drawn in Fig. A·1. Note that the authorization scheme given in this section is nonmonotonic and its TR graph actually follows the proposed scheme in this paper.

Now we consider the safety property of the described scheme. First of all, recall that the safety problem is undecidable in general as discussed in Sects. 1 and 2. Namely, systematic ways (i.e. algorithms) to analyze it do not exist. Thus, in order to consider the safety property, usually we must investigate every reachable state one by one. However, such an analysis of even small-scale systems often leads to "state explosion" [9] and fails (safety analysis is even more difficult than ordinary protocol analyses because we must consider the cases of infinite objects to be newly created). Of course, for our proposed scheme, such an investigation process eventually terminates (i.e., decidable). Therefore, in order to explain how the safety analysis is conducted, let us consider a simple protection system below:

- authorization scheme: (L_S, L, R, CM), which has been defined so far.
- an initial state $Q_0 = (S_0, O_0, A_0, f_{L_0})$:
  - $S_0 = \{u1, u2, u3, so\}$,
  - $O_0 = \{f\}$,
  - $A_0 : A_0[s, o] = \{own\}$ if $s = u2$ and $o = f$, $A_0[s, o] = \phi$ otherwise, and
  - f_{L_0}: $f_{L_0}(u1) = l^{s,init}_{high}$, $f_{L_0}(u2) = l^s_{high}$, $f_{L_0}(u3) = l^s_{low}$, $f_{L_0}(so) = l^s_{security\_officer}$, $f_{L_0}(f) = l^f_{high}$.

Now consider whether or not user $u3$ can obtain the read rights for file f. A part of state transitions of the protection system above is given in Fig.A·2. Observe that one state transition actually reaches a state where $u3$ can read f. The investigation process given here is really a simplification of actual safety analyses.

Masakazu Soshi He received his B.E. and M.S. degrees from the University of Tokyo in 1991 and in 1993, respectively, and his Ph.D. degree from the University of Electro-Communications in 1999. He worked as an associate for the University of Electro-Communications from 1997 to 1998 and for the Japan Advanced Institute of Science and Technology (JAIST) from 1999 to January 2003. Since February 2003, he has been a research associate professor of JAIST. His research interests include theoretical analysis of access matrix models, anonymous communication, and development of security architectures in general.

Mamoru Maekawa He received his BS and PhD degrees from Kyoto University and the University of Minnesota, respectively. He is currently Dean and Professor of the Graduate School of Information Systems, University of Electro-Communications, Tokyo, Japan. His research interests include distributed systems, operating systems, software engineering, and GIS. He is listed in many major Who's Who's.

Eiji Okamoto Eiji Okamoto received his B.S., M.S. and Ph.D degrees in electronics engineering from the Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. He worked and studied communication theory and cryptography for NEC central research laboratories since 1978. Then he became a professor at JAIST (Japan Advanced Institute of Science and Technology) from 1991, and at Toho University from 1999 until 2002. He is currently a professor at the Institute of Information Sciences and Electronics, University of Tsukuba. His research interests are cryptography and information security.

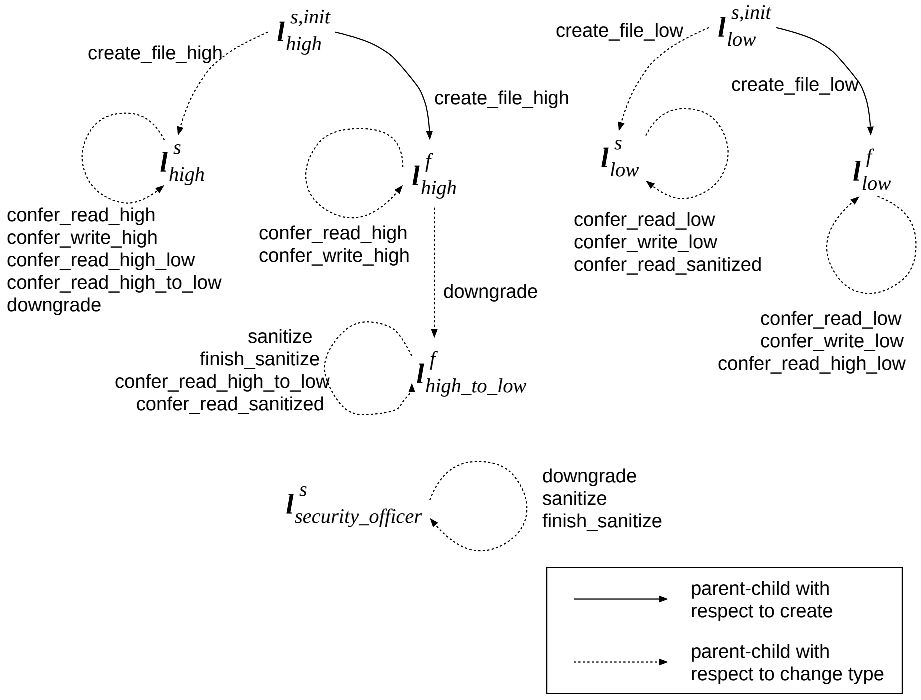

Fig. A· 1 TR graph

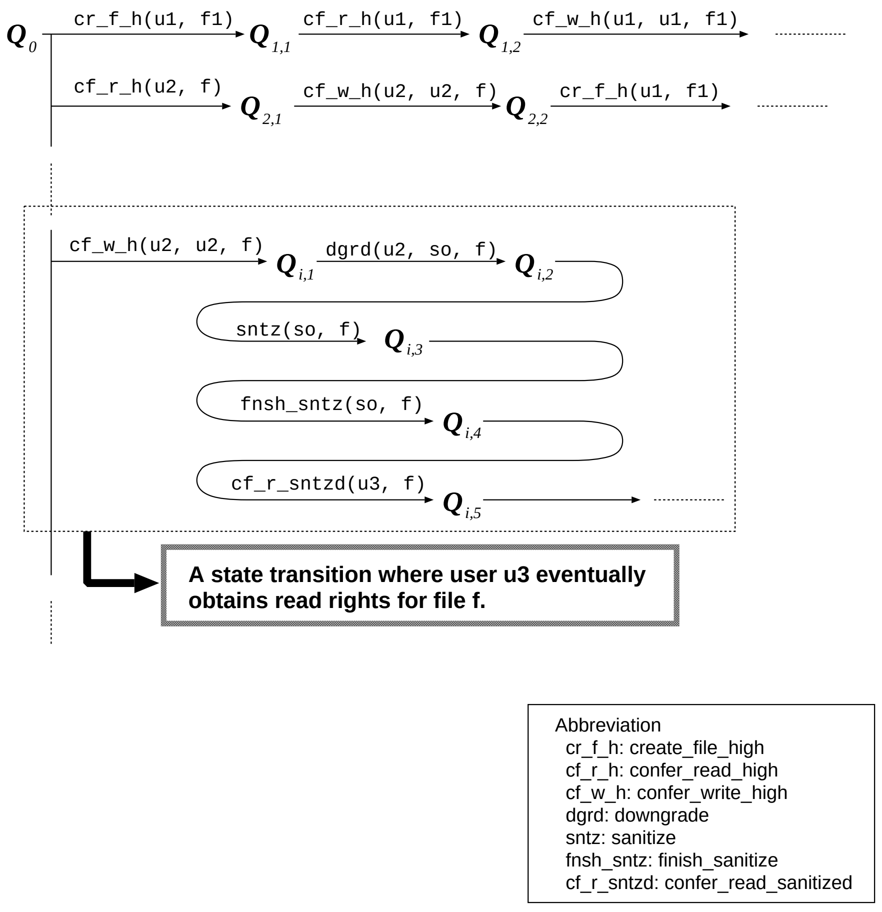

Fig. A· 2 State transitions