Safety Analysis of the Dynamic-Typed Access Matrix Model

Masakazu Soshi

School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi-machi, Nomi-gun, Ishikawa 923-1292, JAPAN
soshi@jaist.ac.jp

Abstract. The safety problem in access matrix models is the one to determine whether or not a given subject can eventually obtain an access privilege to a given object. Unfortunately, little 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 Model, which extends Typed Access Matrix model by allowing the type of an object to change dynamically. DTAM model has an advantage that it can describe non-monotonic protection systems for which the safety problem is decidable. In this paper, we formally define DTAM model and then discuss various aspects of it.

Keywords: 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*. In an access matrix model, a protection system is characterized by a collection 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 security property of protection systems in the access matrix model (HRU model) as the *safety problem* [4].[1] 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 [4, 7]. 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

[1] For lack of space, here we present some relevant work only briefly. See the excellent historical review in [6] for further details.

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 are allowed). For example, Sandhu developed the Typed Access Matrix (TAM) Model [6], 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 TAM model by allowing the type of an object to change dynamically. DTAM model has an advantage that it can describe non-monotonic 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 non-monotonic systems becomes decidable if, roughly speaking, the TR graphs of the systems have no cycle with respect to parent-child relationship between objects.[2] 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.

The remainder of the paper is structured as follows. We formalize DTAM model in Section 2 and make a thorough investigation of the safety property of DTAM model in Section 3. Section 4 discusses various topics on DTAM model and then finally we conclude this paper in Section 5.

2 Dynamic-Typed Access Matrix Model

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

2.1 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* [6].

Every object has its own *identity* inherently. Hence, for instance, no object can be created repeatedly as the 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 $1 \leq |L_S| \leq |L|$.

[2] The precise condition for this case will be given in Section 3.

For example, L_S may consist of three user types, *programmer*, *system-engineer*, and *project-manager*. In addition, we can take $file_{trade-secrets}$ and $file_{public}$ for 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 $f_S : S \rightarrow L_S$. Next we define the function which returns the type of a pure object as $f_O : (O - S) \rightarrow (L - L_S)$. Now we can define the *type function* $f_L : O \rightarrow L$, 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 are denoted by R.

Using Definition 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.

2.2 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 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 [2, 5]. For the example in Section 2.1, 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 3).

We shall define commands based on Definition 7.

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 $o' \in O'$

Table 1. DTAM primitive operations

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 $r_1 \in A[x_{k_{s1}}, x_{k_{o1}}] \wedge r_2 \in A[x_{k_{s2}}, x_{k_{o2}}] \wedge \ldots \quad \wedge r_m \in A[x_{k_{sm}}, x_{k_{om}}]$
  then op_1; op_2; ...; 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.

2.3 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 [6]:

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 non-monotonicity 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 *non-monotonic*. Furthermore, if the authorization scheme of a system is monotonic, the system is said to be monotonic, otherwise non-monotonic.

This completes the formalization of DTAM model.

3 Safety Analysis

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

3.1 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 the 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.

Transformation 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 12. *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 Section 2.1). 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 Transformation 1 and 2 does not make 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 12, we can easily show the next corollary:

Corollary 13. *Set of reachable states from the initial state of a protection system do not change even if all commands in the command set of the system are normalized.*

The most important result of Theorem 12 (or Corollary 13) 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 DTAM model. For that purpose, first we define parent-type relationships between types.

Definition 14. 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 15.
- 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:

command $\alpha_1(x_1 : l_1^s, x_2 : l^o)$
  create object x_2 **of type** l^o
  change type of subject x_1 **to** l_2^s
end

command $\alpha_2(x : l_2^s)$
  change type of subject x **to** l_1^s
end

command $\alpha_3(x : l_3^s)$
  create subject x **of type** l_3^s
end

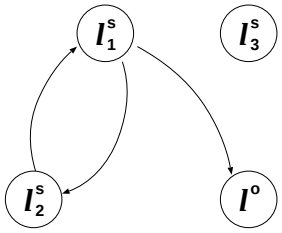

Fig. 1. Example of TR graph

First let us consider α_1. l_1^s 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_2^s 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_2^s 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_1^s is a child type with respect to **change type** in α_2. Concerning α_3, we see that l_3^s 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 16. 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, $v_2 \in V_R$, a 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 Figure 1 TR graph for the three commands α_1, α_2, α_3 in this section .

3.2 Safety Analysis of Non-monotonic Protection Systems (I)

Let us again consider TR graph depicted in Figure 1. Furthermore we assume that a subject s of type l_1^s exists in a state. Now we can create from the state an infinite number of pure objects o_1, o_2, ..., by executing $\alpha_1(s, o_1)$, $\alpha_2(s)$, $\alpha_1(s, o_2)$, $\alpha_2(s)$, ... In addition, as stated in Section 3.1, we can create objects of an orphan type infinitely. In summary, the existence of cycles[3] and orphan

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

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 Section 1.

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

First of all we define creating commands [6] and parent-child relationships between objects.

Definition 17. If command α contains **create subject** or **create object** operations in its body, we say that α is a *creating command*, otherwise a *non-creating command*.

Definition 18. 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 $(1 \leq i \leq k)$ 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 19. *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 $(1 \leq i \leq k)$ 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 $l_i \neq l_i'$.*

Proof. Suppose that for some creating command $\alpha(x_1 : l_1, x_2 : l_2, \ldots, x_k : l_k)$ and some i $(1 \leq i \leq k)$, 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 $l_i \neq l_i'$. In that case, l_i is a parent type as well as a child type with respect to **change type** by Definition 15. Consequently the TR graph must contain at least one self-loop with vertex l_i by Definition 16. This is a contradiction. □

Lemma 19 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 19 is not true.

Using Lemma 19, we can prove Lemma 20:

Lemma 20. *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.*

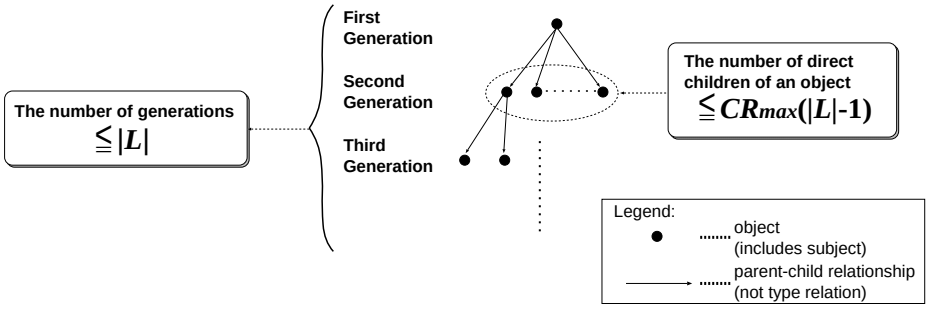

Fig. 2. Descendants of an object

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 Definition 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 Figure 2).[4]

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 be always a parent type with respect to **create** in α since o is already existent. Therefore, by Lemma 19, 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 20. 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

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

same type. This also implies that the TR graph has a cycle with parent types with respect to **create** and causes a contradiction.

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

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

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

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

□

From Lemma 20, we can derive Theorem 21:

Theorem 21. *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 20, the number of objects in arbitrary protection states of the systems in Theorem 21 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:

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

From the discussions above, an upper bound of the number of distinct protection states of the system is given by (recall that a protection state is defined by a four-tuple (S, O, A, f_L)):

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

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 21 are satisfied, the safety problem is decidable regardless of the kinds of primitive operations in command bodies. Namely, Theorem 21 shows the existence of new non-monotonic systems where the safety problem is decidable.

3.3 Safety Analysis of Non-monotonic Protection Systems (II)

In this section, we again discuss the safety problem for non-monotonic systems in Theorem 21, but with further restriction that they have no creating commands.

Theorem 22. *The safety problem is NP-hard for protection systems, provided that:*

1. *the authorization schemes of the systems have no creating command,*[5] *and*
2. *the TR graphs of the systems have no cycle.*

Proof. First we present the subset sum problem [3]:

> 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 22, we run the *authorization scheme construction algorithm* (AC algorithm for short), which is given in Figure 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

$L_S \leftarrow \{l_0^s\}$; $C \leftarrow \{l_0^s\}$;
while $C \neq \phi$ **do begin** /* (1) */
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 begin** /* (3) */
if $i + w_j \leq I$ **then begin** /* (4) */
$L_S \leftarrow L_S \cup \{l_{i+w_j}^s\}$; $C \leftarrow C \cup \{l_{i+w_j}^s\}$; /* (5) */
end
else goto END1
end;
END1:
end;

$CM \leftarrow \{\alpha_{N,end}\}$; $C \leftarrow L_S$;
while $C \neq \phi$ **do begin** /* (6) */
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

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 Figure 4 is the TR graph of the authorization scheme generated by 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 Figure 4.

Now we consider whether AC algorithm eventually stops or not. Regarding **while** statement (1) of AC algorithm, l_i^s whose subscript i is the smallest is removed from C in step (2) and $l_{i+w_j}^s$ 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 $O(In)$.

We are now in a position to consider the polynomial-time reducibility from the subset sum problem to the safety problem in Theorem 22. For that purpose, we consider the protection system $\mathcal{P}$, which has the authorization scheme generated by AC algorithm. The initial state of $\mathcal{P}$, (S_0, O_0, A_0, f_{L0}), is given

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

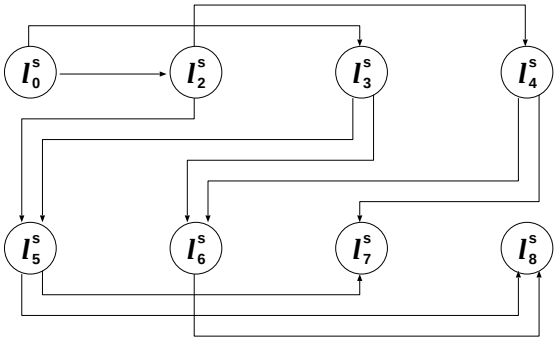

Fig. 4. TR graph generated by AC algorithm

- $S_0 = \{s\}$
- $O_0 = S_0 \cup \{o_1, o_2, \ldots, o_n\}$
- For every pair of $s \in S_0$ and $o \in O_0$, $A_0[s, o] = \phi$
- $f_{L_0}(s) = l_0^s$, $f_{L_0}(o_i) = l_i^o$ $(1 \leq i \leq n)$

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

in Figure 5. In the rest of this section we reduce the subset sum problem to the safety problem for $\mathcal{P}$, which is a restricted case of the safety problem in Theorem 22.

Let us assume 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}, ..., o_{j_l}, we can execute commands $\alpha_{0,j_1}(s, o_{j_1})$, $\alpha_{w(m_{j_1}),j_2}(s, o_{j_2})$, $\alpha_{w(m_{j_1})+w(m_{j_2}),j_3}(s, o_{j_3})$, ..., 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 $l_{w(m_{j_1})}^s$, $l_{w(m_{j_1})+w(m_{j_2})}^s$, ..., $l_{w(m_{j_1})+w(m_{j_2})+\ldots+w(m_{j_l})}^s = l_N^s$ 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}, ..., o_{j_l}, we can run commands $\alpha_{a_0,j_1}(s, o_{j_1})$, $\alpha_{a_1,j_2}(s, o_{j_2})$, ..., $\alpha_{a_{l-1},j_l}(s, o_{j_l})$ 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, ..., z_n and for every i $(1 \leq i \leq n)$, we set $w(m_i)$ to z_i. Consequently, we have $M = \{m_1, m_2, \ldots, m_n\}$, $w(m_1)$, $w(m_2)$, ..., $w(m_n)$.

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. □

4 Discussion

The reason for the decidability of Theorem 21 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 occur is finite since the total number of types is finite by assumption. In consequence, the number of objects is finite (Lemma 20) and the safety problem becomes decidable.

In non-monotonic systems in Theorem 21, 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 21 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 of decidable safety cases in previous work are for monotonic systems. On the other hand, the decidable cases in Theorem 21 and in Theorem 22 are for non-monotonic systems and fall outside the known decidable ones. Especially, we have shown that the safety problem in Theorem 22 belongs to a well-known complexity class, namely, NP-hard. However, in practice safety analysis is intractable unless it has polynomial time complexity. So further research is needed for non-monotonic systems where the safety analysis is decidable in polynomial time.

5 Conclusion

In this paper, we have proposed the Dynamic-Typed Access Matrix (DTAM) Model, which extends TAM model by allowing the type of an object to change dynamically. 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.

In subsequent research, we will go on investigating other non-monotonic systems where the safety analysis is decidable, especially, in polynomial time.

Acknowledgments. The author would like to thank the anonymous referees for valuable and helpful comments on this paper.

References

1. D. Bailey. A philosophy of security management. In M. D. Abrams, S. Jajodia, and H. J. Podell, editors, *Information Security: An Integrated Collection of Essays*, pages 98–110. IEEE Computer Society Press, 1995.
2. 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*, pages 142–153, 1996.
3. M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-completeness.* W. H. Freeman and Co., 1979.
4. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, Aug. 1976.
5. C. Meadows. Policies for dynamic upgrading. In S. Jajodia and C. E. Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 241–250. Elsevier Science Publishers B. V (North-Holland), 1991.
6. R. S. Sandhu. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–136, May 1992.
7. 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.