On the Expressive Power of the Unary Transformation Model*†

Ravi S. Sandhu and Srinivas Ganta‡

Center for Secure Information Systems
&
Department of Information and
Software Systems Engineering
George Mason University
Fairfax, VA 22030-4444

February 15, 1994

*A paper submitted for possible presentation at the *IEEE Computer Security Foundations Workshop VII*, June 14-16, 1994, Franconia, New Hampshire.

†All correspondence should be addressed to Ravi Sandhu, ISSE Department, George Mason University, Fairfax, VA 22030.

‡The work of both authors is partially supported by National Science Foundation grant CCR-9202270. Ravi Sandhu is also supported by the National Security Agency through contract MDA904-92-C-5141. We are grateful to Nathaniel Macon, Howard Stainer, and Mike Ware for their support and encouragement in making this work possible.

ABSTRACT

The Transformation Model (TRM) was recently introduced [10] in the literature by Sandhu and Ganta. TRM is based on the concept of *transformation of rights*. The propagation of access rights in TRM is authorized entirely by existing rights for the object in question. It has been demonstrated in earlier work that TRM is useful for expressing various kinds of consistency, confidentiality, and integrity controls.

In our previous work [10] a special case of TRM named Binary Transformation Model (BTRM) was defined. We proved that BTRM is equivalent in expressive power to TRM. This result indicates that it suffices to allow testing for only two cells of the matrix.

In this paper we study the relationship between TRM and the Unary Transformation Model (UTRM). In UTRM, individual commands are restricted to testing for only one cell of the matrix (whereas individual TRM commands can test for multiple cells of the matrix). Contrary to our initial conjecture, we found, quite surprisingly, that TRM and UTRM are formally equivalent in terms of expressive power. The implications of this result on safety analysis is also discussed in this paper. The construction used to prove the equivalence of TRM and UTRM also indicates that they might not be practically equivalent.

1 Introduction

In this paper we analyze the expressive power of a family of access control models called transformation models [10]. These models are based on the concept of *transformation of rights*, which simply implies that the possession of rights for an object by subjects allows those subjects to obtain and lose rights for that object and also grant and revoke the rights (for that object) to other subjects. Hence, in these models, the propagation of access rights is authorized entirely by the existing rights for the object in question. (More generally, propagation could also be authorized by the existing rights for the source and destination subjects, for example, in models such as HRU [4] and TAM [8].) The concept of transformation of rights allows us to express a large variety of practical security policies encompassing various kinds of consistency, confidentiality and integrity controls.

The concept of transformation of access rights was introduced by Sandhu in [7]. Based on it the monotonic transform model [7] and its non-monotonic extension (NMT) [9] were proposed. The simplicity and expressive power of NMT is demonstrated in [9] by means of a number of examples. It was recently discovered by the authors that NMT cannot adequately implement the document release example given in [9]. The reason behind this is the limited testing power of NMT. This led us to the formulation of the Transformation Model (TRM). TRM substantially generalizes NMT.

TRM does have good expressive power (which NMT lacks). TRM can also be implemented efficiently [10] in a distributed environment using a typical client-server architecture. This is due to the fact that the propagation of access rights in TRM is authorized entirely by existing rights for the object in question. In typical implementations these rights would be represented in an access control list (ACL), stored with the object. The server responsible for managing that object will have immediate access to all the information (i.e., the ACL) required to make access control decisions with respect to that object. Moreover, the effect of propagation commands is also confined only to the ACL of that object.

The Binary Transformation Model (BTRM) was defined in [10]. BTRM is a simpler version of TRM in which testing can involve up to two cells of the matrix. It has been proved in [10] that BTRM is formally equivalent to TRM. This also implies that it suffices to have systems which test for two cells of the matrix.

In this paper we study the relationship between TRM and the Unary Transformation Model (UTRM) defined in [10]. In UTRM the commands are authorized by checking for rights in a single cell of the access matrix. Contrary to our initial conjecture, we have discovered, surprisingly, that UTRM is theoretically equivalent to TRM in terms of expressive power. We also discuss why this result may not be true practically. The theoretical equivalence of TRM and UTRM helps in concluding that the safety results of UTRM are in no way better than that of TRM.

The rest of the paper is organized as follows. Section 2 gives a brief background of the Transformation Model (TRM). It also describes two models, UTRM and BTRM, which are restricted cases of TRM. In section 3, we prove that UTRM is formally equivalent to TRM in terms of expressive power. We also discuss, in this section, why this result might not be practically feasible, and also the implications of the result on safety analysis. Finally, section 4 concludes the paper.

2 Background

In this section, we review the definition of the Transformation Model (TRM), which was introduced in [10]. Our review is necessarily brief. The motivation for developing TRM, and its relation to other access control models are discussed at length in [10]. Following the review of TRM we briefly review the definitions of UTRM and BTRM.

2.1 The Transformation Model

TRM is an access control model in which authorization for propagation of access rights is entirely based on existing rights for the object in question. As discussed in the introduction this leads to an efficient implementation of TRM in a distributed environment using a simple client-server architecture. The expressiveness of TRM is indicated in [10] by enforcing various kinds of consistency, confidentiality, and integrity controls.

The protection state in TRM can be viewed in terms of the familiar access matrix. There is a row for each subject in the system and a column for each object. In TRM, the subjects and objects are disjoint. TRM does not define any access rights for operations on subjects, which are assumed to be completely autonomous entities. The $[X, Y]$ cell contains rights which subject X possesses for object Y.

TRM consists of a small number of basic constructs and a language for specifying the commands which cause changes in the protection state. For each command, we have to specify the authorization required to execute that command, as well as the effect of the command on the protection state. We generally call such a specification as an *authorization scheme* (or simply scheme) [8].

A scheme in the TRM is defined by specifying the following components.

1. A set of access rights R.
2. Disjoint sets of subject and object types, TS and TO, respectively.
3. A collection of three classes of state changing commands: *transformation commands, create commands*, and *destroy commands*. Each individual command specifies the authorization for its execution, and the changes in the protection state effected by it.

The scheme is defined by the security administrator when the system is first set up and thereafter remains fixed. It should be kept in mind that TRM treats the security administrator as an external entity, rather than as another subject in the system. Each component of the scheme is discussed in turn below.

The Typed Access Matrix Model (TAM) [8] and TRM are strongly related. They differ in state changing commands. In TRM, propagation of access rights is authorized entirely by existing rights for the object in question, whereas in TAM this authorization can involve testing rights for multiple objects. TRM commands can only modify one column at a time, where as TAM can modify multiple columns of the matrix. TRM does allow testing for absence of rights, while the original definition of TAM in [8] does not allow for such testing. If TAM is augmented with testing for absence of rights (as in [1]), it is then a generalization of TRM.

2.1.1 Rights

Each system has a set of rights, R. R is not specified in the model but varies from system to system. R is generally expected to include the usual rights such as *own, read, write, append* and *execute*. However, this is not required by the model. We also expect R to generally include more complex rights, such as *review, pat-ok, grade-it, release, credit, debit*, etc. The meaning of these rights will be explained wherever they are used in our examples.

The access rights serve two purposes. First, the presence of a right, such as r, in the $[S, O]$ cell of the access matrix may authorize S to perform, say, the read operation on O. Secondly, the presence of a right, say o, or the absence of right o, in $[S, O]$ may authorize S to perform some operation which changes the access matrix, e.g., by entering r in $[S', O]$. The focus of TRM is on this second purpose of rights, i.e., the authorization by which the access matrix itself gets changed.

2.1.2 Types of Subjects and Objects

The notion of type is fundamental to TRM. All subjects and objects are assumed to be strongly typed. Strong typing requires that each subject or object is created to be of a particular type which thereafter does not change. The advantage of strong typing is that it groups together subjects and objects into classes (i.e., types) so that instances of the same type have the same properties with respect to the authorization scheme.

Strong typing is analogous to tranquility in the Bell-LaPadula style of security models [2], whereby security labels on subjects and objects cannot be changed. The adverse consequences of unrestrained non-tranquility are well known [3, 5, 6]. Similarly, non-tranquility with respect to types has adverse consequences for the safety problem [8].

TRM requires that a disjoint set of subject types, TS, and object types, TO, be specified in a scheme. For example, we might have TS=$\{user,\ security\text{-}officer\}$ and TO=$\{user\text{-}files,\ system\text{-}files\}$, with the significance of these types indicated by their names.

2.1.3 State Changing Commands

The protection state of the system is changed by means of TRM commands. The security administrator defines a finite set of commands when the system is specified. There are three types of state changing commands in the TRM, each of which is defined below.

Transformation Commands

We reiterate that every command in TRM has a condition which is on a single object and the primitive operations comprising the command are only on that object. In all the commands the last parameter in the command is the object which is being manipulated, and the first parameter is the subject who initiates the command.

A *transformation command* has the following format:

command $\alpha(X_1 : s_1,\ X_2 : s_2,\ \ldots,\ X_k : s_k,\ O : o_i)$
  if *predicate* **then**
   op_1; op_2; $\ldots$; op_n
end

The first line of the command states that α is the name of the command and $X_1, X_2, \ldots, X_k, O$ are the formal parameters. The formal parameters X_1, X_2, $\ldots$, X_k are subjects and of types s_1, s_2, $\ldots$, s_k. The **only** object formal parameter O is of type o_i and is the last parameter in the command.

The second line of the command α is the predicate and is called the *condition* of the command. The predicate consists of a boolean expression composed of the following terms connected by the usual boolean operators (such as $\wedge$ and $\vee$):

$$r_i \in [S, O] \text{ or } r_i \notin [S, O]$$

where r_i is a right in R and S can be substituted with any of the formal subject parameters. Simply speaking the predicate tests for the presence and absence of some rights for subjects on object O. Given below are some examples of TRM predicates:

1. $approve \in [S_1, O] \wedge prepare \notin [S_2, O]$
2. $prepare \in [S, O] \wedge assign \in [S_1, O] \wedge creator \notin [S, O]$

3. $own \in [S, O] \vee write \in [S, O]$

4. $r_1 \in [S_1, O] \wedge (r_2 \in [S_1, O] \vee r_1 \in [S_2, O]) \wedge r_3 \in [S_2, O] \wedge r \in [S_3, O]$

If the condition is omitted, the command is said to be an *unconditional command*, otherwise it is said to be a *conditional command*.

The third line of the command consisting of sequence of operations op_1; op_2; ...; op_n is called the *body* of α. Each op_i is one of the following two primitive operations:

- **enter** r **into** $[X, O]$
- **delete** r **from** $[X, O]$

It is important to note that all the operations enter or delete rights for subjects on object O alone. The **enter** operation enters a right $r \in R$ into an existing cell of the access matrix. The contents of the cell are treated as a set for this purpose, i.e., if the right is already present, the cell is not changed. The **delete** operation has the opposite effect of **enter**. It (possibly) removes a right from a cell of the access matrix. Since each cell is treated as a set, **delete** has no effect if the deleted right does not already exist in the cell. The **enter** operation is said to be *monotonic* because it only adds and does not remove from the access matrix. Because **delete** removes from the access matrix it is said to be a *non-monotonic* operation.

A command is invoked by substituting actual parameters of the appropriate types for the formal parameters. The condition part of the command is evaluated with respect to its actual parameters. The body is executed only if the condition evaluates to true.

Some examples of *transformation commands* are given below.

command *transfer-ownership* $(S_1 : s, S_2 : s, O : o)$
  if $own \in [S_1, O]$ **then**
  enter own in $[S_2, O]$
  delete own from $[S_1, O]$
end

command $grade\ (S_1 : professor, S_2 : student, O : project)$
  if $own \in [S_2, O] \wedge grade \in [S_1, O]$ **then**
  enter *good* in $[S_2, O]$
  delete $grade$ from $[S_1, O]$
end

command *issue-check* $(S_1 : clerk, O : voucher)$
  if $prepare \notin [S_1, O] \wedge approve \notin [S_1, O]$ **then**
  enter *issue* in $[S_1, O]$
end

Command *transfer-ownership* transfers the ownership of a file from one subject to another. In the command *grade*, the professor gives right *good* to the students project. In command *issue-check*, a clerk gets an *issue* right only if he/she is not the one who prepared and approved it.

Create Commands

A *create command* is an unconditional command. The creator of an object gets some rights for the created object like *own, read*, etc., as specified in the body of the command. No subject other than the creator will get rights to the created object in the *create command*. Subjects other than the creator can subsequently acquire rights for the object via transformation commands. In short, the effect of a *create command* is to introduce a new column in the matrix with some new rights for the subject who created it.

A typical *create command* is given below.

command $create(X_1 : s_1, O : o_i)$
  create object O
  enter *own* in $[X_1, O]$
end

In the general case the body of the command may enter any set of rights in the (X_1, O) cell.

A *create command* is an unconditional command as the command cannot check for rights on an object which does not exist, and TRM commands do not allow testing for rights on objects other than the object which is being created. The *create object* operation requires that the object being created have an unique identity different from all other objects. A *create command* is monotonic.

Destroy Commands

A *destroy command* is a conditional command. The effect of a *destroy command* on the matrix will be removal of the corresponding column from the access matrix. A typical *destroy command* is given below.

command $destroy(X_1 : s_1, O : o_i)$
  if $own \in [X_1, O]$ **then**
  destroy object O
end

In this case the condition ensures that only the owner can destroy the object. More generally, deletion can be authorized by some combination of rights possessed by the destroyer. A *destroy command* is non-monotonic.

2.1.4 Summary of TRM

To summarize, a system is specified in TRM by defining the following finite components.

1. A set of rights R.
2. A set of disjoint subject and object types T.
3. A set of state-changing transformation, creation and destroy commands, as defined above.
4. The initial state.

We say that the rights, types and commands define the system *scheme*. Note that once the system scheme is specified by the security administrator it remains fixed thereafter for the life of the system. The system state, however, changes with time.

2.2 The Unary Transformation Model (UTRM)

The Unary Transformation Model is a simpler version of TRM in which testing in a command can be on only one cell of the matrix. A UTRM predicate consists of a boolean expression composed of the following terms:

$$r_i \in [X_j, O] \text{ or } r_i \notin [X_j, O]$$

where r_i is a right in R and X_j can be any one of the formal subject parameters, but all the terms in the expression must have the same X_j. In other words, the predicate tests for the presence and absence of rights for a single subject X_j on object O. Usually X_j will be the first parameter in the command, since that is the one who initiates the command.

UTRM generalizes the model called NMT (for Non-Monotonic Transform) [9]. The transformation commands in NMT, viz., grant transformation and internal transformation, are easily expressed as UTRM commands and test for rights in one cell of the matrix.

2.3 The Binary Transformation Model (BTRM)

The Binary Transformation Model is also a simpler version of TRM in which testing in a command can involve up to two cells of the matrix. A BTRM predicate consists of a boolean expression composed of the following terms:

$$r_i \in [X_j, O] \text{ or } r_i \notin [X_j, O]$$

where r_i is a right in R and X_j can be any one of the formal subject parameters, but the expression can have at most two different X_j's from the given parameters. In other words, the predicate tests for the presence and absence of rights for at most two subjects (on object O). One of the X_j's will typically be the first parameter which is the initiator of the command.

3 Expressive Power of UTRM

We now analyze the relative expressive power of TRM and UTRM. Recall that UTRM is a restricted version of TRM. It is the same as TRM except that the testing in a command can only be on a single cell. It has been proved in [10] that TRM is equivalent to BTRM with just three parameters. Thus to prove the equivalence of TRM and UTRM, it is sufficient to show that for every BTRM scheme with three parameters, there exists an equivalent UTRM scheme.

We will now show how any given BTRM command can be simulated by multiple UTRM commands. The Boolean condition of any BTRM command, say Y, can be converted into the familiar *disjunctive normal form* which consists of a disjunction (i.e., $\vee$) of *minterms*. Each minterm is a conjunction (i.e., $\wedge$) of primitive terms of the form $r_i \in (S_i, O)$ or $r_i \notin (S_i, O)$. The command Y can then be factored into multiple commands, each of which has one minterm as its condition and the original body of Y as its body. Hence, we can assume without loss of generality that the predicate of every BTRM command consists of a conjunction of primitive terms.

We will illustrate the construction by simulating a BTRM command X (which has three parameters) of the following format.

command $X(S_1 : t_1, S_2 : t_2, O : o)$
if $P_1 \wedge P_2$ **then**
    operations in (S_1, O)
    operations in (S_2, O)
end

In the above command, each P_i is itself composed of a conjunction of terms $r_j \in (S_i, O)$ or $r_j \notin (S_i, O)$, where $r_j \in R$. Intuitively P_i tests for the presence of, and

	$O : o$
$S_1 : t_1$	α_1
$S_2 : t_1$	α_2
$S_3 : t_2$	α_3
...	
$S_n : t_x$	α_n

(a) Initial state of BTRM

	$O : o$
$Lock : lock$	L
$S_1 : s_1$	α_1
$S_2 : s_2$	α_2
...	
$S_n : s_n$	α_n

(b) Initial state of UTRM

Figure 1: UTRM simulation of command X

absence of some rights in the single cell (S_i, O). In the body of command X, the phrase "**operations** in (S_i, O)" denotes a sequence of enter and delete (or possibly empty) operations in the (S_i, O) cell. Note that the types t_1 and t_2 need not be distinct. The formal parameters S_1, S_2 must of course be distinct, but the actual parameters used on a particular invocation of this command may have repeated parameters as allowed by parameter types. For ease of exposition, we will initially assume that the actual parameters S_1 and S_2 are distinct. The simulation of a BTRM command with repeated parameters, will be explained at the end of this section.

We now consider how the BTRM command X can be simulated by several UTRM commands. As X tests two cells, it is obvious that the simulation of X cannot be a single UTRM command. Since UTRM can test for only one cell, the simulation of X must be done by multiple commands in the UTRM system. The key to doing this successfully is to prevent other UTRM commands from interfering with the simulation of the given BTRM command, X. The simplest way to do this is to ensure that BTRM commands can be executed in the UTRM simulation only one at a time. To do this we need to synchronize the execution of successive BTRM commands in the UTRM simulation.

This synchronization is achieved by introducing an extra subject called *Lock* of type *lock*, and an extra right, L. The role of *Lock* is to sequentialize the execution of simulation of BTRM commands in the UTRM system. The type *lock* is assumed, without loss of generality, to be distinct from any type in the given BTRM system.

Also the initial state of the UTRM system is modified in such a way that every subject of the BTRM system is given a different type. This assumption is acceptable within the framework of these models, because the number of subjects in the system is static (as there is no creation and destruction of subjects in Transformation Models). We will further discuss the implication of this assumption at the end of this section. If the initial state of the BTRM system resembles figure 1(a), then in our construction the initial state of the UTRM system resembles figure 1(b). The α_i's are sets of rights in the indicated cell.

The UTRM simulation of X proceeds in five phases as indicated in figure 2 and 3. In these figures we show only the relevant portion of the access matrix, and only those rights introduced specifically for the UTRM simulation. Hence, for clarity of the diagram, we do not show the α_i's rights, but these are intended to be present. Since the focus in TRM is on a single object, the matrix reduces to a single column for that object.

The objective of the first phase is to make sure that no other UTRM command corresponding to another BTRM command can execute (on object O) until the simulation of X is complete. The first phase also ensures that the actual parameters of the UTRM commands are tied to the actual parameters of the BTRM command. In the second phase, if P_1 part of the condition of X is true, then that fact is indicated to all the subjects in the system. If P_1 is false, the second phase indicates the failure of the condition of X by entering right $cleanX$ in $(Lock, O)$. In the third phase, if the condition of X is true, then the body of X is partly executed. If the condition of X is false, the third phase also indicates the failure of the condition of X. In the fourth phase, the rest of the body of X is executed. And finally the fifth phase removes all the additional bookkeeping rights and also indicates that the simulation of X is complete. Each of the phases and the commands used are explained briefly below.

The UTRM command *X-1-invocation* corresponds to phase I. It checks for right L in $(Lock, O)$, and if present deletes it, to make sure that no other UTRM command (simulating some other BTRM command) can execute (on object O) until the simulation of X is complete. It also makes sure that the actual parameters of X are used in the simulation by entering rights p_1, p_2 in cells (S_1, O) and (S_2, O) respectively. It also enters the right X in cells $(S_1, O), (S_2, O)$ to indicate that the simulation of X is currently in progress. The matrix, after the execution of command *X-1-invocation* resembles figure 2(a). To simulate X, we need a different *X-i-invocation* command for each distinct combination of a subject of type t_1 and a subject of type t_2. For example, if there are m subjects of type t_1 and n subjects of type t_2 in the BTRM system, then in phase I, the simulation of command X requires mn commands in the UTRM system. Phase I command simulating X with actual parameters corresponding to types s_1 and s_2 respectively is given below.

command *X-1-invocation*$(S_1 : s_1, S_2 : s_2, Lock : L, O : o)$
if $L \in (Lock, O)$ **then**
    delete L from $(Lock, O)$
    enter p_1 in (S_1, O)
    enter p_2 in (S_2, O)
    enter X in (S_1, O)
    enter X in (S_2, O)
end

In phase II, the commands test if the P_1 part of the condition of X is true. If so, the command *X-2-successfull* gives the right P_1^* to all the subjects (to indicate that

	O
$Lock$	
S_1	p_1, X
S_2	p_2, X
S_n	

(a) End of phase I

	O
$Lock$	
S_1	p_1, X, P_1^*
S_2	p_2, X, P_1^*
$\ldots$	
S_n	P_1^*

(b) End of phase II

	O
$Lock$	
S_1	p_1, X, P_1^*, P_2^*
S_2	$operations, p_2, X, P_1^*, P_2^*$
$\ldots$	
S_n	P_1^*, P_2^*

(c) End of phase III

	O
$Lock$	$cleanX$
S_1	$operations, p_1, X, P_1^*, P_2^*$
S_2	p_2, X, P_1^*, P_2^*
$\ldots$	
S_n	P_1^*, P_2^*

(d) End of phase IV

Figure 2: UTRM simulation of the authorized BTRM command X

P_1 is true). The matrix at the end of successfull phase II, resembles figure 2(b). If P_1 is false, the command *X-2-fail* enters the right $cleanX$ in $(Lock, O)$ to indicate that the condition of command X is false. The right $cleanX$ in $(lock, O)$ also indicates that simulation has reached the final phase. In this case, the matrix at the end of failed phase II, resembles figure 3(a). It is important to note that in phase II, only one of *X-2-fail* or *X-2-successfull* can execute. To simulate X, we need a different *X-2-successfull* command for each subject of type t_1 and a different *X-2-fail* command for each subject of type t_1. Phase II commands simulating X with actual parameters corresponding to types s_1 and s_2 respectively, are given below.

command *X-2-successfull*$(S_1 : s_1, S_2 : s_2, S_3 : s_3, \ldots, S_n : s_n, Lock : L, O : o)$
if $p_1 \in (S_1, O) \wedge P_1 \wedge X \in (S_1, O)$ **then**
    enter P_1^* in (S_1, O)
    $\ldots$
    enter P_1^* in (S_n, O)
end

command *X-2-fail*$(S_1 : s_1, Lock : L, O : o)$
if $p_1 \in (S_1, O) \wedge \neg P_1 \wedge X \in (S_1, O)$ **then**
    enter $cleanX$ in $(Lock, O)$
end

	O
$LOCK$	$cleanX$
S_1	p_1, X
S_2	p_2, X
S_n	

(a) End of phase II

	O
$LOCK$	$cleanX$
S_1	p_1, X, P_1^*
S_2	p_2, X, P_1^*
S_n	P_1^*

(b) End of phase III

Figure 3: UTRM simulation of unauthorized BTRM command X

Note that these are valid UTRM commands because all tests in the condition part are in the (S_1, O) cell.

In phase III, the rest of the condition of X is tested in *X-3-successfull*. If the condition is true, part of the body of X is executed. The matrix at the end of successfull phase III, resembles figure 2(c). If the condition is not true, the command *X-3-fail* enters the right $cleanX$ in $(Lock, O)$ to indicate that the simulation of X has failed. In this case the matrix at the end of phase III, resembles figure 3(b). It is important to note that in phase III, only one of *X-3-fail* or *X-3-successfull* can execute. Here also to simulate X, we need a different *X-3-successfull* command for each subject of type t_2 and a different *X-3-fail* command for each subject of type t_2. Phase III commands simulating X with actual parameters corresponding to types s_1 and s_2 respectively, are given below.

command *X-3-successfull*$(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, Lock : L, O : o)$
if $p_2 \in (S_2, O) \wedge P_1^* \in (S_2, O) \wedge P_2 \wedge X \in (S_2, O)$ **then**
    operations in (S_2, O)
    enter P_2^* in (S_1, O)
    ...
    enter P_2^* in (S_n, O)
end

command *X-3-fail*$(S_2 : s_2, Lock : L, O : o)$
if $p_2 \in (S_2, O) \wedge \neg P_2 \wedge X \in (S_2, O)$ **then**
    enter $cleanX$ in $(Lock, O)$
end

In the fourth phase, the rest of the body of X is executed. Also right $cleanX$ is entered in $(lock, O)$ also indicate that simulation has reached the final phase. It is also important to note that the phase IV command is executed only if the commands executed in phases II and III are successfull commands. The matrix at the end of

phase IV resembles figure 2(d). Here also to simulate X, we need a different *X-4-successfull* command for each subject of type t_1. Phase IV commands simulating X with actual parameters corresponding to types s_1 and s_2 respectively, are given below.

command *X-4-successfull*$(S_1 : s_1, Lock : L, O : o)$
if $p_1 \in (S_1, O) \wedge P_2^* \in (S_1, O) \wedge X \in (S_1, O)$ **then**
    operations in (S_1, O)
    enter $cleanX$ in $(Lock, O)$
end

In the final phase, all the bookkeeping rights $R^* = \{p_1, p_2, X, P_1^*, P_2^*, cleanX\}$ are deleted. Also right L is entered back into $(Lock, O)$ to indicate that the simulation of X is complete and the simulation of some other BTRM command (on object O) can now begin. The matrix after the final phase, resembles figure 1(b). The phase V command to simulate X is given below.

command *X-5-complete*$(S_1 : s_1, S_2 : s_2, S_3 : s_3, \ldots, S_n : S_n, Lock : L, O : o)$
if $cleanX \in (Lock, O)$ **then**
    delete R^* from (S_1, O)
    ...
    delete R^* from (S_n, O)
    delete $cleanX$ from $(Lock, O)$
    enter L in $(Lock, O)$
end

The important thing to be noted from our construction is that once the UTRM simulation of command X proceeds with some actual parameters in phase I, then in all other phases, the commands execute with the same parameters.

We have shown how a BTRM command X, can be simulated by UTRM commands. The command X has actual parameters (S_1,S_2) which are distinct (as they are of types t_1 and t_2). A BTRM command can also have actual parameters which are repeated. This is possible if the command has two parameters of the same type. Our construction can be easily extended to simulate such commands. For example, if the BTRM command X has both the subject parameters of type t_1, then the following type of commands are needed **along** with the five phases of commands explained before. The command *X-1-invocation-repeated* will make sure that the two actual subject parameters of X are same and the command *X-repeated-done* does the necessary operations (if the two actual subject parameters of X are same). If there are m subjects of type t_1 in the BTRM system, then we need to give m *X-1-invocation-repeated* commands and m *X-repeated-done* commands. The UTRM commands simulating X with repeated actual parameters corresponding to type s_1 are given below.

command *X-1-invocation-repeated*$(S_1 : s_1, Lock; L, O : o)$
if $L \in (Lock, O)$ **then**
    delete L from $(Lock, O)$
    enter p_1 in (S_1, O)
    enter p_2 in (S_1, O)
    enter X in (S_1, O)
end

command *X-repeated-done*$(S_1 : s_1, Lock; L, O : o)$
if $X \in (Lock, O) \wedge p_1 \in (S_1, O) \wedge p_2 \in (S_1, O) \wedge P_1 \wedge P_2$ **then**
    operations in (S_1, O)
    enter $cleanX$ in $(Lock, O)$
end

A proof sketch for the correctness of the construction is given below.

Theorem 1 *For every BTRM system SYS_1, the construction outlined above produces an equivalent UTRM system SYS_2.*

Proof Sketch: It is easy to see that any reachable state in SYS_1 can be reached in SYS_2 by simulating each BTRM command by UTRM commands, as discussed above. Conversely any reachable state in SYS_2, with $L \in (LOCK, O)$, will correspond to a reachable state in SYS_1. A reachable state in SYS_2, with $L \notin (LOCK, O)$ and which passes phase III, will correspond to a state in SYS_1 where one BTRM command has been partially completed. A state in SYS_2, with $L \notin (LOCK, O)$ and which fails the testing phase, will then lead SYS_2 to a previous state where $L \in (LOCK, O)$, which is reachable in SYS_1. Our construction also ensures that once the UTRM simulation passes the first phase, then the simulation proceeds with the same actual parameters of the first phase. Hence the above construction proves the equivalence of TRM and UTRM. A formal inductive proof can be easily given, but is omitted for lack of space.

Discussion

The construction given in this section proves formally the equivalence of TRM and UTRM. The construction assumes that the initial state of the UTRM system is modified in such a way that every subject of the BTRM system is given a different type. This assumption is acceptable within the framework of these models, because the number of subjects is static (as there is no creation and destruction of subjects in Transformation Models). However, this assumption is not a realistic one in practical situations. If subject creation is allowed, then our construction breaks down. This is due to the fact that the number of commands in the UTRM system depend on the number of subjects in the system and if subject creation is allowed, then the commands in the initial state of the UTRM system would not be enough to accommodate

new subjects. New commands need to be introduced whenever a subject is created and this is not allowed in the framework of TRM. Hence even though, TRM and UTRM are theoretically equivalent, practically this might not be the case.

If this construction is not allowed, we could prove the same result, by assuming that every subject in the UTRM system is associated with a different right. This assumption is similar to the assumption that every subject in the BTRM system should be of different type. We are not sure if we can prove the theoretically equivalence of TRM and UTRM without these assumptions. Our future works involves proving the equivalence or non-equivalence of TRM and UTRM without these assumptions.

The theoretical equivalence of TRM and UTRM would imply that the safety results of UTRM are not any better than TRM. As TRM does not have any efficient non-monotonic safety results, neither would UTRM. This leads to the fact that it is difficult to have a model which can express some simple policies and at the same time have efficient non-monotonic safety results.

4 Conclusion

In this paper we have shown that the Transformation Model (TRM) [10] and the Unary Transformation Model (UTRM) [10] are formally equivalent in expressive power. The theoretical equivalence of TRM and UTRM would imply that the safety results of UTRM are not any better than TRM. The fact that TRM does not yet have any efficient non-monotonic safety results indicates that it is difficult to have a model which can express some simple policies and at the same time have efficient non-monotonic safety result. Our construction used in proving the equivalence of TRM and UTRM also indicates that TRM and UTRM might not be practically equivalent.

References

[1] Ammann, P.E. and Sandhu, R.S. "Implementing Transaction Control Expressions by Checking for Absence of Access Rights." *Proc. Eighth Annual Computer Security Applications Conference*, San Antonio, Texas, December 1992.

[2] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, Mitre, Bedford, Massachusetts (1975).

[3] Denning, D.E. "A Lattice Model of Secure Information Flow." *Communications of ACM* 19(5):236-243 (1976).

[4] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D. "Protection in Operating Systems." *Communications of ACM* 19(8), 1976, pages 461-471.

[5] McLean, J. "A Comment on the 'Basic Security Theorem' of Bell and LaPadula." *Information Processing Letters* 20(2):67-70 (1985).

[6] McLean, J. "Specifying and Modeling Computer Security." *IEEE Computer* 23(1):9-16 (1990).

[7] Sandhu, R.S. "Transformation of Access Rights." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1989, pages 259-268.

[8] Sandhu, R.S. "The Typed Access Matrix Model" *IEEE Symposium on Research in Security and Privacy*, Oakland, CA. 1992, pages 122-136.

[9] Sandhu, R.S. and Suri, G.S. "Non-monotonic Transformations of Access Rights." *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1992, pages 148-161.

[10] Sandhu, R.S. and Srinivas Ganta. "On the Minimality of Testing for Rights in Transformation Models." *To appear in IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 16-18, 1994.