EXPRESSIVE POWER OF ACCESS CONTROL MODELS BASED ON PROPAGATION OF RIGHTS

by

SRINIVAS GANTA

A Dissertation

Submitted to the

Graduate Faculty

of

George Mason University

in Partial Fulfillment of

the Requirements for the Degree

of

Doctor of Philosophy

Information Technology

Committee:

_____ Dr. Ravi Sandhu, Dissertation Director

_____ Dr. Larry Kerschberg

_____ Dr. Sushil Jajodia

_____ Dr. Paul Ammann

_____ Dr. Pearl Wang

_____ Dr. W. Murray Black, Associate for Graduate Studies and Research

_____ Dr. Andrew P. Sage, Dean, School of Information Technology and Engineering

Date: _____

Spring 1996

George Mason University

Fairfax, Virginia

Expressive Power of Access Control Models Based on Propagation of Rights

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University.

By

SRINIVAS GANTA
B.Tech., Jawaharlal Nehru Technological University, Kakinada, INDIA 1989
M.S., George Mason University, Fairfax, VA, 1991

Director: Dr. Ravi Sandhu, Professor
Information and Software Systems Engineering

Spring 1996
George Mason University
Fairfax, Virginia

Acknowledgments

I would like to express my sincere gratitude and appreciation to my PhD advisor and dissertation director, Professor Ravi Sandhu, for all his valuable guidance, teaching, and encouragement during the realization of this research.

I sincerely thank Professor Sushil Jajodia for letting me be part of Center for Secure Information Systems and also for always asking me to work harder. Thanks also goes to Professor Larry Kerschberg, Professor Paul Amman, and Professor Pearl Wang for their general guidance, useful comments, valuable suggestions, and thoughtful revision of the research material.

I like to thank my sister Geeta for all her love and affection. I like to thank Bunty, Rama and Raja for taking me into their family. I like to thank Varun, Latha, Raju, Lakshmi and RK for making my life so enjoyable. I like to thank Srini and Gino for their friendship, valuable advices and for treating me like a brother. I also like to thank all of my friends at Mason who made my stay at Mason a very memorable one.

I dedicate this dissertation to my parents Uma Devi and Krishna Murthy, and to Professor Sandhu. I cannot thank my parents enough for their support and encouragement during all the years with perseverance for the accomplishment of my greatly desired dream and goal of completing my PhD degree. I would like to thank Professor Sandhu for always being there for me like a father, and I would like to let him know that if it was not for him I would not be getting a PhD.

Table of Contents

List of Tables

Table **Page**

List of Figures

Abstract

EXPRESSIVE POWER OF ACCESS CONTROL MODELS BASED ON PROPAGATION OF RIGHTS

SRINIVAS GANTA, Ph.D.

George Mason University, 1996

Dissertation Director: Dr. Ravi Sandhu

Access control models provide a formalism and framework for specifying, analyzing and implementing security policies in multi-user systems. These models are usually defined in terms of the well-known abstractions of subjects, objects and access rights. Access control models should be flexible enough to express a wide range of policies. The flexibility of an access control model can be measured through its expressive power.

In this thesis, we compare the expressive power of access control models in a relative manner. Intuitively, model A is at least as expressive as model B, if every policy that can be expressed by B can also be expressed in A. If the converse is also true, than the models are equivalent and can express exactly the same set of policies. In particular, we compare the expressive power of Typed Access Matrix Model (TAM), Augmented TAM (ATAM) and their variations. TAM was introduced by Sandhu, and it is known to have broad expressive power. ATAM an extension of TAM was introduced by Ammann and Sandhu and it allows for testing for absence of rights, whereas TAM does not.

We first develop a formalism to compare the relationship between expressive

power of two models. We define two notions of equivalence: *Strong* and *Weak*. Strong equivalence implies weak equivalence, but not vice-versa. Our formalism helps in proving whether two models are equivalent (strongly or weakly or both) or not equivalent (either strongly or weakly). We specifically show that TAM and ATAM are not strongly equivalent, but they are weakly equivalent. This indicates that adding testing for absence of rights does increase the expressive power of access control models. We also illustrate the practical significance of this fact by showing that implementing transaction control expressions does require the ability to test for absence of rights in access control models.

We then prove an important fact that very simple models do have the most general expressive power of the more general models they are derived from. We get to this conclusion by defining simple models obtained by posing restrictions on TAM and ATAM and comparing their expressive power. We also indicate that simplification of models can be carried to a point, beyond which they loose some expressive power. We also discuss the implications of the expressive power results on safety and implementation issues of access control models.

Chapter 1

Introduction and Problem Statement

1.1 Introduction

Access control policies are needed in any information system that facilitates controlled sharing of data and other resources among multiple users. Access control policies are specified, analyzed, and implemented through the formalism and framework specified by access control models. It is desirable that the access control models be flexible enough so that the system can support the security administrator in enforcing a policy appropriate for the organization. The following examples illustrate the need for flexible access control models.

Consider a typical document approve/release example where a scientist creates a document, and prior to releasing that document, needs approval from two separate officers: a security-officer and a patent-officer. After review of the document, the security-officer and the patent-officer each grant the scientist an appropriate approval. After obtaining approval from both officers, the scientist can publish the document. The access control model should be flexible enough to enforce this example in many different ways. One possibility is where once the scientist requests approval from the two officers, he can never change that document again. If one of the officers rejects the document then the document is considered to be dead. Another possibility is — instead of re-creating a new document following a rejection, it might be more efficient from the viewpoint of the scientist, to be allowed to edit the existing document.

A second example is one where a check needs to be approved by three different supervisors, and a clerk can issue the check only after the approval of the supervisors. If an approval of a check by a manager is equivalent to approval by two supervisors, than there are many ways of achieving the desired approval. The access control model should be flexible enough to allow the approval of the check by three different supervisors or by two managers or by one supervisor and a manager or by two supervisors and a manager. If the model is not flexible and if it only allows three supervisors to approve, then this might lead to a situation where all the supervisors are busy to approve a check and the managers (who are free) cannot approve. Hence the above examples indicate the necessity for flexible access control models.

Several access control models, have been published in the literature (see for example [AS92a, HRU76, LM82, San88b, San89b, San92b, SS92a]). These models are defined in terms of the well-known abstractions of subjects, objects and access rights. Three important issues concerned with access control models are expressive power, safety and implementation.

Access control models should have adequate expressive power, i.e, they should enforce policies of practical interest. One measure of expressive power of a model (and there can be other measures) is relative and is measured in terms of the expressive power of some other model. Security models based on propagation of access rights must confront the safety problem. In its most basic form, the safety question asks: is there a reachable state in which a particular subject possesses a particular right for a specific object? It is obviously desirable that for any policy which is enforced, we should be able to answer the safety question efficiently. Last but not the least is implementation. In particular, we are interested in implementing the models in a distributed environment using a simple client-server architecture. The next section discusses the history of access control models with respect to these issues.

1.2 Brief History of Access Control Models

The Access Matrix proposal of [Lam71] contains a particular set of rules to control the propagation of access rights. These rules basically give the owner of an object complete discretion regarding rights to that object. Graham and Denning [GD72] proposed various rules by which the discretionary ability of the owner could be granted to other subjects. Even though many such rules can be proposed, no one set of rules could be argued to be the single universal policy which everyone should implement in their systems. This led Harrison, Ruzzo and Ullman to develop a model called HRU [HRU76]. This model could easily express complex policies for propagation of access rights. HRU does have good expressive power, unfortunately it has extremely weak safety properties. In general, safety is undecidable [HR78]. Safety is undecidable for most policies of practical interest, even in the monotonic version of HRU [HR78]. Monotonic models do not allow deletion of access privileges. As the ability to delete access privileges is an important requirement, monotonic models are too restrictive to be of much practical use. It appears that HRU does not have an useful special case for which safety is efficiently decidable.

The take-grant model was introduced by Lipton and Snyder [LS77] in response to the negative safety results of HRU. The take-grant model was analyzed by a number of authors [LM82, Bis88, Sny81]. The take-grant model has linear time algorithms for safety. But the disadvantage of take-grant model is that it was deliberately designed to be of limited expressive power to eliminate the undecidable safety of HRU. Hence, there is a significant difference between the expressive power of take-grant and HRU.

The Schematic Protection Model (SPM) [San88b] was proposed by Sandhu to fill the gap in expressive power between take-grant and HRU. SPM has strong safety properties and it can express wide variety of policies of practical interest. SPM is in fact less expressive than monotonic HRU [ALS92]. This led to the development

of Extended Schematic Protection Model (ESPM) [AS92a] by Ammann and Sandhu. ESPM extends single parent creation operation in SPM to allow multiple parents for a child. ESPM is equivalent to monotonic HRU [AS90, AS92a] in terms of expressive power and it still retains the positive safety results of SPM [AS90, AS91]. Even though SPM and ESPM have very good safety properties, they still do not have the expressive power of HRU, as SPM and ESPM are monotonic.

Typed Access Matrix Model (TAM) [San92b] was proposed by Sandhu and its extension Augmented Typed Access Matrix Model (ATAM) [AS92b] was proposed by Ammann and Sandhu to accommodate both the strong expressive power of HRU and positive safety results of ESPM. TAM was defined by introducing strong typing into HRU (i.e., each subject or object is created to be of a particular type which thereafter does not change). TAM has the same expressive power as HRU, and at the same time monotonic TAM has strong safety properties similar to SPM and ESPM. ATAM is same as TAM with the addition ability to test for absence of access rights. Hence ATAM can express all the policies that can be expressed by TAM.

Single-Object TAM (SOTAM) [SS92b] was defined by Sandhu and Suri. SOTAM is a simplified version of TAM, with the restriction that all operations in a command are required to operate on a single object. It has been shown in [SS92b] that SOTAM has a very simple implementation in a typical client-server architecture. It has also been conjectured in [SS92b] that SOTAM is equivalent to TAM in terms of expressive power.

1.3 Problem Statement

In this thesis, we consider Typed Access Matrix Model (TAM) [San92b] and its extension Augmented Typed Access Matrix Model (ATAM) [SS92b] as they both have strong expressive power (as discussed above). We would like to compare the expres-

sive power of TAM and ATAM to address the impact of adding testing for absence of rights in access control models (on expressive power). If adding testing for absence of rights increases the expressive power of access control models, we would also like to know whether testing for absence of rights is actually needed in practical situations. We are also interested in the expressive power of various simpler models obtained by posing some restrictions on TAM and ATAM. We compare the expressive power of these simpler models with TAM and ATAM. We are also interested to know how far TAM and ATAM can be restricted beyond which they loose some expressive power. We would like to address the implications of these expressive power results on safety and implementation.

1.4 Summary of Contributions

(1) Our first contribution in this thesis is that we provide a formalism to compare the relationship between expressive power of two models. We define two notions of equivalence: *Strong* and *Weak*. Our formalism helps in proving whether two models are equivalent (strongly or weakly or both) or not equivalent (both strongly and weakly).

(2) The second contribution is that we formally prove the result of adding testing for absence of rights (in access control models) on expressive power. By formally proving that TAM is not strongly equivalent to ATAM, we prove that adding testing for absence of rights does increase the expressive power of access control models. We also show that the ability to test for absence of rights is desired in practical situations by arguing that implementing transaction control expressions does require the ability to test for absence of rights in access control models. [1]

[1] Our construction to show this builds upon and extends the construction outlined in [AS92b].

(3) The third contribution of this thesis is that we prove an important fact that very simple models do have the most general expressive power of the models they are derived from. We also indicate that simplification of models can be carried to a point, beyond which they loose some expressive power. The expressive power results obtained in this thesis indicate that the safety analysis of these simple models is as difficult as the most general model they are derived from.

1.5 Organization of the Thesis

Chapter 2 gives a brief background on access control models. In particular, it reviews the Typed Access Matrix Model (TAM) defined in [San92b]. It also defines simpler models which are obtained by posing some restrictions on TAM and ATAM. Chapter 3 provides a formalism to compare the expressive power of two models. It also defines the notion of strong and weak equivalence. Chapter 4 addresses the impact of adding testing for absence of rights in access control models on expressive power. Chapter 5 addresses the need for testing for absence of rights in implementing transaction control expressions (TCE's) for separation of duties. Chapter 6 compares the expressive power of ATAM and its variations. Chapter 7 compares the expressive power of TAM and its variations. Finally, chapter 8 enumerates the contributions of this dissertation and discusses future research directions.

Chapter 2

Access Control Models: TAM and its Variations

This chapter gives a brief background on access control models. In particular, we review the Typed Matrix Model (TAM) defined in [San92b] and few other models which are defined by either posing some restrictions on TAM or by extending TAM. Section 2.1 explains TAM. Section 2.1.1 describes Single-Object Typed Access Matrix Model (SOTAM). Section 2.1.2 defines Binary TAM (BTAM) and Unary TAM (UTAM). Section 2.2 describes Augmented TAM (ATAM) and section 2.2.1 defines Augmented SOTAM (SO-ATAM). Finally section 2.2.2 defines Augmented Unary TAM (U-ATAM) and Augmented Binary TAM (B-ATAM).

2.1 Typed Access Matrix (TAM) Model

In this section we review the definition of TAM, which was introduced by Sandhu in [San92b]. The principal innovation of TAM is to introduce strong typing of subjects and objects, into the access matrix model of Harrison, Ruzzo and Ullman [HRU76] (i.e., each subject or object is created to be of a particular type which thereafter does not change). This innovation is adapted from Sandhu's Schematic Protection Model [San88b], and its extension by Ammann and Sandhu [AS92a].

As one would expect from its name, TAM represents the distribution of rights in the system by an access matrix. The matrix has a row and a column for each subject and a column for each object. Subjects are also considered to be objects.

The $[X, Y]$ cell contains rights which subject X possesses for object Y.

Each subject or object is created to be of a specific type, which thereafter cannot be changed. It is important to understand that the types and rights are specified as part of the system definition, and are not predefined in the model. The *security administrator* specifies the following sets for this purpose:

- a finite set of access *rights* denoted by R, and
- a finite set of *object types* (or simply *types*), denoted by T.

For example, $T = \{user, so, file\}$ specifies there are three types, viz., user, security-officer and file. A typical example of rights would be $R = \{r, w, e, o\}$ respectively denoting read, write, execute and own. Once these sets are specified they remain fixed, until the security administrator changes their definition. It should be kept in mind that TAM treats the security administrator as an external entity, rather than as another subject in the system.

The *protection state* (or simply *state*) of a TAM system is given by the four-tuple (OBJ, SUB, t, AM) interpreted as follows:

- OBJ is the set of *objects*.
- SUB is the set of *subjects*, $SUB \subseteq OBJ$.
- $t : OBJ \rightarrow T$, is the *type function* which gives the type of every object.
- AM is the *access matrix*, with a row for every subject and a column for every object. The contents of the $[S, O]$ cell of AM are denoted by $AM[S, O]$. We have $AM[S, O] \subseteq R$.

The rights in the access matrix cells serve two purposes. First, presence of a right, such as $r \in AM[X,Y]$ may authorize X to perform, say, the read operation on Y. Second, presence of a right, say $o \in AM[X,Y]$ may authorize X to perform some operation which changes the access matrix, e.g., by entering r in $AM[Z,Y]$. In other words, X as the owner of Y can change the matrix so that Z can read Y.

The protection state of the system is changed by means of TAM commands. The security administrator defines a finite set of TAM commands when the system is specified. Each TAM *command* has one of the following formats.

command $\alpha(X_1 : t_1,\ X_2 : t_2,\ \ldots,\ X_k : t_k)$
  if $r_1 \in [X_{s_1}, X_{o_1}] \wedge r_2 \in [X_{s_2}, X_{o_2}] \wedge \ldots \wedge r_m \in [X_{s_m}, X_{o_m}]$
  then op_1; op_2; $\ldots$; op_n
end

or

command $\alpha(X_1 : t_1,\ X_2 : t_2,\ \ldots,\ X_k : t_k)$
  op_1; op_2; $\ldots$; op_n
end

Here α is the *name* of the command; $X_1,\ X_2,\ \ldots,\ X_k$ are *formal parameters* whose types are respectively $t_1,\ t_2,\ \ldots,\ t_k$; $r_1,\ r_2,\ \ldots,\ r_m$ are rights; and $s_1,\ s_2,\ \ldots,\ s_m$ and $o_1,\ o_2,\ \ldots,\ o_m$ are integers between 1 and k. Each op_i is one of the *primitive operations* discussed below. The predicate following the **if** part of the command is called the *condition* of α, and the sequence of operations op_1; op_2; $\ldots$; op_n is called the *body* of α. If the condition is omitted the command is said to be an *unconditional command*, otherwise it is said to be a *conditional command*. Note that a disjunctive condition, such as $r_1 \in [X_{s_1}, X_{o_1}] \vee r_2 \in [X_{s_2}, X_{o_2}]$, can be simulated by two separate commands with conditions $r_1 \in [X_{s_1}, X_{o_1}]$ and $r_2 \in [X_{s_2}, X_{o_2}]$ respectively. Hence, for simplicity, we define a condition to be a conjunction without any loss of generality.

A TAM 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.

There are six primitive operations in TAM, grouped into two classes, as follows.

enter r **into** $[X_s, X_o]$
create subject X_s **of type** t_s
create object X_o **of type** t_o

(a) Monotonic Primitive Operations

delete r **from** $[X_s, X_o]$
destroy subject X_s
destroy object X_o

(b) Non-Monotonic Primitive Operations

It is required that s and o are integers between 1 and k, where k is the number of parameters in the TAM command in whose body the primitive operation occurs.

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 **enter** operation is said to be *monotonic* because it only adds and does not remove from the access matrix. 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. Because **delete** (potentially) removes a right from the access matrix it is said to a *non-monotonic* operation.

The **create subject** and **destroy subject** operations make up a similar monotonic versus non-monotonic pair. The **create subject** operation requires that the subject being created has a unique identity different not only from existing subjects, but also different from all subjects that have ever existed thus far.[1] The **destroy**

[1]There is some question about whether or not creation should be treated as a monotonic opera-

subject operation requires that the subject being destroyed currently exists. Note that if the pre-condition for any **create** or **destroy** operation in the body is false, the entire TAM command has no effect. The **create subject** operation introduces an empty row and column for the newly created subject into the access matrix. The **destroy subject** operation removes the row and column for the destroyed subject from the access matrix. The **create object** and **destroy object** operations are much like their subject counterparts, except that they work on a column-only basis.

Two examples of TAM commands are given below.

- **command** *create-file*$(U : user, F : file)$
  create object F **of type** $file$;
  enter *own* **in** $[U, F]$;
  end

- **command** *transfer-ownership*$(U : user, V : user, F : file)$
  if $own \in [U, F]$ **then**
  delete *own* **from** $[U, F]$;
  enter *own* **in** $[V, F]$;
  end

The first command authorizes users to create files, with the creator becoming the owner of the file. The second command allows ownership of a file to be transferred from one user to another.

---

tion. The fact that creation consumes a unique identifier for the created entity, which cannot be used for any other entity thereafter, gives it a non-monotonic aspect. In our work we have always treated creation as a monotonic operation. This is principally because systems without creation are not very interesting. Treating creation as non-monotonic would therefore make the class of monotonic systems uninteresting. Monotonic systems with creation are, however, an important and useful class of systems.

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

1. A set of rights R.
2. A set of types T.
3. A set of state-changing 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.

2.1.1 The Single-Object TAM (SOTAM) Model

SOTAM is a simplified version of TAM, with the restriction that all primitive operations in the body of a command are required to operate on a single object. An object is represented as a column in the access matrix. Similarly, when a subject is the "object" of an operation, that subject is viewed as a column in the access matrix. SOTAM stipulates that all operations in the body of a command are confined to a single column.

Given below are two commands. The command *review* is a SOTAM command as the body of the command has operations on a single object. The command *share-ownership* is not a SOTAM command as the body of the command has operations on two objects.

command $review(S : sci, O : doc, SO : sec \Leftrightarrow off, PO : pat \Leftrightarrow off)$
  if $own \in [S, O]$ **then**
    enter $review$ **in** $[SO, O]$;
    enter $review$ **in** $[PO, O]$;
end

command *share-ownership*$(S_1 : s_1, S_2 : s_2, O_1 : o_1, O_2 : o_2)$
  if $own \in [S_1, O_1] \wedge own \in [S_2, O_2]$ **then**
    enter own **in** $[S_2, O_1]$;
    enter own **in** $[S_1, O_2]$;
end

To appreciate the motivation for SOTAM consider the usual implementation of the access matrix by means of access control lists (ACL's). Each object has an ACL associated with it, representing the information in the column corresponding to that object in the access matrix. The restriction of SOTAM implies that a single command can modify the ACL of exactly one object. These modifications can therefore be done at the single site where the object resides. This greatly simplifies the protocols for implementing the commands. In particular, we do not need to be concerned about coordinating the completion of a single command at multiple sites. There is therefore no need for a distributed two-phase commit for SOTAM commands. Further details on an implementation outline of SOTAM are given in [SS92b]. One of the results proved in this thesis is that SOTAM is strongly equivalent to TAM in terms of expressive power (see section 7.1).

2.1.2 Unary TAM (UTAM), Binary TAM (BTAM) and KTAM

Unary TAM is same as TAM except that the commands have the ability to test for at most one cell. The following is an example of a typical UTAM command.

command $utam(X_1 : t_1,\ X_2 : t_2,\ \ldots,\ X_k : t_k)$
  if $P1 \subseteq [X_{s_a}, X_{o_b}]$
  then $op_1;\ op_2;\ \ldots;\ op_n$
end

Here *utam* is the *name* of the UTAM command; $X_1,\ X_2,\ \ldots,\ X_k$ are *formal parameters* whose types are respectively $t_1,\ t_2,\ \ldots,\ t_k$; $r_1,\ r_2,\ \ldots,\ r_m$ are rights; and $s_1,\ s_2,\ \ldots,\ s_m$ and $o_1,\ o_2,\ \ldots,\ o_m$ are integers between 1 and k. The predicate of the command tests only one cell represented by $[X_{s_a}, X_{o_b}]$ and $P1$ is a set of rights. The body of the command has *primitive operations* which can modify multiple cells. The command *review* given earlier is an UTAM command as it only tests one cell and the command *share-ownership* given earlier is not an UTAM command as it tests for two cells.

Binary TAM is same as TAM except that the commands have the ability to test for at most two cells. The following is an example of a typical BTAM command.

command $btam(X_1 : t_1,\ X_2 : t_2,\ \ldots,\ X_k : t_k)$
  if $P1 \subseteq [X_{s_a}, X_{o_b}] \wedge P2 \subseteq [X_{s_c}, X_{o_d}]$
  then $op_1;\ op_2;\ \ldots;\ op_n$
end

Here *btam* is the *name* of the BTAM command; $X_1,\ X_2,\ \ldots,\ X_k$ are *formal parameters* whose types are respectively $t_1,\ t_2,\ \ldots,\ t_k$; $r_1,\ r_2,\ \ldots,\ r_m$ are rights; and $s_1,\ s_2,\ \ldots,\ s_m$ and $o_1,\ o_2,\ \ldots,\ o_m$ are integers between 1 and k. The predicate of the command tests at most two cells represented by $[X_{s_a}, X_{o_b}]$ and $[X_{s_c}, X_{o_d}]$. $P1$ and $P2$ both represent a set of rights. The body of the command *btam* has *primitive operations* which can modify multiple cells. The commands *review* and *share-ownership* given earlier are BTAM commands as they test for one cell and two cells respectively.

Just as Unary TAM and Binary TAM are defined as TAM with only the ability to test for at most one and two cells respectively, we define KTAM to be same as TAM with the ability to test for at most K cells.

In section 7.2 we prove that UTAM and BTAM are weakly equivalent to TAM in expressive power, and we also conjecture that UTAM and BTAM (and in general KTAM) are not strongly equivalent to TAM in terms of expressive power. [2]

2.2 Augmented TAM (ATAM)

ATAM was defined in [AS92b] to be TAM extended with ability to test for the absence of a right in a cell of the access matrix. In other words, a test of the form $r_i \notin [X_{s_i}, X_{o_i}]$ may be present in the condition part of ATAM commands. In this thesis we show that adding testing for absence of rights does increase the expressive power of TAM. We prove in chapter 4 that ATAM is weakly equivalent but not strongly equivalent to TAM in terms of expressive power. We also show in chapter 5 (informally) that dynamic separation of duties requires the ability to test for absence of access rights. In particular, we show how transaction control expressions [San88c] can be specified in ATAM.

2.2.1 Augmented SOTAM (SO-ATAM)

Augmented SOTAM is same as SOTAM with the additional ability to test for absence of rights. We prove in chapter 6 that SO-ATAM is strongly equivalent to ATAM in terms of expressive power.

[2] Chapter 3 gives formal definitions of strong and weak equivalence. Intuitively, weak equivalence is meant to be theoretical equivalence and strong equivalence is meant to be practical equivalence. Strong equivalence implies weak equivalence but not vice-versa. Whenever we say equivalence, we understand the equivalence to be strong unless otherwise specified.

SOTAM (Modifies one column)	SO-ATAM (SOTAM with the ability to test for absence of rights)
UTAM (Tests at most one cell)	U-ATAM (UTAM with the ability to test for absence of rights)
BTAM (Tests at most two cells)	B-ATAM (BTAM with the ability to test for absence of rights)
KTAM (Tests at most K cells)	K-ATAM (KTAM with the ability to test for absence of rights)

Table 2.1: Variations of TAM and ATAM

2.2.2 Augmented Unary TAM (U-ATAM), Augmented Binary TAM (B-ATAM) and K-ATAM

Augmented Unary TAM (U-ATAM) is same as ATAM except that the commands have the ability to test for at most one cell. The earlier UTAM command *utam* with its predicate having the ability to test for absence of rights is an example of an U-ATAM command.

Similarly, augmented Binary TAM (B-ATAM) is same as ATAM except that the commands have the ability to test for at most two cells. The earlier BTAM command *btam* with its predicate having the ability to test for absence of rights is an example of an B-ATAM command. We prove in chapter 6, U-ATAM (and hence B-ATAM) is strongly equivalent to ATAM in terms of expressive power.

Table 2.1 lists different variations of TAM and ATAM defined in this chapter. In this thesis, we compare the expressive power of TAM and ATAM. We also compare the expressive power of all the models given in table 2.1 with respect to TAM and ATAM. The expressive power results proved in this thesis are summarized in table 2.2.

Expressive Power of TAM and ATAM
TAM $\not\equiv_{strongly}$ ATAM
TAM $\equiv_{weakly}$ ATAM
Expressive Power of ATAM and its Variations
ATAM $\equiv_{strongly}$ SO-ATAM
ATAM $\equiv_{strongly}$ U-ATAM
Expressive Power of TAM and its variations
TAM $\equiv_{strongly}$ SOTAM
TAM $\equiv_{weakly}$ UTAM
TAM $\not\equiv_{strongly}$ UTAM (conjecture)
TAM $\not\equiv_{strongly}$ KTAM (conjecture)

Table 2.2: Summarized Results

Chapter 3

Expressive Power of Access Control Models

As mentioned in chapter 1, access control models should have adequate expressive power, i.e., they should state policies of practical interest. One measure of an expressive power of a model is relative and is measured in terms of the expressive power of another model. Usually when a model is said to be equivalent to another in expressive power, it has simply been perceived that those two models enforce the same policies. In this chapter we give a rigorous formalism to compare the expressive power of two models.

Section 3.1 defines an access control model. Section 3.2 defines the notion of simulation between two systems. Finally section 3.3 gives the definitions to compare the expressive power of two models.

3.1 Access Control Models

In this thesis we will be comparing the expressive power of access control models. In general, we compare the expressive power between the Augmented Typed Access Matrix Model (ATAM) and the models obtained by enforcing some restrictions on it. ATAM represents the distribution of rights by an access matrix and this matrix is changed by means of state-changing commands.

In this section, we formally define what an access control model is. Most of the definitions defined in this section were informally introduced in chapter 2.

Definition 1 (Subject) *: A subject is anything, that can possess access rights in the system.*

A subject is usually a a process (program or application) executing on behalf of a user in the system. A subject can also be a passive entity, such as a directory.

Definition 2 (Object) *: An object is anything on which a subject can perform operations.*

Usually objects are passive, for example files and directories are objects. A subject can also be an object, e.g., a process may have suspend and resume operations executed on it by some other process. In general, subjects are viewed as a subset of the objects.

Definition 3 (Rights) *: Each system has a set of rights, R. The presence or absence of a* right *allows a subject to perform some operations (which may or may not change the access matrix).*

The access rights of subjects to objects are conceptually represented by an access matrix.

Definition 4 (Access matrix) *: The* access matrix *has a row and a column for each subject, and a column for each object. The [X,Y] cell contains rights which subject X possesses for object Y. Every subject is also an object but not vice versa.* [1]

Definition 5 (Strong Typing) *: Each subject or object is created to be of a specific* type*, which cannot be changed.*

[1] In some cases it is convenient to assume every object is also a subject. This can be done without loss of generality as TAM subjects are not necessarily active entities.

The HRU model can be considered to have all objects and subjects of a type and hence it still qualifies under the definition of strong typing.

Definition 6 (State) *: The* state *of a system is given by the four tuple (OBJ, SUB, t, AM) interpreted as follows:*

1. *OBJ is the set of objects.*
2. *SUB is the set of subjects, SUB* $\subseteq$ *OBJ.*
3. *t: OBJ* $\rightarrow$ *T, is the type function which gives the type of every object.*
4. *AM is the access matrix, with a row for every subject and a column for every object.*

The state of the system is changed by means of commands.

Definition 7 (State Changing Commands or Command Definition) *: A state changing command has the following format:*

command $\alpha(X_1 : t_1, X_2 : t_2, \ldots, X_k : t_k)$
  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 $X_1, X_2, ..., X_k$ are the formal parameters. The second line is the predicate and is called the condition of the command. The condition tests for presence/absence of rights in the access matrix. The third line is called the body of α which has a sequence of primitive operations; enter, delete, destroy and create operations (as defined in chapter 2). Commands can also be unconditional.

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.

Definition 8 (Command Invocation) *: A command is said to be* invoked *if it is executed with some actual parameters.*

From now on whenever we say a *command*, we simply mean a command with formal parameters and when we say an *invoked command*, we mean a command with some actual parameters.

Definition 9 (Scheme) *: The rights, types and the state changing commands combined are called the* scheme. *These do not change.*

Definition 10 (System) *: A scheme together with an initial state is a* system.

The system evolves as the state of the system keeps changing through the state changing commands.

Definition 11 (Access Control Models) *: An* Access control model *is defined by enforcing restrictions on the general structure of commands, given in definition 7.*

The Typed Access Matrix Model (TAM) and other models defined in chapter 2 are examples of access control models.

In order to prove that one model is equivalent to another, we need to prove that for every system in one model, there is an equivalent system in another and vice versa. To prove the equivalence of two systems and two models, we need to formalize the notion that one system simulates another. The next section defines the notion of one system simulating another.

3.2 Simulation of Systems

This section first describes some definitions which are used in defining the semantics of one system simulating another and it then defines what it is meant by one system simulating another

Definition 12 (Original and Simulation Systems) *: The system that is being simulated is the* original system*, denoted A, and the system implementing the simulation, is called the* simulation system*, denoted B.*

Definition 13 (Mapping Between Schemes) *: Scheme A and scheme B should satisfy the following in order for B to Map A:*

(1) $R_A \subseteq R_B$

(2) $T_A \subseteq T_B$

(3) ${c_A}^i \mapsto \{T_i, F_i\}$ *and the* $\mapsto$ *function is Disjoint and Proper.*

Condition (1) indicates that every right in the original system is also in the simulating system. The additional rights in B are usually present due to bookkeeping details in the simulation of A by B. The intuition behind this is that if some right/rights in B represent a right a of A, then those rights can be replaced by the same right a in B. Hence, to make simulations easier to understand it is a reasonable assumption to represent rights in A with the same rights in B.

Condition (2) indicates that every type defined in the original system is also defined in the simulating system and there might be additional types defined in B due to some bookkeeping. The intuition behind these is that if a subject S_1 of type s_1 in A is represented by one or more subjects of a type or types, then those subjects can be represented by S_1. To make simulations easier to understand, it is reasonably to assume that the subject is represented by S_1 rather than some other subjects.

Condition (3) indicates the following:

- Every command ${c_A}^i$ of A is mapped to two partial orders of commands T_i and F_i in B[2]. The partial order T_i corresponds to the successful execution of the mapped command ${c_A}^i$ and the other one F_i corresponds to the unsuccessful execution of ${c_A}^i$. The commands in the partial orders T_i and F_i are given by commands ${C_1}^{it}, {C_2}^{it}, \ldots, {C_p}^{it}$ and commands ${C_1}^{if}, {C_2}^{if}, \ldots, {C_q}^{if}$ respectively (where i corresponds to i of command ${c_A}^i$ and t, f correspond to T_i and F_i respectively). The partial orders T_i and F_i specify the order of commands to be executed.

- For each parameter of a command in A, its corresponding parameter in all the mapped commands in B, i.e., if a command in ${c_A}^i$ has a parameter S_1, then the mapping indicates which parameter in all the commands mapped from ${c_A}^i$ is mapped to S_1.

- A successful execution of a command ${c_A}^i$ in B is completed only if one of the commands in T_i which do not have any successors execute. Similarly an unsuccessful execution of a command ${c_A}^i$ in B is completed only if one of the commands in F_i which do not have any successors execute.

- The mapping of commands is also Disjoint and Proper. The below two definitions explain what Disjoint and Proper mapping mean.

[2] We actually could map every command of A to a single partial order rather than two, but to make it easier to understand we map every command of A to two partial orders. In most of the simulations (encountered in this thesis, and in general) a command of A is mapped to a single partial order which indicates successful execution. In these cases an unsuccessful execution does not require any commands in the simulation. In the more general case the predicate in the original command is tested by multiple commands in the simulation. If this test eventually fails, we need the F_i partial order to restore the state prior to the testing.

Definition 14 (Disjoint Mapping) *: The mapping function $\mapsto$ is Disjoint for all commands ${c_A}^i$ and ${c_A}^j$ of system A, such that ${c_A}^i \neq {c_A}^j$ and ${c_A}^i \mapsto \{T_i, F_i\}$, ${c_A}^j \mapsto \{T_j, F_j\}$, there do not exist any two commands from the partial orders in T_i, F_i and T_j, F_j which are identical.*

In other words, the mapping is said to be *disjoint* if the partial order of commands of B mapped from one command in A is disjoint from any other partial order of commands of B which are mapped from a different command of A. The intuition behind this condition is that in A, there wouldn't be two different commands with identical parameters, predicate and body (even if there are, these will be mapped to disjoint commands).

Definition 15 (Proper Mapping) *: For all commands ${c_A}^i$ of system A such that ${c_A}^i \mapsto \{T_i, F_i\}$, if there exists a parameter P in commands in the partial orders in T_i, F_I, which is not a parameter in ${c_A}^i$, then $P \in T_B$-T_A.*

The definition indicates that whenever there is a parameter P in a command belonging to a partial order in T_i or F_i which is mapped from a command ${c_A}^i$, and P is not a parameter in ${c_A}^i$, then P should belong to T_B-T_A. In simple words the above definition indicates that mapping is proper if all the additional parameters in B belong to T_B-T_A. This simply means that any parameter in a command of B which is not in a command mapped from A, should be one of the additional types of B. [3]

Definition 16 (Conflicting Commands) *: Two invoked commands are said to* conflict *if they both modify the same cell, or if one of them tests a cell and the other modifies the same cell.*

[3] As rights are not parameters we do not address rights here. But we address this issue with respect to rights in the definition of simulation given later in this section, where we make sure that rights not entered/deleted in A are also not entered/deleted in B.

Definition 17 (History) *: If $C = \{C_1, C_2, \ldots, C_n\}$ is a set of invoked commands. A* history *H over C is a partial ordering relation $<_H$ on C where for any two conflicting commands $C_i, C_j \in C$, either $C_i <_H C_j$ or $C_j <_H C_i$.*

The definition for history says that the ordering of every pair of conflicting invoked commands is determined by $<_H$, i.e., if $C_i <_H C_j$, this indicates that execution of C_i is complete before the start of execution of C_j. A history is not a total order as the order of non-conflicting invoked commands is not important (i.e., we can assume that non-conflicting invoked commands can execute concurrently). A history indicates the order in which the operations of the invoked commands were executed relative to each other.

We now use the following commands to illustrate the definitions given in this section. Say command $c1$ of A is mapped to $C1$ and $C2$ of B and command $c2$ is mapped to $C3$ and $C4$. All the commands given below are invoked commands.

command *c1* $(S_1 : s, O_1 : o, S_2 : s, O_2 : o)$
if $a \notin [S_1, O_1] \wedge c \in [S_2, O_2]$**then**
    enter a in $[S_1, O_1]$;
    enter b in $[S_1, O_1]$;
    enter b in $[S_2, O_2]$;
end

command *c2* $(S_1 : s, O_1 : o, S_2 : s, O_2 : o)$
if $a \notin [S_1, O_2] \wedge a \notin [S_2, O_1] \wedge a \in [S_1, O_1] \wedge a \in [S_2, O_2]$**then**
    enter c in $[S_2, O_2]$;
    enter a in $[S_1, O_2]$;
    enter a in $[S_2, O_1]$;
end

command *C1* $(S_1 : s, O_1 : o)$

if $a \notin [S_1, O_1]$**then**

enter a in $[S_1, O_1]$;

enter b in $[S_1, O_1]$;

end

command *C2* $(S_2 : s, O_2 : o)$

if $c \in [S_2, O_2]$**then**

enter b in $[S_2, O_2]$;

end

command *C3* $(S_1 : s, O_1 : o, S_2 : s, O_2 : o)$

if $a \in [S_1, O_1] \wedge a \in [S_2, O_2]$**then**

enter c in $[S_2, O_2]$;

end

command *C4* $(S_1 : s, O_1 : o, S_2 : s, O_2 : o)$

if $a \notin [S_1, O_2] \wedge a \notin [S_2, O_1]$**then**

enter a in $[S_1, O_2]$;

enter a in $[S_2, O_1]$;

end

Example of histories are $(C1C2)(C3C4)$ and $(C4C1)C3C2$. Here the invoked commands in the parenthesis indicate that they could be executed in any order. Since $C1, C2$ and $C4$ do not conflict with each other and only $C1$ and $C2$ conflict with $C3$, $(C1C2)(C3C4)$ and $(C4C1)C3C2$ are histories. But $(C1C3)(C2C4)$ is not a history as it doesn't specify the order of two conflicting invoked commands $C1$ and $C3$.

Definition 18 (Equivalence of Histories of the same system) *: We define two histories H and H' of B to be equivalent iff:*

1. *they are defined over the same set of invoked commands and*

2. *they order conflicting invoked commands in the same way.*

The idea underlying this definition is that the outcome of a concurrent execution of invoked commands depends on the relative order of conflicting invoked commands. To see this, observe that executing two non-conflicting invoked commands in either order has the same effect on the protection state. Conversely, the protection state depends on the order of execution of any two conflicting operations.

For example the history $(C1C2)(C3C4)$ is equivalent to history $(C2C1)(C3C4)$ as the commands $C1, C2$ and $C4$ do not conflict with each other and only $C1$ and $C2$ conflict with $C3$ and they both execute before $C3$ in both the histories. The histories $(C1C2)(C3C4)$ and $C3(C1C2C4)$ are not equivalent as in one history $C3 < C1$ and in the other $C1 < C3$, and $C3$ conflicts with $C1$.

Definition 19 (Concurrent History) *: If in a history H of B, if there exists commands Cp^i, Cq^i in either F_i or T_i and if there exists commands Cr^j, Cs^j in either F_j or T_j, and if $Cp^i <_H Cr^j$ and $Cs^j <_H Cq^i$, then the history H is called* Concurrent.

The execution of invoked commands in B is said to be *Concurrent*, if B first executes an invoked command from a partial order (either F_i or T_i) and if some other invoked command of a different partial order (either F_j or T_j) executes before the execution of all the invoked commands from partial order represented by either F_i or T_i. In simple words, a history is concurrent if there is interleaving of commands from two different partial orders.

The history $C1C3(C2C4)$ is a concurrent history as $C1 < C3$, $C3 < C2$, and due to the fact that $C1$ and $C2$ are mapped from $c1$, and $C3$ is mapped from $c2$.

Definition 20 (Serial History) *: A history H in B is said to be* Serial *if for all invoked commands C^i of H belonging in either F_i or T_i, and for all invoked commands C^j of H belonging in either F_j or T_j, if there exists an $C^i <_H C^j$ then all $C^i <_H C^j$.*

A history in B is said to be *serial* if it does not have any interleaving of invoked commands from two different partial orders.

An example of a serial history is $(C1C2)(C3C4)$ as $C1$ and $C2$ execute before any of $C3$ and $C4$.

Definition 21 (Serializable History) *: A history H of B is said to be serializable if it is equivalent to some serial history.*

The history $(C1C4C2)C3$ is a serializable history as it is equivalent to the serial history $(C1C2)(C3C4)$. The history $C1C3(C2C4)$ is not a serializable history as it does not have an equivalent serial history.

Definition 22 (Complete History) *: A history H in B is said to be* complete *if for all commands in H, if there exists a command C^i in H which belongs to either F_i or T_i, then there exists a command C^j (belonging to either F_i or T_i) in H, which do not has a successor in either F_i or T_i.*

The definition indicates that if an invoked command belonging to a partial order (either T_i or F_i) is in the history, then there should be an invoked command in that history which do not has any successors in either T_i or F_i. In simple words, a history in B is said to be *complete* if it doesn't have any partial execution of invoked commands.

Examples of complete histories are $(C1C2)$ and $C1C3(C2C4)$. The history $C1(C3C4)$ is not complete as $C2$ is not in the history.

Definition 23 (Completion state) *: A state in the history H of B is said to be* complete *if that history is complete.*

The state at the end of history $C1C3(C2C4)$ is a complete state.

Definition 24 (Intermediate state) *: A state in the history H of B is said to be* intermediate *if the history is not complete.*

The state at the end of history $C1C3C2$ is an intermediate state as $C4$ is not in the history.

Definition 25 (Equivalence of Histories of A and B) *: A history H of A and a serial complete history H′ of B are equivalent if for all commands ${c_A}^i$ and ${c_B}^j$ in H, such that ${c_A}^i <_H {c_A}^j$, then in H′, for all commands $C^i \in T^i$ and for all commands $C^j \in T^j$, $C^i <_{H'} C^j$.*

A history H of A and a serial complete history H′ of B are equivalent if the order of commands in H is same as the order of their mapping commands in H′.

The history $c1c2$ of A is equivalent to history $(C1C2)(C3C4)$ of B.

Definition 26 (Complete Extension) *: A* complete extension *is defined as a history obtained by extending an incomplete history to a complete history.*

3.2.1 Formal Definition of Simulation

This section defines what is meant by one system simulating another and it also explains the semantics of this definition.

Definition 27 (Simulation) *: A system B* simulates *system A iff the following is satisfied:*

1. *Scheme of A maps to scheme of B (def 13).*

2. *$\forall$ histories H of A, $\exists$G a serial history of B, and G is equivalent to H (def 25).*

3. *$\forall$ complete histories G of B, $\exists$ complete serial history G$'$ of B, such that G$'$ is equivalent to G, and $\exists$ H of A such that G$'$ is equivalent to H.*

4. *$\forall$ incomplete histories U of B, $\forall$ complete extensions U$'$ of U, $\exists$ complete serial history G$'$ of B, such that G$'$ is equivalent to U$'$, and $\exists$ H of A such that G$'$ is equivalent to H.*

5. *$\forall$ incomplete histories U of B, $\exists$ complete extension U$'$ of U.*

6. *The following condition for the states should hold.*

   At completion states $AM[X,Y]_A = AM[X,Y]_B \cap R_A$

   At intermediate states $AM[X,Y]_A^{pre} \supseteq AM[X,Y]_B \cap R_A$ *or* $AM[X,Y]_A^{post} \supseteq AM[X,Y]_B \cap R_A$

   where $AM[X,Y]_A^{pre}$ *is the state in A before an execution of an invoked command and* $AM[X,Y]_A^{post}$ *is the state in A after an execution of that particular invoked command.*

The above conditions should be satisfied in order for B to simulate A. We now explain the intuition behind these conditions.

The first one indicates that scheme of B should simulate scheme of A according to definition 13. The intuition behind these is given earlier.

The second condition indicates that for all histories of A, there should exist at least one equivalent serial history of B. This is obvious from the fact that in order for B to simulate A, B should be able to do whatever A can and so for every history

in A, there should be at least one equivalent serial history in B (and there might be many other histories in B which are all equivalent to that serial history).

The third condition indicates that all complete histories of B should have an equivalent serial history in B, which is equivalent to some history of A. The need for this condition is from the fact that in order for B to simulate A, it shouldn't be able to do more than what A can.

The fourth condition indicates that for all incomplete histories of B and for all their complete extensions, there should exist at least one equivalent serial history in B, which is equivalent to a history of A. The reason behind this condition is the same as the third condition except that it accounts for all the incomplete histories of B. i.e., it makes sure that there doesn't exist any incomplete history (or its complete extension) in B which doesn't have an equivalent history in A.

The fifth condition also ensures that in order for B to simulate A, there cannot be any deadlocks in B (i.e., all incomplete histories in B can be extended to complete histories).

The final condition gives the relationship between the states of A and B at both intermediate and completion states. The intuition for having this condition for intermediate and completion states is given below.

At completion states, the matrix of A and the matrix of B should be identical with respect to the rights and subjects of A. i.e., for all cells $AM[X,Y]$ of both A and B, the contents should be same with respect to the rights defined in A and might not with additional rights defined in B. This should be obvious from the fact that A and B should have the same behavior with respect to rights defined for system A.

At intermediate states the following condition should hold true:

$AM[X,Y]_A^{pre} \supseteq AM[X,Y]_B \cap R_A$ or $AM[X,Y]_A^{post} \supseteq AM[X,Y]_B \cap R_A$

where $AM[X,Y]_A^{pre}$ is the state in A before an execution of a command and $AM[X,Y]_A^{post}$ is the state in A after an execution of that particular command. This condition ensures that the following hold:

1. No right defined in A is entered/deleted in B which is not entered/deleted in A. The reason for this is obvious from the fact that B shouldn't be able to reach a state which cannot be reached in A, even if this is an intermediate state in B.

2. Rights which cannot coexist in A, shouldn't coexist in an intermediate state in B (even though they might not coexist at the completion state in B). This would take care of the fact that B cannot reach an intermediate state that A would not allow. Consider for example that A checks for a right a in a cell and if so, it deletes a and gets right b in return. In A, if the rights a and b do not coexist, then this condition would ensure that even at intermediate states of B these rights do not coexist. If it was not for this condition, B could have rights a and b together in an intermediate state. But in A this is not allowed in any state and hence this should not be allowed in B also.

The next section formalizes the notion of expressive power.

3.3 Expressive Power of Models

This section gives definitions to compare the expressive power of two models.

Definition 28 (Scheme Based Simulation) *: If the scheme for the simulation system is derived from the scheme of the original system, then we call the simulation to be* scheme based*.*

The above definition indicates that the scheme of the simulation system can be derived only from the scheme of the original system and is independent of the initial state of the original system.

Definition 29 (Bounded Simulation) *: To simulate an execution of a command in the original system, the simulating system can execute each of its mapping commands with at most one set of actual parameters.*

The above definition indicates that if a command executes in A, then that command is simulated in B by executing each command from a mapping partial order with at most one set of actual parameters. This definition indicates that bounded simulation would not allow a command being simulated by some unbounded number of commands. This definition is different from one which enforces every command to execute at most once as this stricter definition wouldn't allow commands to execute more than once with the same parameters and this definition wouldn't apply to monotonic commands. To allow such cases, this definition allows a mapping command in B to execute at most with one set of actual parameters.

Finally we formalize the notion of expressive power.

Definition 30 (Weakly Expressive) *: Model β is* weakly expressive *as model α iff the following holds: For every system A in model α there exists a system B in model β such that system B is a scheme based simulation of system A.*

Definition 31 (Strongly Expressive) *: Model β is* strongly expressive *as model α iff the following holds: For every system A in model α there exists a system B in model β such that system B is a bounded and scheme based simulation of system A.*

Definition 32 (Strong Equivalence of two Models) *: Model α is* strongly equivalent *to model β iff model α is as strongly expressive as model β and model β is as strongly expressive as model α*

Definition 33 (Weak Equivalence of two Models) *: Model α is* weakly equivalent *to model β iff both α and β are weakly expressive as the other.*

We will usually use the term equivalent to mean strongly equivalent. Where the context requires us to carefully distinguish between strong and weak equivalence we will be appropriately precise.

The definition of strong equivalence requires that a command in a system be simulated by a constant number of commands (rather than some arbitrary number of commands which depend on the state). This requirement is usually needed in real world practical applications. Hence we understand strong equivalence to be a *practical equivalence*. Similarly we understand weak equivalence to be a *theoretical equivalence*. Any two models which are strongly equivalent are also weakly equivalent, but two models which are weakly equivalent may or may not be strongly equivalent. In this thesis we will show there are models which are strongly equivalent and models which are only weakly equivalent. A central result of this thesis is that TAM and ATAM are only weakly equivalent. The construction which proves the weak equivalence of TAM and ATAM illustrate the fact that it is important that models be strongly equivalent rather than weakly equivalent. Hence in this thesis we define two types of equivalence between models.

Chapter 4

Expressive Power of ATAM and TAM

Recall that ATAM is same as TAM with the additional ability to test for absence of access rights. In this chapter, two important results are proved regarding TAM and ATAM. The first result is the weak equivalence of TAM and ATAM and this result has appeared in [SG93b]. The equivalence of TAM and ATAM is weak due to the fact that the simulation used in proving the equivalence is not bounded. The second result is the strong non-equivalence of TAM and ATAM. This result indicates that TAM and ATAM are not strongly equivalent as there can be a system in ATAM for which there cannot be a bounded simulation in TAM. This chapter is organized as follows. Section 4.1 proves the weak equivalence of TAM and ATAM and section 4.2 proves the strong non-equivalence of TAM and ATAM.

4.1 Weak Equivalence of ATAM and TAM

In this section we give a construction to show the weak equivalence of TAM and ATAM. We first show in section 4.1.1 that ATAM schemes without **create** or **destroy** operations can be reduced to TAM schemes. We than show, in section 4.1.2, how ATAM schemes with just **create** and **destroy** commands can be simulated in TAM. Finally in section 4.1.3 we give a procedure which converts any given ATAM scheme into an equivalent TAM scheme. This procedure is not proper and is given here for simplicity. We illustrate at the end of this section how the procedure can be made

proper.

4.1.1 Equivalence Without Create or Destroy Operations

We now prove the equivalence of TAM and ATAM in the absence of create and destroy operations. This is done by giving a construction to construct a TAM system that can simulate any given ATAM system. Recall that ATAM extends TAM by allowing commands to test for absence of access rights in the condition part. Thus, TAM is a restricted version of ATAM. To establish equivalence we therefore need to show that every ATAM system can be simulated by a TAM system.

The basic idea in the construction is to represent the absence of rights in the ATAM system by the presence of *complementary rights* in the TAM system. Suppose that the given ATAM system has set of rights R. In the TAM simulation we include the rights R, as well as the complementary rights $\bar{R} = \{\bar{x} \mid x \in R\}$. The construction will ensure that

$$x \in [S_i, O_j] \Leftrightarrow \bar{x} \notin [S_i, O_j]$$

The initial state of the TAM access matrix has all the rights of the initial matrix of ATAM, as well as all the complementary rights implied by the above equation.

If the ATAM system has no creation operations, the following procedure constructs an equivalent TAM system:

- Whenever a right x is entered in a cell of the ATAM system, it is also entered in the identical cell in the TAM system; but, moreover, the complementary right $\bar{x}$ is deleted from that cell in the TAM system.
- Similarly whenever a right x is deleted from a cell of the ATAM system, it is deleted from the identical cell in the TAM simulation. At the same time, the

complementary right $\bar{x}$ is entered in that cell in the TAM simulation.

- Also if an ATAM command tests for the absence of some rights, than the corresponding TAM command produced by our construction tests for the absence of rights by means of testing for presence of complementary rights. For example, the test $x \notin [S, O]$ in an ATAM command will be simulated by the test $\bar{x} \in [S, O]$ in the TAM system.

Theorem 1 For every ATAM system A the construction outlined above produces an equivalent TAM system B.

Proof: In order to prove that the construction outlined above produces an equivalent TAM system B, it is enough to show that TAM system B can simulate ATAM system A. In order to show this, we need to show that definition 27 is satisfied. As every ATAM command is mapped to a single TAM command, conditions 2 to 5 of definition 27 are trivially satisfied. Condition 6 of definition 27 is satisfied as the TAM system will maintain the invariant $x \in [S_i, O_j] \Leftrightarrow \bar{x} \notin [S_i, O_j]$ for all cells in the access matrix. □

4.1.2 Equivalence With Create and Destroy Operations

The occurrence of create operations in the given ATAM system considerably complicates the construction. We will focus only on creation of subjects, since every ATAM subject or object will be simulated in the TAM system as a subject (i.e., every column has a corresponding row in the access matrix). In other words the access matrix of the TAM system is square. This entails no loss of generality, since TAM subjects are not necessarily active entities.

A primitive "**create subject** S_j" operation introduces a new empty row and column in the access matrix. To follow through with the complementary rights con-

struction, we need to introduce the $\bar{R}$ rights in *every* cell involving S_j. The number of primitive operations required to do this is directly proportional to the number of subjects existing, at that moment, in the system. Since this is a variable number, a single TAM command cannot achieve this result. Instead we must use a sequence of TAM commands. The number of TAM commands required is unbounded, being proportional to the size of the access matrix.

Linked List Structure

To facilitate the TAM simulation, our construction organizes the subjects in a linked list structure, which can be traversed by following its pointers. The pointers, and the head and tail locations of the list, are easy to implement by rights in the access matrix. New subjects are inserted at the tail of the list when they are created. In order to fill up the row and column for the new subject with the complementary rights $\bar{R}$, the list is traversed from head to tail filling in $\bar{R}$ in the cells for the new subject along the way. Hence three new additional rights *head*, *tail*, and *next* are introduced in the initial state of the matrix. The right *head* in a cell $[S_i, S_i]$ of the matrix implies that S_i is the first subject in the linked list. Similarly, the right *tail* in a cell $[S_i, S_i]$ of the matrix implies that S_i is the last subject in the linked list. The right *next* in a cell $[S_i, S_j]$ of the matrix implies that S_j is the successor to S_i in the linked list (or equivalently that S_i is the predecessor to S_j in the list).

A create operation in an ATAM system is simulated by multiple commands in the TAM system. The key to doing this successfully is to prevent other TAM commands from interfering with the simulation of the given ATAM command. The simplest way to do this is to ensure that ATAM commands can be executed in the TAM simulation only one at a time. To do this we need to synchronize the execution of successive ATAM commands in the TAM simulation. Thus the problem of simulating

	SNC	S_1	S_2	...	S_n
SNC	*token*				
S_1		*head*	*next*		
S_2					
...					
S_n					*tail*

Figure 4.1: Initial Access Matrix of the TAM Simulation

ATAM in TAM requires solution of a synchronization problem. Synchronization is achieved by introducing an extra subject called SNC, and an extra right *token* as shown in figure 4.1. The role of SNC is to sequentialize the execution of simulation of ATAM commands in the TAM system.

The type of SNC is *snc*, and is assumed, without loss of generality, to be distinct from any type in the given ATAM system. It is also assumed, without loss of generality, that rights *next, head, tail, token,* C and tr are distinct from the rights in the given ATAM system. The rights C and tr are used for "book-keeping" purposes in the simulation, as will be explained below.

To summarize, the initial state of the TAM system consists of the initial state of the given ATAM system augmented in three respects.

- First, an empty row is introduced for every ATAM object, which does not have a row in the given ATAM access matrix. The *head*, *tail* and *next* rights are introduced to order the subjects in a linked list.

- Secondly, complementary rights are introduced as per the following equation:

$$\forall c \in R, x \in [S_i, S_j] \Leftrightarrow \bar{x} \notin [S_i, S_j]$$

- Thirdly, the SNC subject is introduced in the access matrix with $[SNC, SNC]$= $\{token\}$, and all other cells involving SNC being empty.

Simulation of ATAM create commands

We now consider how the ATAM command $CCreate$ given below can be simulated by several TAM commands.

command $CCreate\ (S_1 : s_1, S_2 : s_2, \ldots, S_m : s_m, S_c : s_c)$
if $\alpha(S_1, S_2, \ldots, S_m)$ **then**
    create subject S_c **of type** s_c;
end

The name $CCreate$ is a mnemonic for conditional creation. This command tests for the condition $\alpha(S_1, S_2, \ldots, S_m)$. If the condition is true, the command creates a new subject S_c.

The TAM simulation of $CCreate$ proceeds in three phases, respectively as illustrated in figure 4.2, figure 4.3 and figure 4.4. In these figures we show only the relevant portion of the access matrix, and only those rights introduced specifically for the TAM simulation. Complementary rights are shown only in the cells involving the newly created subject S_c. It is understood that the original ATAM rights are distributed exactly as in the ATAM system, along with complementary rights required to maintain the equation $x \in [S_i, S_j] \Leftrightarrow \bar{x} \notin [S_i, S_j]$. In the figures, n represents the total number of subjects in the system prior to the create operation.

The first phase consists of a single TAM command *CCreate-I* which tests whether (i) the condition of the ATAM command $\alpha(S_1, S_2, \ldots, S_m)$ is true, and (ii) whether $token \in [SNC, SNC]$. The former test is obviously required. The predicate α may involve tests for absence of access rights. Hence, in the TAM simulation we

	SNC	S_1	S_2	...	S_n
SNC	*token*				
S_1		*head*	*next*		
S_2					
...					
S_n					*tail*

(a) Prior to Phase I

	SNC	S_1	S_2	...	S_n	S_c
SNC						
S_1		*head*	*next*			
S_2						
...						
S_n					*tail*	
S_c						*tr*

(b) After Phase I

Figure 4.2: TAM Simulation of the ATAM Command $CCreate$: Phase I

replace α by α', which is obtained by substituting tests for presence of complimentary rights in place of tests for absence of rights in α. The latter test for the *token* right in $[SNC, SNC]$ ensures that the TAM simulation of $CCreate$ can commence only if no other ATAM command is currently being simulated. It also ensures that once phase I of the simulation of $CCreate$ has started, the simulation will proceed to completion before simulation of another ATAM command can begin. The phase I TAM command is given below.

command *CCreate-I*$(S_1 : s_1, S_2 : s_2, ..S_m : s_m, S_c : s_c, SNC : snc)$

if $\alpha'(S_1, S_2, \ldots, S_m) \wedge token \in [SNC, SNC]$ **then**

delete *token* **from** $[SNC, SNC]$;

create S_c;

enter tr **in** $[S_c, S_c]$;

end

The body of this command deletes *token* from $[SNC, SNC]$ and creates subject S_c. It also enters tr in $[S_c, S_c]$ indicating that all cells corresponding to S_c have to be traversed. The states of the access matrix, before and after execution of *CCreate-I*, are outlined in figure 4.2(a) and figure 4.2(b) respectively.

In phase II of the simulation the right C is passed, in turn, from $[S_c, S_1]$ to $[S_c, S_2]$ and so on to $[S_c, S_n]$. A right C in a cell of a matrix indicates that all complementary rights have to be introduced in that cell. Hence complementary rights $\bar{R}$ are introduced in the cell from which the right C is removed. The phase II commands are given below. The type of subjects S_i and S_r indicated in the commands by $T \Leftrightarrow snc$ implies that these subjects can be of any type in $T \Leftrightarrow snc$ (i.e., any type other than *snc*). Strictly speaking, we should have a separate command for each type in $T \Leftrightarrow snc$, but we allow this slight abuse of notation to simplify the presentation.

command *CCreate-1-II*$(S_c : s_c, S_i : T \Leftrightarrow snc)$

if $tr \in [S_c, S_c] \wedge head \in [S_i, S_i]$ **then**

enter C **in** $[S_c, S_i]$;

end

command *CCreate-2-II* $(S_c : s_c, S_i : T \Leftrightarrow snc, S_r : T \Leftrightarrow snc)$

if $tr \in [S_c, S_c] \wedge next \in [S_i, S_r] \wedge C$ in $[S_c, S_i]$ **then**

enter $\bar{R}$ **in** $[S_c, S_i]$;

enter $\bar{R}$ **in** $[S_i, S_c]$;

delete C **from** $[S_c, S_i]$;

enter C **in** $[S_c, S_r]$;

end

	SNC	S_1	S_2	...	S_n	S_c
SNC						
S_1		$head$	$next$			
S_2						
...						
S_n					$tail$	
S_c		C				tr

(a) After execution of *CCreate-1-II*

	SNC	S_1	S_2	...	S_n	S_c
SNC						
S_1		$head$	$next$			$\bar{R}$
S_2						
...						
S_n					$tail$	
S_c		$\bar{R}$	C			tr

(b) After one execution of *CCreate-2-II*

	SNC	S_1	S_2	...	S_n	S_c
SNC						
S_1		$head$	$next$			$\bar{R}$
S_2						$\bar{R}$
...						$\bar{R}$
S_n					$tail$	
S_c		$\bar{R}$	$\bar{R}$	$\bar{R}$	C	tr

(c) End of phase II

Figure 4.3: TAM Simulation of the ATAM Command $CCreate$: Phase II

	SNC	S_1	S_2	...	S_n	S_c
SNC	$token$					
S_1		$head$	$next$			$\bar{R}$
S_2						$\bar{R}$
...						$\bar{R}$
S_n						$next$,$\bar{R}$
S_c		$\bar{R}$	$\bar{R}$	$\bar{R}$	$\bar{R}$	$tail$,$\bar{R}$

Figure 4.4: TAM Simulation of the ATAM Command $CCreate$: Phase III

The command *CCreate-1-II* tests if phase I is completed by looking for right tr, and than enters right C in the head column of the list of subjects. Command *CCreate-2-II* introduces complementary rights in all cells involving S_c except the tail subject by passing right C along the linked list of subjects. The insertion of complimentary rights in the tail column of the linked list is deferred until Phase III. The execution of Phase II commands is illustrated in figure 4.3.

In phase III of the simulation, the new subject S_c is inserted at the end of the linked list. At the same time, complementary rights are introduced in the previous and the new tail cells. Also $token$ is introduced in the cell $[SNC, SNC]$ indicating that the simulation of $CCreate$ is complete. The Phase III command is given below.

Command *CCreate-III*($S_c : s_c, S_n : T \Leftrightarrow snc, SNC : snc$)
if $tr \in [S_c, S_c] \wedge C \in [S_c, S_n] \wedge tail \in [S_n, S_n]$ **then**
    delete C from $[S_c, S_n]$;
    enter $\bar{R}$ **in** $[S_c, S_n]$;
    enter $\bar{R}$ **in** $[S_n, S_c]$;
    enter $\bar{R}$ **in** $[S_c, S_c]$;
    enter $next$ **in** $[S_n, S_c]$;

delete *tail* **from** $[S_n, S_n]$;

enter *tail* **in** $[S_c, S_c]$;

delete *tr* **from** $[S_c, S_c]$;

enter *token* **in** $[SNC, SNC]$;

end

Prior to execution of the *CCreate-III* command we have the situation shown in figure 4.3(a). After execution of *CCreate-III* we have the situation of figure 4.4. The TAM simulation is now ready to proceed with execution of another ATAM command.

Simulation of ATAM destroy commands

In order to simulate creation,we have seen that the subjects need to be related in a linked list structure. Hence, whenever a subject is destroyed the linked lists should still be maintained. For instance, when subject S_3 is destroyed in context of figure 4.5 (a), we should maintain the linked list as shown in figure 4.5(b).

To be concrete, consider the following ATAM command whose name *CDestroy* is a mnemonic for conditional destroy.

command *CDestroy* $(S_1 : s_1, S_2 : s_2, ..S_m : s_m, S_d : s_d)$

if $\alpha(S_d, S_1, S_2, \ldots, S_m)$ **then**

destroy subject S_d;

end

This command can be simulated by a single TAM command, since maintenance of the linked list requires adjustment to a fixed number of cells of the access matrix. However, we do need several variations of the TAM command, depending upon whether the subject being destroyed is in the middle, or at the head or tail of the linked

	SNC	S_1	S_2	S_3	S_4	S_5
SNC	*token*					
S_1		*head*	*next*			
S_2				*next*		
S_3					*next*	
S_4						*next*
S_5						*tail*

(a) Before destruction of S_3

	SNC	S_1	S_2	S_4	S_5
SNC	*token*				
S_1		*head*	*next*		
S_2				*next*	
S_4					*next*
S_5					*tail*

(b) After destruction of S_3

Figure 4.5: Destruction of S_3

list. We also need a variation to simulate the extreme case where the subject being destroyed is the only subject in the linked list.

The TAM command to simulate *CDestroy*, when S_d is in the middle of the list, is as follows.

command *CDestroy-middle* $(S_d : s_d, S_l : T \Leftrightarrow snc, S_r : T \Leftrightarrow snc,$
$S_1 : s_1, S_2 : s_2, \ldots, S_m : s_m, SNC : snc)$
if $\alpha'(S_d, S_1, S_2, \ldots, S_m) \wedge token \in [SNC, SNC] \wedge next \in [S_d, S_r]$
$\wedge\ next \in [S_l, S_d]$ **then**
destroy subject S_d;
enter $next$ **in** $[S_l, S_r]$;
end

The predicate α' is obtained by substituting tests for presence of complimentary rights in place of tests for absence of rights in α. The test for the *token* right ensures that ATAM commands are simulated one at a time. The tests for *next* ensure that S_l and S_r are respectively the predecessor and successor of S_d in the linked list. The body of the command maintains the linked list.

If S_d is at the head or at the tail of the linked list, we respectively have the following two commands.

command *CDestroy-head* ($S_d : s_d$,$S_r : T \Leftrightarrow snc$,
$S_1 : s_1, S_2 : s_2, \ldots, S_m : s_m,\ SNC : snc$)
if $\alpha'(S_d, S_1, S_2, \ldots, S_m) \wedge token \in [SNC, SNC] \wedge next \in [S_d, S_r]$
$\wedge\ head \in [S_d, S_d]$ **then**
    destroy subject S_d;
    enter *head* **in** $[S_r, S_r]$;
end

command *CDestroy-tail* ($S_d : s_d, S_l : T \Leftrightarrow snc$,
$S_1 : s_1, S_2 : s_2, \ldots, S_m : s_m, SNC : snc$)
if $\alpha'(S_d, S_1, S_2, \ldots, S_m) \wedge token \in [SNC, SNC] \wedge tail \in [S_d, S_d]$
$\wedge\ next \in [S_l, S_d]$ **then**
    destroy subject S_d;
    enter *tail* **in** $[S_l, S_l]$;
end

Finally, for the extreme case where S_d is the only subject in the linked list we have the following TAM command.

command *CDestroy-last* ($S_d : s_d,\ SNC : snc$)
if $\alpha'(S_d) \wedge token \in [SNC, SNC] \wedge head \in [S_d, S_d] \wedge tail \in [S_d, S_d]$ **then**

destroy subject S_d;

end

4.1.3 Simulation of ATAM schemes

So far we have seen how ATAM commands which do not have create and destroy operations, and ATAM commands which just have either create or destroy operations are converted into TAM commands. A general procedure for simulating an arbitrary ATAM command in TAM can be obtained by combining these ideas. Consider a ATAM command with multiple operations (i.e., a sequence of **enter**, **delete**, **create**, and **destroy** operations). Based on the previous discussion, we know how to simulate each primitive operation in turn. With some additional "book-keeping" we can keep track of a "program counter" which moves down the sequence of primitive operations in the body of the ATAM command, as each one gets simulated.

The procedure given in this section is not *proper* as some of the parameters in phases II and phase III commands of the simulating system might not belong to a type given by (T_{TAM}-T_{ATAM}). The procedure given in this section can be easily made proper. The same procedure can be used with the following additional modifications.

- For every subject S_i in the initial state an extra subject $S_i^{'}$ is created and a right *cohort* is entered in cell $(S_i, S_i^{'})$.
- The modified procedure organizes the cohort subjects in the list structure and not the other subjects. Indirectly the other subjects are arranged in a list through these cohorts.
- Complementary rights are introduced in the cohort cells. For example, testing for absence of right a in cell (S_i, S_j) is done in ATAM by testing for presence of complementary right $\bar{a}$ in $(S_i^{'}, S_j^{'})$.

- When a right is entered in a cell (S_i, S_j) in ATAM, then in TAM its complementary right is entered in $(S_i^{'}, S_j^{'})$ along with the right in (S_i, S_j). Similarly when a right is deleted from a cell (S_i, S_j), then in TAM that right is deleted from the same cell and at the same time its complementary right is introduced in $(S_i^{'}, S_j^{'})$.

- When a subject is created in ATAM, the commands in TAM to simulate the creation are almost similar to the commands given earlier. Phase I command creates a new subject along with its cohort. Phase II and Phase III commands are same except the parameters are all cohorts and the complementary rights and all other rights are modified in the cohorts.

- When a subject is destroyed in ATAM, the commands in TAM are similar. They along with destroying the subject and its cohort make sure that the list is not broken.

We conclude this section by stating the relationship between the expressive power of TAM and ATAM.

Theorem 2 TAM and ATAM are weakly equivalent in expressive power, i.e., for every ATAM system A the construction outlined above produces an equivalent TAM system B.

Proof: In order to prove that the construction outlined above produces an equivalent TAM system B, it is enough to show that TAM system B can simulate ATAM system A. In order to show this, we need to show that definition 27 is satisfied. Every command of an ATAM system is mapped to a single partial order of commands (the partial order corresponds to the successful execution) and also satisfying condition 1 of definition 27 (as the construction ensures that mapping is disjoint and proper).

The construction ensures that conditions 2 to 5 are satisfied by simulating a single ATAM command at a time and by ensuring that once a simulation of a command has started, the simulation will proceed to completion before simulation of another ATAM command can begin. Condition 6 of definition 27 is satisfied as the TAM system maintains the invariant $x \in [S_i, O_j] \Leftrightarrow \bar{x} \notin [S_i, O_j]$ for all cells in the access matrix and as the TAM system behaves exactly like the ATAM system in terms of rights of ATAM system. □

All our constructions in this thesis ensure that conditions 2 to 5 are easily satisfied by simulating a single command at a time and by ensuring that once a simulation of a command has started, the simulation will proceed to completion before simulation of another command can begin.

4.2 Strong Non-Equivalence of ATAM and TAM

It has been shown in the previous section that TAM and ATAM are strongly equivalent in terms of expressive power when there is no creation. In this section, we look at the relationship between the expressive power of ATAM and TAM when creation is allowed. We formally prove that ATAM is not strongly equivalent to TAM. We formally prove the non-equivalence by proving that TAM simulation of ATAM cannot be bounded. The construction given in the previous subsection is not bounded.

The fact that TAM and ATAM are equivalent (when creation is not allowed) was demonstrated earlier by introducing complementary rights in the initial state. Whenever a subject is created, the earlier construction requires that complementary rights be introduced in every cell of that subject. We prove in this section that this is not possible in a bounded simulation.

For the System A given below, we prove that there is no strongly equivalent TAM System B (which is a bounded simulation of A).

Definition of System A

(1) Rights: R={a,b}

(2) Types: T={p,c,o}

(3) The following are the commands:

command *create-object* $(S : p, O : o)$

  create O;

end

command *create-subject* $(S : p, S_c : c)$

  create S_c;

end

command *create-subject1* $(S : p, S_c : c, O : o)$

  create S_c;

  enter a in $[S_c, O]$;

end

command *atam* $(S_c : c, O : o)$

if $a \notin [S_c, O]$**then**

  enter a in $[S_c, O]$;

  enter b in $[S_c, O]$;

end

(4) Initial state of the ATAM system A has object O_1 of type o and a subject S of type p. The subject has right a for object O_1. The initial state is shown in figure 4.6. We now illustrate the effect of the commands given in system A.

	$S:p$	$O_1:o$
$S:p$		a

Figure 4.6: Initial state of ATAM system A

The effect of command *create-object* is the creation of a new object and hence a new column in the matrix. For example after the creation of an object O_c through command *create-object*, the initial state of system A evolves to figure 4.7 (a).

The effect of command *create-subject* is the creation of a new subject and hence a new row and column in the matrix. For example after the creation of a subject S_c through command *create-subject*, the matrix of system A evolves from figure 4.7 (a) to figure 4.7 (b).

The effect of command *create-subject1* is the creation of a new subject and hence a new row and column in the matrix. The newly created subject also gets a right a for its parent object. For example after the creation of a subject S_c through command *create-subject1*, system A evolves from figure 4.7 (b) to figure 4.7 (c). The newly created subject S_c gets a right a for the object O_c involved in its creation.

The command *atam* enters rights a and b in a cell which doesn't has a right a. For example when the command *atam* executes with actual parameters S_c and O_1, the matrix evolves from figure 4.7 (c) to figure 4.7 (d). We just illustrated the effect of the commands in system A.

We now show that ATAM system A doesn't have any equivalent TAM system B. Say there exists an equivalent TAM system B. Then according to definition 13, every command of A is mapped to exactly two partial orders of commands of B and for every successful execution of a command in A, there exists a a corresponding successful partial order in B. According to lemma 1, the partial order of commands

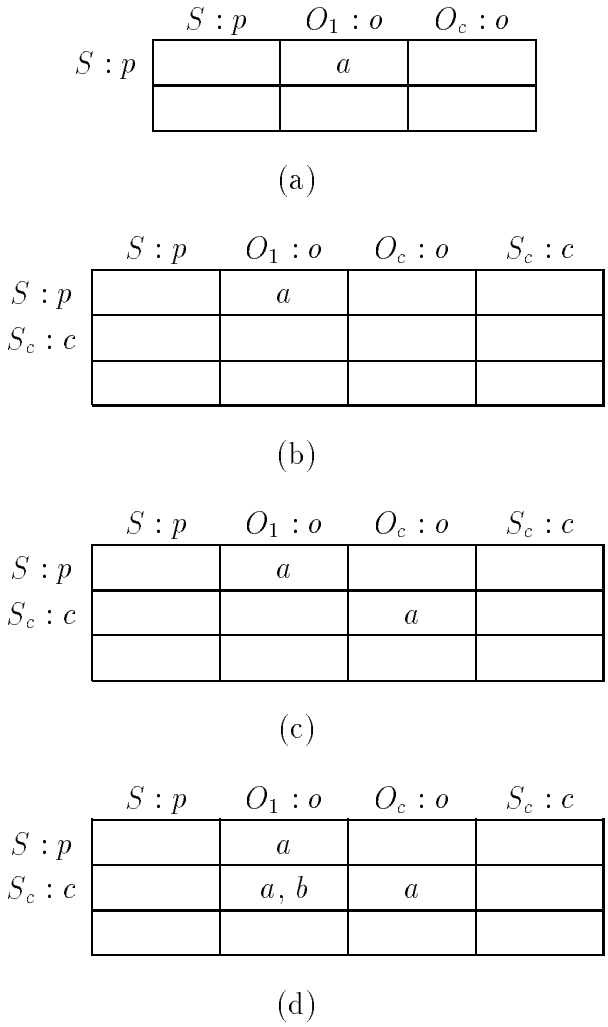

Figure 4.7: Example of ATAM System A

mapped from the successful execution of *atam* should test for the predicate of *atam* (i.e., test for absence of a in a cell).

Lemma 1 *B can simulate A only if the partial order mapped from the successful execution of atam tests for the predicate of atam (the absence of right a in a cell).*

Proof: We prove by contradiction. We assume that B doesn't have any commands which test for absence of a. The partial order mapped from the successful execution of *atam* should enter rights a and b in a cell which does not have a. Since none of the commands test for absence of a, they can enter right b in a cell which already has a. This cannot be achieved in A as from lemma 2 a cell which has a right a cannot get a right b. Hence the theorem is true. □

Lemma 2 *Any cell in A which has only right* a *(and doesn't have right* b*) can never get right* b.

Proof: A cell in A say $[S, O_1]$ of figure 4.7 (a) has only right a and doesn't have any rights. We now prove why $[S, O_1]$ cannot get a right b. As system A doesn't have any commands which delete a right a from a cell, the right a can never be removed from $[S, O_1]$. System A has only one command which enters a right b in a cell and this command is *atam*. But as *atam* tests for absence of right a in a cell before entering a right b, the cell $[S, O_1]$ can never get right b. Hence the lemma is true. □

According to lemma 1, B can simulate A only if the partial order mapped from the successful execution of *atam* tests for the predicate of *atam* (which is the absence of a in a cell). The testing for predicate of *atam* (the absence of a in a cell) can be done in TAM either through commands which test for presence of rights or by commands which indicate the absence of a by testing for the existence of a subject

(i.e., the encoding of the fact that a is not in a cell should be done either through testing for presence of rights or through the existence of subjects). The creation of a new subject S_c in B involves not only creating an empty row and column for S_c, but also encoding the fact that there are no rights in that row and column. This encoding is done through some rights or through the existence of subjects. Theorems 3 and 4 prove that, the fact a does not belong to all the cells of S_c cannot be encoded by either rights or through existence of subjects. These theorems use the fact that whenever a right a is entered in a cell (which does not have a), either some right is deleted when a is entered or some subject should be destroyed once a is entered.

Theorem 3 *When a subject S_c is created, the encoding of the fact that right a is not in all the cells of S_c by testing for presence of rights cannot be done in B.*

Proof: Say the encoding can be done through testing for presence of rights. i.e., say B has commands which tests for L cells to indicate the absence of a in a cell. When a is entered in that cell, then the test in at least one of those L cells is made false. Say the test is made false in M cells. Then these M cells can be used to indicate the absence of a in some constant number of cells and this number is a function of M and N [1].

The initial state has constant cells and they can only represent absence of rights in a constant number of cells. When a new subject is created, rights can be only entered in some constant number of cells (as the simulation is bounded and so only constant number of commands can be used to enter rights in the new cells) . At

[1] Actually M cells can be used to indicate the absence of a for at most $(\frac{N!}{2!(N-2)!})^M$ cells, where N is the number of rights in B. i.e. M cells can at most represent absence of right a in $(\frac{N!}{2!(N-2)!})^M$ cells, which is constant as M and N are constants. This equation is derived from the fact every cell has at most $(\frac{N!}{2!(N-2)!})$ combinations of sets of rights, where one is not a proper subset of another. Since rights are removed from M cells, the M cells can represent at most $(\frac{N!}{2!(N-2)!})^M$ cells.

some point, when a new subject is created the system can reach a state where all the newly created cells would not be encoded of the fact that a is not present in them! □

Theorem 4 *When a subject S_c is created, the encoding of the fact that a is not in a cell cannot be done through testing for existence of subjects.*

Proof: Assume that the theorem is false. i.e., the encoding can be done through testing for existence of subjects. Then B has commands which indicate the absence of a in all cells of the created subject by testing for existence of subject or subjects for each cell. Then there should be a command which has a subject of some type say s_x as its parameter which indicates the absence of a in some cell (S_c, O_i). When a is entered in (S_c, O_i), then that subject or all the subjects of type s_x should be destroyed. This indicates that for each object we need a different type to simulate the command *atam*. As the number of types should be specified in the scheme, there can be only constant of them. Let the number of these types be β. But B can have objects greater in number than β and hence there would be a cell where *atam* cannot be simulated. Hence the theorem is false. □

Lemma 1 states that B can simulate A only if the partial order mapped from the successful execution of *atam* tests for the predicate of *atam* (which is the absence of a in a cell). The predicate of *atam* can be tested either through testing for presence of rights or through testing for existence of subjects. Theorems 3 and 4 indicate that whenever a new subject is created, the encoding of the fact that a is not in all the cells of the created subject cannot be done either through testing for presence of rights or through testing for existence of subjects. Hence we conclude that ATAM $\not\equiv$ TAM.

Chapter 5

Dynamic Separation of Duties Based on ATAM

The principal contribution of this chapter is that it demonstrates that specifying *dynamic* separation of duties requires the ability to test for the absence of rights in access control models. It also demonstrates that the ability to test for absence of rights is needed to ensure that a given subject does not perform two conflicting operations on the same object.

In this chapter we consider the implementation of separation of duties. Separation of duties is an important real-world requirement that useful access control models need to support. The particular separation of duties mechanism we implement is the transaction control expression (TCE), introduced in [San88c]. It was shown in [San88c, San91] that TCEs could easily specify typical transactions in which both separation and coincidence of duties were clear requirements. In particular, TCEs go beyond the static separation of duties stipulated by Clark and Wilson [CW87]. Static specification of separation of duties is too restrictive. For example, in modeling the common real world scenario in which a subject takes one role with respect to object A but another role with respect to object B, completely static specification of separation of duties is inadequate.

The principal contribution of this chapter is that it demonstrates that specifying *dynamic* separation of duties requires the ability to test for the absence of rights in access control models. Specifically, the ability to test for the absence of a right is

needed to ensure that a given subject does not perform two conflicting operations on the same object. This chapter builds upon and extends the constructions outlined in [AS92b].

This chapter's organization is as follows. Section 5.1 translates the TCE examples given in [San88c] into an augmented TAM implementation. Section 5.2 summarizes the important points for implementing general TCE expressions in augmented TAM by automated translation. Finally section 5.3 summarizes this chapter.

5.1 Implementing Transaction Control Expressions

In this section we show how the examples of transaction control expressions given in [San88c] can be expressed in augmented TAM. A transaction control expression represents the potential history of an information object. Sandhu [San88c] distinguished two kinds of information objects: *transient objects* which can have a bounded number of operations applied to them, versus *persistent objects* which can potentially have an unbounded number of operations. Transient objects are intuitively modeled on forms, which are filled out and after appropriate approvals lead to some action such as issuing a check. Persistent objects intuitively model books in which account balances are maintained.

5.1.1 Transient Objects

The classic example of a transient object is a voucher that ultimately results in a check being issued. The potential history of a voucher is represented by the following transaction control expression [San88c].

prepare • clerk;
approve • supervisor;
issue • clerk;

Each *term* in this expression has two parts. The first part names a transaction. The transaction can be executed only by a user with *role* specified in the second part. For simplicity in discussion assume each user has only one role. So 'prepare • clerk' specifies that the prepare transaction can be executed on a voucher only by a clerk. The semi-colon signifies sequential application of the terms. That is a supervisor can execute the approve transaction on a voucher only after a clerk has executed the preceding prepare transaction. Finally, separation of duties is specified by requiring that the users who execute different transactions in the transaction control expression all be distinct.

We now show how the given TCE is specified in augmented TAM. We make use of the following sets of types and rights:

1. Rights R={prepare, prepare$'$, approve, approve$'$, issue, issue$'$}
2. Types T={voucher, clerk, supervisor, manager}, all of which are subject types, with principal types T_P={clerk, supervisor, manager}

Rights are used as a means of keeping track of the current location in the progression of a transaction control expression. *Undecorated rights*, i.e., those rights without a trailing apostrophe, are used to indicate that current operation in the transaction control expression is in progress. *Decorated rights*, i.e., those rights with a trailing apostrophe, are used to indicate that current operation in the transaction control expression is complete. The decorated rights are useful in ensuring both separation and coincidence of duties.

The augmented TAM commands for the voucher transaction control expression are given below. Each step of the TCE is translated into two commands: the first indicating that the step in question is in progress, and the second indicating that the step has been completed.

(a) **command** begin-prepare-voucher

$(C : clerk, V : voucher)$

create subject V;

enter prepare **into** $[C, V]$;

end

(a′) **command** complete-prepare-voucher

$(C : clerk, V : voucher)$

if prepare $\in [C, V]$ **then**

delete prepare **from** $[C, V]$;

enter prepare′ **into** $[C, V]$;

enter prepare′ **into** $[V, V]$;

end

(b) **command** begin-approve-voucher

$(S : supervisor, V : voucher)$

if prepare′ $\in [V, V]$ **then**

delete prepare′ **from** $[V, V]$;

enter approve **into** $[S, V]$;

end

(b′) **command** complete-approve-voucher

$(S : supervisor, V : voucher)$

if approve $\in [S, V]$ **then**

delete approve **from** $[S, V]$;

enter approve′ **into** $[S, V]$;

enter approve′ **into** $[V, V]$;

end

(c) **command** begin-issue-check

$(C : clerk, V : voucher)$

if approve$'$ $\in [V, V] \wedge$ prepare$'$ $\notin [C, V]$ **then**

delete approve$'$ **from** $[V, V]$;

enter issue **into** $[C, V]$;

end

(c$'$) **command** complete-issue-check

$(C : clerk, V : voucher)$

if issue $\in [C, V]$ **then**

delete issue **from** $[C, V]$;

enter issue$'$ **into** $[C, V]$;

enter issue$'$ **into** $[V, V]$;

end

To control progress of the TCE, the clerk in command (a) creates a voucher subject and acquires the undecorated right prepare, indicating that the first operation of the TCE is in progress. (As will be discussed later, command (a) can be modified to tie the voucher subject to one or more particular accounts with respect to which the voucher is being prepared.) Once the voucher has been prepared command (a$'$) is invoked to indicate, via the prepare$'$ right, that voucher preparation is complete. Command (a$'$) can be invoked only by the same clerk who invoked command (a) for a given voucher. Command (a$'$) enters the prepare$'$ right in the $[C, V]$ cell to record which clerk prepared the voucher. It also enters prepare$'$ in the $[V, V]$ cell to signify that the next step of the TCE can proceed. The commands (b) and (b$'$) allow a supervisor to obtain the approve right for the voucher provided preparation is complete; and subsequently denote, via the approve$'$ right, that voucher approval

is granted. The command (c) give the named clerk the issue right for the voucher provided the voucher has been approved, and the specific clerk named in the command does *not* hold the prepare′ right for the voucher. This is where the facility to test for absence of rights is crucial. Command (c′) subsequently indicates, via the issue′ right, that the check has been issued. At this point the voucher's TCE is complete and the voucher can be archived. (The ATAM command for archival has been omitted for simplicity.)

Several points about the example warrant attention. In particular, command (c) enforces dynamic separation of duties by checking for the absence of the prepare′ right before allowing a specific clerk to obtain the issue right. Testing for the absence of a right in a cell in the access matrix is outside the expressive power of non-monotonic access matrix formulations such HRU and TAM[1].

Also, commands "clean up" after themselves so as to ensure that only one thread is followed. For example, once a clerk has obtained the issue right, via command (c), no other clerk can obtain the issue right (because the approve′ right has been deleted from $[V, V]$). Thus it is assured that two clerks will not concurrently issue the check, with the undesirable consequence that two checks get issued for the same voucher.

Let us now see what happens when the same TCE is specified without using testing for absence of rights. If the model does not have the ability to test for absence of rights, than there should be a command which ensures that the clerk who issues the check did not prepare it. Since the commands cannot test for absence of rights, they should test for presence of some rights. This is similar to testing for complimentary rights in the construction given in section 4.1. If the construction of 4.1 is used,

[1]It has been proved in chapter 4.2 that ATAM and TAM are not strongly equivalent in terms of expressive power.

than whenever a voucher is created, *complementary rights* have to be introduced for all the subjects existing in the system (to that created voucher). A *complementary right* $\bar{x}$ in a cell of the matrix implies that the right x is not present in that cell. In large scale systems, introducing *complementary rights* to all the subjects existing in the system whenever a creation occurs is practically infeasible. Hence to implement TCE's, the ability to testing for absence of rights is needed in access control models.

Now suppose the check requires approval by three supervisors. We can specify this with the following TCE.

prepare • clerk;
approve • supervisor;
approve • supervisor;
approve • supervisor;
issue • clerk;

With this expression the three approve transactions must be executed sequentially. This is appropriate in a manual system where there is one physical representation of the check, which can be accessed by only one supervisor at a time. However, in a computerized system, it should be possible to request concurrent approval. Sandhu [San88c] proposed the following notation for expressing multiple approval.

prepare • clerk;
3 : approve • supervisor;
issue • clerk;

The colon is a voting constraint specifying 3 votes from 3 different supervisors in this case, without requiring the voting to be sequential.

We can implement this example in augmented TAM by modifying the commands (b) and (b′) from the previous example into the commands shown below. The other commands remain as they are.

(1b) **command** begin-approve-voucher-supervisor-1

$(S : supervisor, V : voucher)$

if prepare$' \in [V, V]$ **then**

delete prepare$'$ **from** $[V, V]$;

enter approve **into** $[S, V]$;

enter approve^{n-1} **into** $[V, V]$;

enter approve$_0^{'}$ **into** $[V, V]$;

end

(1bk) **command** begin-approve-voucher-supervisor-k (n-1 $\geq$ k $\geq$ 1)

$(S : supervisor, V : voucher)$

if approve $\notin [S, V] \wedge$ approve$^k \in [V, V]$ **then**

enter approve **into** $[S, V]$;

delete approve^k **from** $[V, V]$;

enter approve^{k-1} **into** $[V, V]$;

end

(1b$'$) **command** complete-approve-voucher-supervisor-k (n-1 $\geq$ k $\geq$ 0)

$(S : supervisor, V : voucher)$

if approve $\in [S, V] \wedge$ approve$_k^{'} \in [V, V]$ **then**

delete approve **from** $[S, V]$;

enter approve$'$ **into** $[S, V]$;

delete approve$_k^{'}$ **into** $[V, V]$;

enter approve$_{k+1}^{'}$ **into** $[V, V]$;

end

For our example the value of n in the above commands is three. By substituting the value of n suitably, the construction works for any arbitrary number of supervisors needed to approve. For our example six new additional rights approve^2, approve^1, approve$_0^{'}$, approve$_1^{'}$ approve$_2^{'}$ and approve$_3^{'}$ are added to the set R. A right approve^i

in [V,V] implies that the voucher has still to be approved by i supervisors. A right approve$_i'$ in [V,V] implies that the voucher has already been approved by i supervisors. Command (1b) checks via the prepare$'$ right, whether voucher preparation is complete and if so, it enters right approve in [S,V]. It also enters right approve^2 in $[V, V]$ to indicate that voucher also needs approval from two other supervisors (in addition to the one in its command argument). Command (1bk) is not a single command. It represents a different command for each value of k and the range of values that k can take is indicated in each command. In our example k takes values of 1 and 2 (as the value that k takes is given by n-1 $\geq$ k $\geq$ 1 and as n is three for our case). This representation is used here for brevity rather than showing commands for each value of k. Every command represented by (1bk) tests if the supervisor in its argument doesn't have approve right and also if the voucher has to be approved by any other supervisor and if so, it enters approve right to the supervisor in its argument indicating that he is ready to approve the command. Also in doing so it makes sure that the number of supervisors still needed to approve is one less than before. Every command represented by (1b$'$) tests if any one of the supervisors is ready to approve by testing for the right approve in [S,V] and if so, it enters right approve$'$ in [S,V] indicating the approval. It also makes sure that the the number of supervisors that have approved the voucher is one more than before.

In the next step the clerk tests for the completion phase by testing for right approve$_n'$ (approve$_3'$ in our case) in [V,V].

The preceding implementation is asynchronous in the sense that each command has only a single supervisor and a voucher as arguments. It can be argued that asynchronous agreement better models organizational requirements.

Following [San88c], further consider the requirement that either three supervisors approve the check or the department manager plus one supervisor approve it.

The TCE notation allows weights for different roles as follows.

prepare • clerk;
3 : approve • manager=2, supervisor=1;
issue • clerk;

Approve transactions with sufficient votes are required before proceeding to the next term. In this case approve transactions executed by managers have weight 2 whereas those executed by supervisors have weight 1. If two managers approve the check we get 4 votes. It seems reasonable to allow this so we interpret the number of votes required as a lower bound. The moment 3 or more votes are obtained the next step is enabled.

Essentially the implementation must allow for progress to be made by the disjunction of various possible steps. A natural way to implement this is with a corresponding variety of augmented TAM commands for a given step, each of which is capable of enabling the following step.

For this example, a possible TAM implementation is as follows. The previous commands (1b), (1bk) and (1b′) are still acceptable and necessary, in that they represent a possible way in which approval might be achieved. They are not sufficient, however, since the implementation must account for other ways in which votes may be collected. There needs to be a means by which a manager can combine with one or two supervisors, and also a means by which two managers can generate an approval. This results in the following commands. These commands are similar to commands (1b), (2bk) and (2b′) except that whenever a manager is given an approve right, the right indicating the number of supervisors(or managers) still needed for approval is replaced in a way that it indicates the new number of supervisors needed to approve is two fewer than before. Also when ever a manager approves a voucher, than the right indicating the total number of approvals is replaced in a way that it indicates the number of supervisors that approved the voucher is two more than before.

(2b) **command** begin-approve-voucher-manager-1

$(M : manager, V : voucher)$

if prepare$'$ $\in [V, V]$ **then**

delete prepare$'$ **from** $[V, V]$;

enter approve **into** $[M, V]$;

enter approve^1 **into** $[V, V]$;

enter approve$'_0$ **into** $[V, V]$;

end

(2bk) **command** begin-approve-voucher-manager-k (n-2 $\geq$ k $\geq$ 1)

$(M : manager, V : voucher)$

if approve $\notin [M, V] \wedge$ approve$^k \in [V, V]$ **then**

enter approve **into** $[M, V]$;

delete approve^k **from** $[V, V]$;

enter approve^{k-2} **into** $[V, V]$;

end

(2b$'$) **command** complete-approve-voucher-manager-k (n-1 $\geq$ k $\geq$ 0)

$(M : manager, V : voucher)$

if approve $\in [M, V] \wedge$ approve$'_k \in [V, V]$ **then**

delete approve **from** $[M, V]$;

enter approve$'$ **into** $[M, V]$;

delete approve$'_k$ **into** $[V, V]$;

enter approve$'_{k+2}$ **into** $[V, V]$;

end

The concise voting notation in the TCE has been fully enumerated in the translation to augmented TAM commands. Provided the translation is automated,

this expansion is not problematical for many typical cases.

5.1.2 Coincidence of Duties

Sometimes different transactions in an object history must be executed by the same user. Consider a purchase order with the following transaction control expression.

requisition • project-leader;
prepare • clerk;
approve • manager;
agree • project-leader;
issue • clerk;

The idea is that a project leader initiates a requisition, a purchase order is prepared from the requisition, approved by a purchasing manager, and then needs agreement of the project leader before finally being issued by a clerk. Our rule of distinct identity implies different project leaders be involved in requisitioning and agreeing, contrary to the desired policy. The following TCE syntax identifies which steps must be executed by the same user.

requisition • project-leader ↓ x;
prepare • clerk;
approve • manager;
agree • project-leader ↓ x;
issue • clerk;

The anchor symbol '↓' identifies steps which must be executed by the same individual. The x following it is merely a token for relating multiple anchors, as for example in the TCE given below.

requisition • project-leader ↓ x;
prepare • clerk;
approve • manager ↓ y;
agree • project-leader ↓ x;
reapprove • manager ↓ y;
issue • clerk;

In this case there are two steps to be executed by the same project leader, and two to be executed by the same purchasing manager.

The prepare, approve and issue steps in this TCE are similar to those in the voucher example of section 5.1.1, and can be accomplished by similar augmented TAM commands. In addition we need commands to initiate the requisition, agree to the purchase order, and to reapprove the purchase order. To implement this TCE we define the following rights and types.

1. Rights R={requisition, requisition$'$, prepare, prepare$'$, approve, approve$'$, agree, agree$'$, reapprove$'$, reapprove$'$, issue, issue$'$}

2. Types T={purchase-order, project-leader, clerk, manager}, all of which are subject types, with principal types T_P={project-leader, clerk, manager}

The full set of augmented TAM commands is given below.

(a) **command** begin-initiate-requisition
(P : *project-leader*, O : *purchase-order*)
create subject O;
enter requisition **into** $[P, O]$;
end

(a$'$) **command** complete-initiate-requisition
(P : *project-leader*, O : *purchase-order*)
if requisition $\in [P, O]$ **then**
delete requisition **from** $[P, O]$;
enter requisition$'$ **into** $[P, O]$;
enter requisition$'$ **into** $[O, O]$;
end

(b) **command** begin-prepare-po

$(C : clerk, O : purchase\text{-}order)$

if requisition$' \in [O, O]$ **then**

delete requisition$'$ **from** $[O, O]$;

enter prepare **into** $[C, O]$;

end

(b$'$) **command** complete-prepare-po

$(C : clerk, O : purchase\text{-}order)$

if prepare $\in [C, O]$ **then**

delete prepare **from** $[C, O]$;

enter prepare$'$ **into** $[C, O]$;

enter prepare$'$ **into** $[O, O]$;

end

(c) **command** begin-approve-po

$(M : manager, O : purchase\text{-}order)$

if prepare$' \in [O, O]$ **then**

delete prepare$'$ **from** $[O, O]$;

enter approve **into** $[M, O]$;

end

(c$'$) **command** complete-approve-po

$(M : manager, O : purchase\text{-}order)$

if approve $\in [M, O]$ **then**

delete approve **from** $[M, O]$;

enter approve$'$ **into** $[M, O]$;

enter approve$'$ **into** $[O, O]$;

end

(d) **command** begin-agree-to-po

$(P : project\text{-}leader, O : purchase\text{-}order)$

if approve$' \in [O, O] \wedge$ requisition$' \in [P, O]$ **then**

delete approve$'$ **from** $[O, O]$;

enter agree **into** $[P, O]$;

end

(d$'$) **command** complete-agree-to-po

$(P : project\text{-}leader, O : purchase\text{-}order)$

if agree $\in [P, O]$ **then**

delete agree **from** $[P, O]$;

enter agree$'$ **into** $[P, O]$;

enter agree$'$ **into** $[O, O]$;

end

(e) **command** begin-reapprove-po

$(M : manager, O : purchase\text{-}order)$

if agree$' \in [O, O] \wedge$ approve$' \in [M, O]$ **then**

delete agree$'$ **from** $[O, O]$;

enter reapprove **into** $[P, O]$;

end

(e$'$) **command** complete-reapprove-po

$(M : manager, O : purchase\text{-}order)$

if reapprove $\in [M, O]$ **then**

delete reapprove **from** $[M, O]$;

enter reapprove$'$ **into** $[M, O]$;

enter reapprove′ **into** $[O, O]$;

end

(f) **command** begin-issue-po

$(C : clerk, O : purchase\text{-}order)$

if reapprove′ $\in [O, O] \wedge$ prepare′ $\notin [C, O]$ **then**

delete reapprove′ **from** $[O, O]$;

enter issue **into** $[C, O]$;

end

(f′) **command** complete-issue-po

$(C : clerk, O : purchase\text{-}order)$

if issue $\in [C, O]$ **then**

delete issue **from** $[C, O]$;

enter issue′ **into** $[C, O]$;

enter issue′ **into** $[O, O]$;

end

In commands (d) and (e), a check is made to ensure coincidence of duties. Thus the same project-leader who makes the requisition agrees to the subsequent form of the requisition. Also, the same supervisor who approves the requisition does the reapproval after the project-leader has indicated agreement. In command (f), on the other hand, a check is made for absence of an access right, thus ensuring separation of duties.

5.1.3 Persistent Objects

We now turn our attention to persistent objects. We propose the following transaction control expression for representing the potential history of an account.

create • supervisor;
{debit • clerk + credit • clerk};
close • supervisor;

The curly parenthesis denote repetition while '+' gives a choice on each repetition. The idea is that an account is created, thereafter repeatedly debited or credited, and at some point closed. Any object whose transaction control expression contains indefinite repetition is, by definition, a persistent object. Similarly any object whose transaction control expression does not contain repetition is, by definition, transient.

The history of a persistent object may be lengthy. It is impractical to convert the transaction control expression incrementally into an history, as done for transient objects. We can realistically have only some abbreviated history for persistent objects available to the access control system. Fortunately, it is improper to require that all transactions executed on a persistent object be performed by distinct users. An account may have hundreds of debit and credit operations, while the organization employs only a few dozen clerks. Separation of duties carried to this extreme will paralyze the organization. The fundamental principle is that transactions are executed on persistent objects only as the side effect of executing them on transient objects [San88c]. Separation of duties can be enforced by keeping the following history information.

1. The entire history of transient objects.

2. A partial fixed length history of persistent objects for non-repetitive portions of the transaction control expression.

For the account example, assume that Dick is the supervisor who creates the account, as a side effect of executing a transaction on some transient object. The TCE of the account is modified to record this fact as follows.

create • Dick;
{debit • clerk + credit • clerk};
close • supervisor;

Thereafter, as debit and credit transactions are executed on the account, again as a side effect, the expression remains unmodified. Finally when the account is closed by some supervisor other than Dick, say Jerry, this fact is recorded in the TCE to give us the following.

create • Dick;
{debit • clerk + credit • clerk};
close • Jerry;

There is a separation of duty involved in creating and closing the account. But separation of duty in debiting and crediting it is enforced only to the extent specified in the transaction control expressions on the transient objects related to this account.

There is no great difficulty in implementing the transient object/permanent object TCE distinction in augmented TAM. The general rule is that there must be some TAM object created for each transaction on which separation of duties needs to be enforced. In the above example, the create right for an account is given to Dick, and the absence of the create right in the cell [Jerry, account] allows Jerry to obtain the close right for that same account.

For the repetitive debit or credit operations, a separate voucher subject is created each time, and transactions as illustrated in earlier examples can manipulate the column of the access matrix associated with the voucher leading up to a debit or credit on the account when the check is issued. To relate the voucher subject to the account in question, the account can be tied to the voucher subject at the time the voucher is created. For example consider the classic example of the previous section where a voucher is prepared by a clerk, approved by a supervisor and issued by a different clerk. Than commands (a′), (c) and (c′) given in that section are modified

to relate the voucher to an account and the modified commands are given below. Three new additional rights assign, debit and debit′ are introduced along with a new subject type account. The right assign is used to associate every voucher in the system to a particular account and a right debit in an account indicates that a voucher has been prepared to be issued and hence money is ready to be debited from the account to which the voucher is assigned. The modified new command (a′) also associates the created voucher to an account by introducing the right assign in $[V, A]$. The modified command (c) also enters right debit in $[C, A]$ indicating that the clerk is ready to debit money from the account and command (c′) also debits money from account as the voucher has been issued by introducing right debit′ in $[A, A]$.

(a′) **command** complete-prepare-voucher
$(C : clerk, V : voucher, A : account)$
if prepare $\in [C, V]$ **then**
delete prepare **from** $[C, V]$;
enter prepare′ **into** $[C, V]$;
enter prepare′ **into** $[V, V]$;
enter assign **into** $[V, A]$;
end

(c) **command** begin-issue-check
$(C : clerk, V : voucher, A : account)$
if approve′ $\in [V, V] \wedge$ prepare′ $\notin [C, V]$ **then**
delete approve′ **from** $[V, V]$;
enter issue **into** $[C, V]$;
enter debit **into** $[C, A]$;
end

(c′) **command** complete-issue-check

$(C : clerk, V : voucher, A : account)$

if issue $\in [C, V] \wedge$ assign $\in [V, A]$ **then**

delete issue **from** $[C, V]$;

enter issue′ **into** $[C, V]$;

enter issue′ **into** $[V, V]$;

delete debit **from** $[C, A]$;

enter debit′ **into** $[A, A]$;

end

The stipulation in [San88c], that Dick cannot approve vouchers for accounts that he has created, can be easily accommodated. Consider the same classic example. Command (b) is modified as given below. Also we also assume that when Dick creates an account, he gets a creator right for that account. All other commands remain as they are.

(b) **command** begin-approve-voucher

$(S : supervisor, V : voucher, A : account)$

if prepare′ $\in [V, V] \wedge$assign $\in [V, A] \wedge$ creator $\notin [S, A]$

then

delete prepare′ **from** $[V, V]$;

enter approve **into** $[S, V]$;

end

The new command (b) checks whether the voucher is prepared. It also checks whether the supervisor who is going to approve the voucher is not the one who created the account to which the voucher is assigned to.

5.2 Automatic Translation of TCEs

In this section, we provide some general observations on the implementation of TCEs in augmented TAM by automated translation. It is clear that any translation scheme, based on the examples of section 5.1, must accommodate at least the following.

- A subject, such as a voucher, must be created to serve as the communication channel by which the TCE proceeds. The communication subject effectively stores a "program counter" for the TCE and controls which operation can occur next. At creation time, communication subjects for transient objects can be tied to related subjects, such as accounts and responsible users, for persistent objects. When a transaction is complete, the communication subject can be destroyed, typically after audit information has been archived.

- As part of the conditional test in the ATAM commands for successive operations in a transaction control expression, satisfactory completion of prior steps must be checked. Such checking is done by consulting the rights stored for the communication subject for the TCE.

- Separation of duties is enforced by explicitly checking for the absence of a particular right or set of rights. The specific checks are easily determined by examining the prior operations in the transaction control expression.

- Conversely, coincidence of duties, i.e., when the same principal must perform two or more tasks, is enforced by explicitly checking for the presence of a particular right or set of rights. Again, the specific checks are easily determined by examining the prior operations in the transaction control expression.

- Voting is achieved by either multiple ATAM commands or by disjunction in the conditional test of a ATAM command. In general, a concise voting expression

in a transaction control expression may result in a combinatorial number of resulting ATAM commands.

- The invocation of a ATAM command with a given set of arguments must imply real world agreement by the users represented by those arguments. For instance, when a ATAM command enters approve into the $[S, V]$ cell, there must be assurance that the action represents the supervisor's instructions, and not some malicious party. Briefly, authentication issues require attention in an actual implementation.

5.3 Conclusion on the use of testing for absence of rights

In this chapter we have analyzed the implementation for transaction control expressions in the augmented typed access matrix model. Transaction control expressions are important because they provide a natural mechanism for the specification of separation of duties applications. The main result of this chapter is that, to implement transaction control expressions in the access matrix model, the models should have the ability to check for the absence of access rights. Such checks are practically outside the expressive power of nonmonotonic HRU and (unaugmented) TAM. Examples of translations of transaction control expressions into the augmented access matrix (augmented TAM) were given, and general considerations in the translation were outlined.

Chapter 6

Expressive Power of ATAM and its Variations

This chapter compares the expressive power of ATAM and its variations. In particular it compares the expressive power of Augmented SOTAM (SO-ATAM) and Unary-ATAM (U-ATAM) with ATAM. In section 6.1, we prove that SO-ATAM is strongly equivalent to ATAM. In section 6.2, we prove that U-ATAM is strongly equivalent to ATAM in expressive power. Since U-ATAM is a restricted version of B-ATAM, we conclude from this chapter that $ATAM \equiv SO\text{-}ATAM \equiv B\text{-}ATAM \equiv U\text{-}ATAM$. Since all the equivalences shown in this chapter are strong, we will understand equivalence to mean strong equivalence throughout this chapter.

6.1 Expressive Power of Augmented SOTAM

This section compares the expressive power of ATAM and Augmented SOTAM (SO-ATAM). We prove that ATAM is strongly equivalent to SO-ATAM. Since SO-ATAM is a restricted version of ATAM, every SO-ATAM system is also an ATAM system. To establish equivalence we therefore need to show that every ATAM system can be strongly simulated by an SO-ATAM system. For ease of exposition, and understanding, we develop the construction in several phases. In section 6.1.1, we first show that ATAM systems without **create** or **destroy** operations can be reduced to SO-ATAM systems. In section 6.1.2, we then show how ATAM systems with **create** and **destroy** operations can be reduced to SO-ATAM systems.

6.1.1 Equivalence Without Create and Destroy Operations

It is helpful to approach the ATAM to SO-ATAM simulation by first looking at monotonic systems. Recall that a scheme is monotonic if it does not delete any rights, and does not destroy subjects or objects. An important fact in monotonic systems is that once the condition for a command is satisfied with respect to a given set of actual parameters, no evolution of the protection state can cause the condition to become false. In other words, once a command is authorized it will always remain authorized in the future.

Given any monotonic ATAM scheme, we can therefore get an equivalent monotonic SO-ATAM scheme as follows. Each ATAM command that modifies n columns is simulated by n SO-ATAM commands. Each of these SO-ATAM commands has the same condition as the original ATAM command, but each SO-ATAM command modifies exactly one of the columns modified by the original ATAM command. It is easy to see that every sequence of ATAM operations can be simulated by the corresponding SO-ATAM operations. Conversely, any sequence of SO-ATAM operations corresponds to a sequence of ATAM operations some of which may only be partially completed. However, the SO-ATAM sequence can be extended to complete all the partial ATAM operations. Therefore the two systems are equivalent.

The construction outlined above does not extend to non-monotonic systems. In a non-monotonic system, operations which are currently authorized may have their preconditions falsified due to deletion of access rights by other non-monotonic operations. At the same time, in SO-ATAM we have no choice but to simulate ATAM commands which modify multiple columns with multiple commands. The key to doing this successfully is to prevent other ATAM operations from interfering with the execution of a given ATAM operation. The simplest way to do this is to ensure that ATAM operations can be executed in the SO-ATAM simulation only one

at a time. To do this we need to synchronize the execution of successive ATAM commands in the SO-ATAM simulation. This synchronization is done with the help of a subject SNC of type *snc*, where *snc* is distinct from any type in the given ATAM system. SNC is also used to sequentialize the execution of ATAM operations, and to sequentialize the multiple SO-ATAM operations needed to simulate a given ATAM operation.

Every ATAM subject or object is simulated in the SO-ATAM system as a subject (i.e., every column has a corresponding row in the access matrix). In other words the access matrix of the SO-ATAM system is square. This entails no loss of generality, since ATAM subjects are not necessarily active entities.

The SO-ATAM system also contains the following rights, in addition to the rights defined in the given ATAM system.

- $\{token, token'\}$
- $\{p_{i,j} \mid j = 1 \ldots n$, for each ATAM command C_i (where C_i has n parameters)$\}$

All these rights occur only in the SNC row and column. It is assumed, without loss of generality, that these rights are distinct from the rights in the given ATAM system.

The initial state of the SO-ATAM system consists of the initial state of the ATAM system augmented in two respects. First, an empty row is introduced for every ATAM object, which does not have a row in the given ATAM access matrix. Secondly, the SNC subject is introduced in the access matrix with $[SNC, SNC] = \{token\}$.

In the absence of **creates** and **destroys**, the body of a ATAM command with n parameters can be rearranged to have the following structure.

command $C_i(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n)$
  if $\alpha(S_1, S_2, \ldots, S_n)$ **then**

enter in/delete from column S_1;

enter in/delete from column S_2;

...

enter in/delete from column S_n;

end

That is, the primitive operations occur sequentially on a column-by-column basis. Of course, some of the columns may have no operations, being referenced only in the condition part; but for the general case we assume the above structure.

Let us suppose the above ATAM command is invoked with actual parameters $S_1, S_2, \ldots, S_n$.[1] This operation will be simulated by several SO-ATAM operations. The simulation proceeds in three phases, respectively illustrated in figures 6.1, 6.2 and 6.3. In these figures we show only the relevant portion of the access matrix, and only those rights introduced specifically for the SO-ATAM simulation. It is understood that the ATAM rights are distributed exactly as in the ATAM system.

The first phase consists of a single SO-ATAM command C_i-I which tests whether (i) the condition of the ATAM command $\alpha(S_1, S_2, \ldots, S_n)$ is true, and (ii) whether $token \in [SNC, SNC]$. The former test is obviously required. The latter ensures that the SO-ATAM simulation of $C_i(S_1, S_2, \ldots, S_n)$ can begin only if no other ATAM operation is currently being simulated. It also ensures that once phase I of the simulation of $C_i(S_1, S_2, \ldots, S_n)$ has started, the simulation will proceed to completion before simulation of another ATAM command can begin. In other words ATAM operations are simulated serially, with no interleaving. The phase I command is as follows.

[1] For convenience and readability, we are using the same symbols for the formal parameters of the command C_i, as well as for the actual parameters of a particular invocation of C_i. The context will make it clear whether the symbol S_i refers to a formal or actual parameter.

command C_i-$I(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$

if $\alpha(S_1, S_2, \ldots, S_n) \wedge token \in [SNC, SNC]$ **then**

enter $p_{i,1}$ **in** $[S_1, SNC]$;

enter $p_{i,2}$ **in** $[S_2, SNC]$;

$\ldots$

enter $p_{i,n}$ **in** $[S_n, SNC]$;

delete $token$ **from** $[SNC, SNC]$;

end

The body of this command enters $p_{i,j}$ in $[S_j, SNC]$ for $j = 1 \ldots n$, signifying that S_j is the j-th parameter of the ATAM command being simulated. The command also removes the *token* right from $[SNC, SNC]$. The states of the access matrix, before and after execution of C_i-I, are outlined in figures 6.1(a) and 6.1(b) respectively. We call the rights $p_{i,j}$ as *parametric rights*.

In phase II of the simulation there are n commands, one each for the number of columns modified in C_i. Each of these n commands modifies a particular column by testing for a predicate. The predicate of these commands test for the *parametric rights* of modified cells. Later in this section, we illustrate how these n commands are given by the construction by showing an example of how a simple ATAM command *atam* is simulated by our construction. This example indicates how the commands in phase II can be easily derived. The matrix after execution of commands in phase II is given in figure 6.2. Where ever *operations* is in a cell, this simply indicates that operations in that cell have been done.

Finally in Phase III there are two commands. The first command checks if all the operations have been done in all the n columns by testing for absence and presence of rights in all the n columns. If the operations have been done, it enters a right $token'$ in $[SNC, SNC]$, indicating that the body of the ATAM command

	SNC	S_1	S_2	...	S_n
SNC	$token$				
S_1					
S_2					
...					
S_n					

(a)

	SNC	S_1	S_2	...	S_n
SNC					
S_1	$p_{i,1}$				
S_2	$p_{i,2}$				
...					
S_n	$p_{i,n}$				

(b)

Figure 6.1: SO-ATAM Simulation of the n-Parameter ATAM Command C_i: Phase I

have been executed. We illustrate how this command is given by the construction by giving an example of how a simple ATAM command *atam* can be simulated by our construction. The matrix after the execution of this command is similar to the one in figure 6.3(a). And finally the second command in phase III deletes all the bookkeeping rights in SNC column by testing for the right $token'$. The matrix at the end of phase III resembles figure 6.3(b). The final command in Phase III is given below. The condition $\pi(S_1, S_2, \ldots, S_n, SNC)$ is nothing but

$$p_{i,1} \in [S_1, SNC] \wedge p_{i,2} \in [S_2, SNC] \wedge \ldots \wedge p_{i,n} \in [S_n, SNC]$$

command C_i-$III(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$

  if $\pi(S_1, S_2, \ldots, S_n, SNC) \wedge token' \in [SNC, SNC]$ **then**

    delete $p_{i,1}$ **from** $[S_1, SNC]$;

	SNC	S_1	S_2	...	S_n
SNC		*operations*	*operations*		*operations*
S_1	$p_{i,1}$	*operations*	*operations*		*operations*
S_2	$p_{i,2}$		*operations*	*operations*	*operations*
...					
S_n	$p_{i,n}$			*operations*	*operations*

Figure 6.2: SO-ATAM Simulation of the n-Parameter ATAM Command C_i: Phase II

delete $p_{i,2}$ **from** $[S_2, SNC]$;

...

delete $p_{i,n}$ **from** $[S_n, SNC]$;

delete $token'$ **from** $[SNC, SNC]$;

enter $token$ **in** $[SNC, SNC]$;

end

The SO-ATAM system is now ready to simulate another ATAM command.

We now illustrate how the ATAM command *atam* given below can be simulated in SO-ATAM. The ATAM command *atam* tests and modifies two columns.

command $atam(S_1 : s_1, S_2 : s_2, S_3 : s_3)$

if $x \in [S_1, S_1] \wedge y \notin [S_2, S_2]$ **then**

delete x **from** $[S_1, S_1]$;

enter y **in** $[S_3, S_1]$;

enter x **in** $[S_2, S_1]$;

enter y **in** $[S_2, S_2]$;

enter y **in** $[S_1, S_2]$;

end

	SNC	S_1	S_2	...	S_n
SNC	$token'$				
S_1	$p_{i,1}$				
S_2	$p_{i,2}$				
...					
S_n	$p_{i,n}$				

(a)

	SNC	S_1	S_2	...	S_n
SNC	$token$				
S_1					
S_2					
...					
S_n					

(b)

Figure 6.3: SO-ATAM Simulation of the n-Parameter ATAM Command C_i: Phase III

The phase I command to simulate *atam* is given below. This command tests for the predicate of *atam* and for *token* in $[SNC, SNC]$. If the condition is true it deletes *token* and enters rights $p_{i,1}$, $p_{i,2}$, $p_{i,3}$ in $[S_1, SNC]$, $[S_2, SNC]$ and $[S_3, SNC]$ indicating that S_1, S_2, S_3 are the three parameters.

command *atam-I*$(S_1 : s_1, S_2 : s_2, S_3 : s_3, SNC : snc)$

if $x \in [S_1, S_1] \wedge y \notin [S_2, S_2] \wedge token \in [SNC, SNC]$ **then**

delete *token* **from** $[SNC, SNC]$;

enter $p_{i,1}$ **in** $[S_1, SNC]$;

enter $p_{i,2}$ **in** $[S_2, SNC]$;

enter $p_{i,3}$ **in** $[S_2, SNC]$;

end

Phase II commands are given below. Each command modifies a column. As *atam* modifies two columns there are two SO-ATAM commands. These commands just check for the parameter rights of the modified cells and then modifies the column based on these rights.

command *atam-II-1*$(S_1 : s_1, S_2 : s_2, S_3 : s_3, SNC : snc)$

if $p_{i,1} \in [S_1, SNC] \wedge p_{i,2} \in [S_2, SNC] \wedge p_{i,3} \in [S_3, SNC]$ **then**

delete x **from** $[S_1, S_1]$;

enter x **in** $[S_2, S_1]$;

enter y **in** $[S_3, S_1]$;

end

command *atam-II-2*$(S_1 : s_1, S_2 : s_2, SNC : snc)$

if $p_{i,1} \in [S_1, SNC] \wedge p_{i,2} \in [S_2, SNC]$ **then**

enter y **in** $[S_2, S_2]$;

enter y **in** $[S_1, S_2]$;

end

Phase III commands are given below. The first command tests if the two columns are modified by checking for the presence/absence of rights in the two columns (and also by using the parameters rights) and if so it enters a right $token'$ in $[SNC, SNC]$. Finally the last command deletes all the rights from the SNC column and enters right $token$ back into $[SNC, SNC]$.

command *atam-III-1*$(S_1 : s_1, S_2 : s_2, S_3 : s_3, SNC : snc)$

if $p_{i,1} \in [S_1, SNC] \wedge p_{i,2} \in [S_2, SNC] \wedge p_{i,3} \in [S_3, SNC] \wedge x \notin [S_1, S_1] \wedge y \in [S_3, S_1] \wedge x \in [S_2, S_1] \wedge y \in [S_2, S_2] \wedge y \in [S_1, S_2]$ **then**

enter $token'$ **in** $[SNC, SNC]$;

end

command *atam-III-2*$(S_1 : s_1, S_2 : s_2, SNC : snc)$

if $p_{i,1} \in [S_1, SNC] \wedge p_{i,2} \in [S_2, SNC] \wedge p_{i,3} \in [S_3, SNC] \wedge token' \in [SNC, SNC]$

then

delete $token'$ **from** $[SNC, SNC]$;

delete $p_{i,1}$ **from** $[S_1, SNC]$;

delete $p_{i,2}$ **from** $[S_2, SNC]$;

delete $p_{i,3}$ **from** $[S_3, SNC]$;

enter $token$ **in** $[SNC, SNC]$;

end

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

Theorem 5 For every ATAM system A the construction outlined above produces an equivalent SO-ATAM system B.

Proof: In order to prove that the construction outlined above produces an equivalent SO-ATAM system B, it is enough to show that SO-ATAM system B can simulate ATAM system A. In order to show this, we need to show that definition 27 is satisfied. Every command of an ATAM system is mapped to a single partial order of commands (the partial order corresponds to the successful execution) and also satisfying condition 1 of definition 27. The construction ensures that conditions 2 to 5 are satisfied by simulating a single ATAM command at a time and by ensuring that once a simulation of a command has started, the simulation will proceed to completion before simulation of another ATAM command can begin. Construction also ensures by modifying each column in phase II in such a way that condition 6 of definition 27 is satisfied at both intermediate and completion states. □

6.1.2 Equivalence With Create and Destroy Operations

We now consider ATAM with create and destroy operations. There are several ways in which the construction of section 6.1.1 can be extended to allow for creation and destruction. We describe one of them here. We have already assumed that the SO-ATAM system will have a square access matrix, in which every object (column) is also a subject (row). So our focus will be on subject creation.

The primitive ATAM operation "**create subject** S_i" introduces an empty row and column in the access matrix for the newly created subject S_i. An SO-ATAM command which has this primitive operation in its body is quite restricted in what it can do, since all **enter** and **delete** operations will be confined to the new column S_i. We will therefore simulate creation in SO-ATAM in two steps.

- First we will allow unconditional creation of subjects to occur. However, the created subject will be dormant, indicated by the *dormant* right in the diagonal cell $[S_i, S_i]$. We view the unconditional creation as occurring on demand as needed.

- Dormant subjects will be brought to life by replacing *dormant* in the diagonal cell by *alive*.

A dormant subject cannot be a parameter in any ATAM command. This will be ensured by modifying the ATAM system so that each ATAM command tests for the *alive* right in the diagonal cells for every parameter in the command.

Consider a ATAM command which requires m pre-existing subjects or objects, $S_1, \ldots, S_m$, and creates $n \Leftrightarrow m$ subjects or objects, $S_{m+1}, \ldots, S_n$. We will modify the given ATAM command as follows. Let $\alpha(S_1, \ldots, S_m)$ be the given condition in this command. This condition will be supplemented by the tests $alive \in [S_i, S_i]$, for

$i = 1 \ldots m$. It will be further supplemented by the tests $dormant \in [S_i, S_i]$, for $i = m+1 \ldots n$. All create operations in the body of the original ATAM command will be discarded. Instead the dormant subjects will be made alive by the operations

enter $alive$ **in** $[S_j, S_j]$;
delete $dormant$ **from** $[S_j, S_j]$

for $j = m+1 \ldots n$.

The **destroy** operation can be similarly removed from the body of the given ATAM commands, and relegated to a background "garbage collection" activity. To do this every "**destroy** S_j" primitive operation is replaced in the ATAM command by the following operations

enter $dead$ **in** $[S_j, S_j]$;
delete $alive$ **from** $[S_j, S_j]$;

The meaning of $dead \in [S_j, S_j]$ is that S_j has effectively been destroyed, since it cannot function as a parameter in any ATAM command. The background garbage collection can be done by introducing the following command for every type s.

command expunge($S : s$)
  if $dead \in [S, S]$ **then**
    destroy subject S;
end

In this manner the original ATAM command has been reduced to one which only has **enter** and **delete** operations in its body. The construction of the SO-ATAM simulation can now proceed as in section 6.1.1.

6.2 Expressive Power of Unary-ATAM

This section proves that U-ATAM is equivalent to ATAM in terms of expressive power. Since U-ATAM is a restricted version of ATAM, every U-ATAM system is also a ATAM system. To establish the equivalence of U-ATAM and ATAM, we therefore need to show that every ATAM system can be simulated by a U-ATAM system. For ease of exposition, and understanding, we show the equivalence in two phases. The section 6.2.1 gives a construction which converts a given ATAM system without create and destroy operations into an equivalent U-ATAM system and the section 6.2.2 explains how the construction given in section 6.2.1 can be extended to include create and destroy operations.

6.2.1 Equivalence Without Create and Destroy Operations

An ATAM command in general tests multiple cells and modifies multiple cells. Consider a typical ATAM command C_i given below.

command C_i $(S_1 : s_1, S_2 : s_2, \ldots, O_1 : o_1, O_2 : o_2, \ldots, O_n : o_n)$
  if $\alpha(S_1, S_2, \ldots, S_n, O_1, \ldots, O_n)$ **then**
  operations in multiple cells;
end

The command's predicate $\alpha(S_1, S_2, \ldots, S_n, O_1, \ldots, O_n)$ tests for absence/presence of rights in say m cells and modifies n cells. This command doesn't have any create or destroy operations. The next subsection explains how commands with create and destroy operations can be simulated in U-ATAM. As an U-ATAM command can only test for one cell, a single ATAM command like C_i should be simulated by more than one U-ATAM command.

The construction to prove the equivalence of U-ATAM and ATAM is now explained. In practice we can often employ simpler constructions.

The U-ATAM system contains a subject SNC of type snc, where snc is distinct from any type in the given ATAM system. SNC is used to sequentialize the execution of ATAM operations, and to sequentialize the multiple U-ATAM operations needed to simulate a given ATAM operation. Also for every command in the ATAM system, two unique types are defined in the U-ATAM system. The existence of a subject of one type indicates that the predicate for a command which corresponds to that type is true and the existence of a subject of another type indicates that the predicate for that command is false.

The U-ATAM system also contains the following rights, in addition to the rights defined in the given ATAM system.

- $\{token\}$
- $\{c_i \mid i = 1 \ldots k$, where there are K commands$\}$
- $\{r_1, r_2, \ldots, r_m, {r_1}^f, {r_2}^f, \ldots, {r_m}^f, l_1, \ldots, l_n\}$

It is assumed, without loss of generality, that these rights are distinct from the rights in the given ATAM system. The rights $r_1, \ldots, r_m$ are derived from the fact that C_i tests for m cells and the presence of each of these rights in $[SNC, SNC]$ indicates that the predicate is true in a particular cell. Similarly ${r_i}^f, {r_2}^f, \ldots, {r_m}^f$ indicate that the predicate is false. The rights $l_1, \ldots, l_n$ are derived from the fact that C_i modifies n cells and these rights are used to modify the n cells.[2]

[2]The value of m and n is obtained from the fact that in the ATAM system, there does not exist any command which tests for more than m cells and similarly there does not exist a command which modifies more than n cells.

The initial state of the U-ATAM system consists of the initial state of the ATAM system augmented in two respects. First, an empty row is introduced for every U-ATAM object, which does not have a row in the given ATAM access matrix. In other words the access matrix of the U-ATAM system is square. This entails no loss of generality, since ATAM subjects are not necessarily active entities. Secondly, the SNC subject is introduced in the access matrix with a right *token* in $[SNC, SNC]$.

Each command in ATAM is simulated by multiple commands in U-ATAM which are divided into four phases. We now illustrate the construction by giving the U-ATAM commands which simulate the ATAM command C_i. The simulation of this ATAM command is done in four phases.

In the first phase, the command tests for the presence of $token \in [SNC, SNC]$. This ensures that the U-ATAM simulation can begin only if no other ATAM operation is currently being simulated. It also ensures that once phase I of the simulation of C_i has started, the simulation of another ATAM command cannot begin until the simulation of C_i is complete. The body of this command enters a right r_x ($x = 1$ to m) in each cell which the ATAM command C_i tests and enters a right l_y ($y = 1$ to n) in each cell which the ATAM command modifies. i.e, if C_i tests m cells and modifies n cells, then rights $r_1, r_2, \ldots, r_m$ (along with right c_i) are entered one each in the tested cells and rights $l_1, \ldots, l_n$ (along with right c_i) are entered one each in the modified cells. It also removes the *token* right from $[SNC, SNC]$ and enters the right c_i in $[SNC, SNC]$ to indicate the simulation of C_i is in progress. The phase I command is given below.

command *uatam-*C_i*-I* $(S_1 : s_1, S_2 : s_2, \ldots, O_1 : o_1, O_2 : o_2, \ldots, O_n : o_n, SNC : snc)$

if $token \in [SNC, SNC]$ **then**

enter rights (r_i or l_i) and c_i in every cell which is tested and modified in C_i;

delete *token* from $[SNC, SNC]$;

enter c_i in $[SNC, SNC]$;

end

The commands in the second phase test for the predicate of C_i. If the predicate is true, then a new subject CT is created which indicates that the predicate of the command is true. If the predicate is false, then a new subject CF is created which indicates that the predicate of the command is false. The commands in the second phase are divided again into two stages, successful and failed stages.

The commands in the successful stage of phase II are given below. These commands create a subject CT if the predicate of the command they are simulating is true. The command *utam-II-C_i-test-i-Successful* actually represents m commands (for each value of $i = 1$ to m) for each of the m cells tested in C_i (The subject and object type in the command is indicated by s_j and o_k, but actually they represent the type of the subject and object of the cell being tested). These commands test if the predicate is true (by testing each cell and in the commands below the predicate in each cell is represented by P) and if so enter rights r_x in $[SNC, SNC]$ to indicate that the predicate is true in the xth cell tested in C_i. Finally the command *utam-II-C_i-test-Complete-Successful* in phase II checks if the predicate is true in C_i by testing for all the rights $r_1, \ldots, r_m$ in $[SNC, SNC]$. If the predicate is true, it deletes all those rights from $[SNC, SNC]$ and creates a subject CT of type *ct* to indicate that the condition of C_i is true.

command *utam-II-C_i-test-i-Successful* $(S_j : s_j, O_k : o_k, SNC : snc)$

if $r_i \in [S_j, O_k] \wedge c_i \in [S_j, O_k] \wedge P$ in $[S_j, O_k]$ **then**

enter r_i in $[SNC, SNC]$;

delete r_i from $[S_j, O_k]$;

delete c_i from $[S_j, O_k]$;

end

command *utam-II-C_i-test-Complete-Successful* $(SNC : snc, CT : ct)$

if $r_1 \in [SNC, SNC] \wedge r_2 \in [SNC, SNC] \ldots r_m \in [SNC, SNC] \wedge c_i \in [SNC, SNC]$

then

delete r_1 from $[SNC, SNC]$;

...

delete r_m from $[SNC, SNC]$;

create CT of type ct;

end

The commands in the failed stage of phase II are given below. These commands create a subject CF if the predicate of the command they are simulating is false. The command *u-atam-II-C_i-test-i-Failure* actually represents m commands (for each value of $i = 1$ to m) for each of the m cells tested in C_i (The subject and object type in the command is indicated by s_j and o_k, but actually they represent the type of the subject and object of the cell being tested). These commands test if the predicate is true (by testing each cell and in the below commands $\neg P$ represents that the predicate is false in that cell) and if not enter rights ${r_x}^f$ in $[SNC, SNC]$ to indicate that the predicate is false in the xth cell tested in C_i. Finally the command *u-atam-II-C_i-test-Complete-Failure* in phase II checks if the predicate of C_i is false by checking for , and if so, it deletes all those rights from $[SNC, SNC]$ and creates a subject CF of type cf to indicate that the condition of C_i is false. The predicate , is represented by the following expression $({r_1}^f \cup r_1 \in [SNC, SNC]) \wedge ({r_2}^f \cup r_2 \in [SNC, SNC]) \ldots$

$({r_m}^f \cup r_m \in [SNC, SNC]) \wedge c_i \in [SNC, SNC] \wedge \Delta$, where Δ represents that one of ${r_1}^f, {r_2}^f, \ldots, {r_m}^f \in [SNC, SNC]$. Here we allow this slight abuse of notation to simplify the presentation.

command *u-atam-II-C_i-test-i-Failure* $(S_j : s_j, O_k : o_k, SNC : snc)$
if $r_i \in [S_j, O_k] \wedge c_i \in [S_j, O_k] \wedge \neg P$ in $[S_j, O_k]$ **then**
    enter ${r_i}^f$ in $[SNC, SNC]$;
    delete r_i from $[S_j, O_k]$;
    delete c_i from $[S_j, O_k]$;
end

command *u-atam-II-C_i-test-Complete-Failure* $(SNC : snc, CF : cf)$
if , **then**
    delete ${r_1}^f$ from $[SNC, SNC]$;
    ...
    delete ${r_m}^f$ from $[SNC, SNC]$;
    create CF of type cf;
end

In the third phase, the body of C_i is executed if the predicate is true and if the predicate is false, then all the bookkeeping rights $l_1, l_2, \ldots, l_n$ are removed. The commands in phase III are given below. The command *utam-III-C_i-body-i* actually represents n commands (for each of the n cells being modified). Every command modifies a different cell and indicate that the modification in a particular cell is done by entering a right in $[SNC, SNC]$. If the predicate is false, then each of the commands *u-atam-III-C_i-body-i-failed* delete one of the bookkeeping rights $l_1, \ldots, l_n$.

command *utam-III-C_i-body-i* $(S_j : s_j, O_k : o_k, SNC : snc, CT : ct)$

if $l_i \in [S_j, O_k]$ **then**

operations in $[S_j, O_k]$;

delete l_i from $[S_j, O_k]$;

enter l_i in $[SNC, SNC]$;

end

command *u-atam-III-C_i-body-i-failed* $(S_j : s_j, O_k : o_k, SNC : snc, CF : cf)$

if $l_i \in [S_j, O_k]$ **then**

delete l_i from $[S_j, O_k]$;

enter l_i in $[SNC, SNC]$;

end

In the final phase, there are two commands which indicate that the simulation of the ATAM command is either successfully done or successfully failed. The command *utam-IV-C_i-Successfully-done* deletes all the bookkeeping rights from $[SNC, SNC]$ and enters *token* back into $[SNC, SNC]$ to indicate that simulation of the TAM command C_i is done. The command *u-atam-IV-C_i-Successfully-Failure* deletes all the bookkeeping rights from $[SNC, SNC]$ and enters *token* back into $[SNC, SNC]$ to indicate that simulation of the ATAM command C_i is not successfully done.

command *utam-IV-C_i-Successfully-done* $(SNC : snc, CT : ct)$

if $l_1 \in [SNC, SNC] \wedge l_2 \in [SNC, SNC] \ldots \wedge l_n \in [SNC, SNC]$ **then**

delete all rights from $[SNC, SNC]$;

enter *token* in $[SNC, SNC]$;

destroy CT of type ct;

end

command *u-atam-IV-C_i-Successfully-Failure* $(SNC : snc, CF : cf)$

if $l_1 \in [SNC, SNC] \wedge l_2 \in [SNC, SNC] \ldots \wedge l_n \in [SNC, SNC]$ **then**

delete all rights from $[SNC, SNC]$;

enter *token* in $[SNC, SNC]$;

destroy CF of type cf;

end

We now explain how a ATAM command with create and destroy operations is simulated in U-ATAM.

6.2.2 Equivalence With Create and Destroy Operations

We will now explain, how creation and destroy operations can be simulated by U-ATAM commands by extending the construction given in the previous section.

The U-ATAM system also contains the rights given below, in addition to the rights defined earlier.

- $\{alive, dormant\}$
- $\{d_i \mid i = 1 \ldots x$, if a maximum of x subjects are destroyed in a command$\}$
- $\{cr_i \mid i = 1 \ldots y$, if there are operations in y cells of the created subjects$\}$

The commands are similar to the commands in section 6.2.1 with the minor modifications explained below.

When a subject S_d is destroyed in the ATAM command C_i, then a right d_i is entered (in phase I) in $[S_d, S_d]$ to indicate that the S_d should be destroyed. The subject is destroyed in phase III by checking for a right d_i in $[S_d, S_d]$.

When a subject S_c is created in the ATAM command and when there are modifications (enter operations) in some cells of S_c, then in the U-ATAM system, the subject is created in phase I, and right *dormant* is entered in $[S_c, S_c]$ along with a right cr_i in all the cells where operations occur in ATAM. In phase II, once the condition of the ATAM is successful, then along with the creation of a subject CT, the right *dormant* is deleted and a right *alive* is entered in $[S_c, S_c]$. In the third phase the enter operations are executed by testing for rights cr_i.

The rest of the commands are similar to the commands given in the previous section.

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

Theorem 6 For every ATAM system A the construction outlined above produces an equivalent U-ATAM system B.

Proof: In order to prove that the construction outlined above produces an equivalent U-ATAM system B, it is enough to show that U-ATAM system B can simulate ATAM system A. In order to show this, we need to show that definition 27 is satisfied. Every command of an ATAM system is mapped to two partial orders of commands and also satisfying condition 1 of definition 27. The construction ensures that conditions 2 to 5 are satisfied by simulating a single ATAM command at a time and by ensuring that once a simulation of a command has started, the simulation will proceed to completion before simulation of another ATAM command can begin. Construction also ensures by modifying each cell at a time in phase III in such a way that condition 6 of definition 27 is satisfied at both intermediate and completion states. □

Chapter 7

Expressive Power of TAM and its Variations

This chapter compares the expressive power of TAM and its variations. In particular it compares the expressive power of TAM, SOTAM, BTAM and UTAM. In section 7.1 we prove that SOTAM and TAM are strongly equivalent and this result has appeared in [SG93a]. In section 7.2, we conjecture that UTAM and BTAM (in general KTAM) are not strongly equivalent to TAM in terms of expressive power, but we prove that they are weakly equivalent to TAM.

7.1 Expressive Power of SOTAM

This section proves the formal equivalence of the expressive power of TAM and SOTAM. The construction of chapter 6 cannot be used as that construction would require the ability to test for absence of rights. As SOTAM cannot test for absence of rights, we need a different construction to prove the equivalence of TAM and SOTAM. Since SOTAM is a restricted version of TAM, every SOTAM system is also a TAM system. To establish equivalence we therefore need to show that every TAM system can be simulated by a SOTAM system. The construction to do this is quite intricate. For ease of exposition, and understanding, we develop the construction in several phases. First, in section 7.1.1, we identify an essential synchronization protocol which is a critical part of the overall construction. Then, in section 7.1.2, we show that TAM systems without **create** or **destroy** operations can be reduced to SOTAM systems.

Finally we show, in section 7.1.3, how TAM systems with **create** and **destroy** operations can be reduced to SOTAM systems.

7.1.1 Two Column Synchronization Protocol

It is helpful to approach the TAM to SOTAM simulation by first looking at monotonic systems. Recall that a scheme is monotonic if it does not delete any rights, and does not destroy subjects or objects. In monotonic systems, if a command is authorized it will always remain authorized in the future.

Given any monotonic TAM scheme, we can therefore get an equivalent monotonic SOTAM scheme as follows. Each TAM command that modifies n columns is simulated by n SOTAM commands. Each of these SOTAM commands has the same condition as the original TAM command, but each SOTAM command modifies exactly one of the columns modified by the original TAM command. It is easy to see that every sequence of TAM commands can be simulated by the corresponding SOTAM commands. Conversely, any sequence of SOTAM commands corresponds to a sequence of TAM commands some of which may only be partially completed. However, the SOTAM sequence can be extended to complete all the partial TAM commands. Therefore the two systems are equivalent.

The construction outlined above does not extend to non-monotonic systems. In a non-monotonic system, commands which are currently authorized may have their conditions falsified due to deletion of access rights by other non-monotonic operations. At the same time, in SOTAM we have no choice but to simulate TAM commands which modify multiple columns with multiple commands. The key to doing this successfully is to prevent other TAM commands from interfering with the execution of a given TAM command. The simplest way to do this is to ensure that TAM commands can be executed in the SOTAM simulation only one at a time. To do this

	SNC	S_1	S_2	S_3	...
SNC	0	0	0	0	...
S_1		*token*	*next*		...
S_2				*next*	...
S_3					...
...	...	...	...	...	...

(a) S_1 possesses the token

	SNC	S_1	S_2	S_3	...
SNC		0	0	0	...
S_1			*next*		...
S_2			*token*	*next*	...
S_3					...
...	...	...	...	...	...

(b) The token has been transferred to S_2

Figure 7.1: Two Column Synchronization

we need to synchronize the execution of successive TAM commands in the SOTAM simulation (as described in section 7.1.2).

Thus, the problem of simulating TAM in SOTAM requires solution of a synchronization problem. Moreover, the synchronization must be achieved using SOTAM commands which can modify only one column at a time. This effectively rules out standard synchronization solutions based on semaphores, locks, or similar mechanisms. In effect we have to achieve synchronization without having shared global variables that are writable by concurrent processes.

The basic synchronization problem, which we call *two column synchronization*, is illustrated in figure 7.1. The solution to this problem turns out to be critical in constructing a SOTAM simulation of a TAM system. For the moment ignore the SNC row and column in figure 7.1. In figure 7.1(a) subject S_1 possesses the token,

represented by the *token* right in the $[S_1, S_1]$ cell. After S_1 is done using the token, it is passed on to S_2 as indicated in figure 7.1(b). The *next* right in the $[S_1, S_2]$ cell serves to connect S_1 to S_2 in sequence, indicating that the token is to be passed from S_1 to S_2. Similarly, S_2 will pass the token on to S_3 in turn. The exact manner in which the sequence of token passing is encoded in the access matrix is not material to the synchronization problem. For illustrative purposes we have adopted the scheme described here. The construction of section 7.1.2 uses a slightly different technique. For the moment we can ignore typing and, for simplicity, treat all entities as being of the same type s.

The TAM command for solving the two column synchronization problem is straightforward, as follows.

command *transfer-token*$(S_1 : s, S_2 : s)$
  if $token \in [S_1, S_1] \wedge next \in [S_1, S_2]$ **then**
    delete *token* **from** $[S_1, S_1]$;
    enter *token* **in** $[S_2, S_2]$;
end

This command modifies both columns S_1 and S_2, and is therefore not a SOTAM command.

The transfer-token TAM command can be simulated by four SOTAM commands, which use the SNC row in the access matrix to synchronize. We use three rights denoted 0, 1, and 2, for this purpose. Only one of these rights can be present at a time in a $[SNC, S_i]$ cell, and they do not occur outside of the SNC row. Normally each column S_i has $0 \in [SNC, S_i]$ indicating the quiescent state with respect to the synchronization commands. The meaning of $1 \in [SNC, S_i]$ is that S_i is ready to pass the token. The meaning of $2 \in [SNC, S_j]$ is that S_j is ready to receive the token.

The four SOTAM commands to simulate the *transfer-token* TAM command are as follows.

command *transfer-token-1*(S_1, SNC)

  if $token \in [S_1, S_1]$ **then**

    delete $token$ **from** $[S_1, S_1]$;

    delete 0 **from** $[SNC, S_1]$;

    enter 1 **in** $[SNC, S_1]$;

end

command *transfer-token-2*(S_1, S_2, SNC)

  if $1 \in [SNC, S_1] \wedge next \in [S_1, S_2]$ **then**

    delete 0 **from** $[SNC, S_2]$;

    enter 2 **in** $[SNC, S_2]$;

end

command *transfer-token-3*(S_1, S_2, SNC)

  if $1 \in [SNC, S_1] \wedge 2 \in [SNC, S_2] \wedge next \in [S_1, S_2]$ **then**

    delete 1 **from** $[SNC, S_1]$;

    enter 0 **in** $[SNC, S_1]$;

end

command *transfer-token-4*(S_1, S_2, SNC)

  if $0 \in [SNC, S_1] \wedge 2 \in [SNC, S_2] \wedge next \in [S_1, S_2]$ **then**

    delete 2 **from** $[SNC, S_2]$;

    enter 0 **in** $[SNC, S_2]$;

    enter $token$ **in** $[S_2, S_2]$;

end

The correctness of these commands is intuitively obvious, and a formal proof could be easily given. Also note that the **enter** operation in the *transfer-token-4* command can be modified to enter $token'$, rather than $token$, in $[S_2, S_2]$. In this way we can pass a modified token from one column to another by means of SOTAM commands.

We call the protocol described by these four commands as the two column synchronization protocol. As we will see this protocol is repeatedly invoked in the constructions of this paper.

7.1.2 Equivalence Without Create and Destroy

We now prove the equivalence of TAM and SOTAM in the absence of create and destroy operations. This is done by giving a procedure to construct a SOTAM system that can simulate any given TAM system. Every TAM subject or object is simulated in the SOTAM system as a subject (i.e., every column has a corresponding row in the access matrix). In other words the access matrix of the SOTAM system is square. This entails no loss of generality, since TAM subjects are not necessarily active entities.

As in previous simulations in this thesis, the SOTAM system contains a subject SNC of type *snc*, where *snc* is distinct from any type in the given TAM system. The role of SNC is to enable two column synchronization, as discussed in section 7.1.1. As we will see, SNC is also used to sequentialize the execution of TAM commands, and to sequentialize the multiple SOTAM commands needed to simulate a given TAM command.

The SOTAM system also contains the following rights, in addition to the rights defined in the given TAM system.

- $\{0, 1, 2, token, token'\}$
- $\{p_{i,j} \mid j = 1 \ldots n$, for each TAM command C_i (where C_i has n parameters)$\}$

Except for *token*, these rights occur only in the SNC row and column. The *token* right also occurs in the diagonal cells of the SOTAM access matrix. It is assumed, without loss of generality, that these rights are distinct from the rights in the given TAM system.

The initial state of the SOTAM system consists of the initial state of the TAM system augmented in two respects. First, an empty row is introduced for every TAM object, which does not have a row in the given TAM access matrix. Secondly, the SNC subject is introduced in the access matrix with $[SNC, SNC] = \{token, 0\}$, and $[SNC, S_i] = \{0\}$ for all subjects $S_i \neq SNC$.

In the absence of **creates** and **destroys**, the body of a TAM command with n parameters can be rearranged to have the following structure.

command $C_i(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n)$
  if $\alpha(S_1, S_2, \ldots, S_n)$ **then**
    enter in/delete from column S_1;
    enter in/delete from column S_2;
    ...
    enter in/delete from column S_n;
end

That is, the primitive operations occur sequentially on a column-by-column basis. Of course, some of the columns may have no operations, being referenced only in the condition part; but for the general case we assume the above structure.

Let us suppose the above TAM command is invoked with actual parameters $S_1, S_2, \ldots, S_n$. This command will be simulated by several SOTAM commands. The simulation proceeds in three phases, respectively illustrated in figures 7.2, 7.3 and 7.4. In these figures we show only the relevant portion of the access matrix, and only

those rights introduced specifically for the SOTAM simulation. It is understood that the TAM rights are distributed exactly as in the TAM system.

The first phase consists of a single SOTAM command C_i-I which tests whether (i) the condition of the TAM command $\alpha(S_1, S_2, \ldots, S_n)$ is true, and (ii) whether $token \in [SNC, SNC]$. The former test is obviously required. The latter ensures that the SOTAM simulation of $C_i(S_1, S_2, \ldots, S_n)$ can begin only if no other TAM command is currently being simulated. It also ensures that once phase I of the simulation of $C_i(S_1, S_2, \ldots, S_n)$ has started, the simulation will proceed to completion before simulation of another TAM command can begin. In other words TAM commands are simulated serially, with no interleaving. The phase I SOTAM command is as follows.

command C_i-$I(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$
  if $\alpha(S_1, S_2, \ldots, S_n) \wedge token \in [SNC, SNC]$ **then**
    enter $p_{i,1}$ **in** $[S_1, SNC]$;
    enter $p_{i,2}$ **in** $[S_2, SNC]$;
    ...
    enter $p_{i,n}$ **in** $[S_n, SNC]$;
    delete $token$ **from** $[SNC, SNC]$;
    delete 0 **from** $[SNC, SNC]$;
    enter 1 **in** $[SNC, SNC]$;
end

The body of this command enters $p_{i,j}$ in $[S_j, SNC]$ for $j = 1 \ldots n$, signifying that S_j is the j-th parameter of the TAM command being simulated. It also removes the $token$ right from $[SNC, SNC]$, and replaces 0 in $[SNC, SNC]$ by 1 signifying that the $token$ can be moved from SNC column. The states of the access matrix, before and after execution of C_i-I, are outlined in figures 7.2(a) and 7.2(b) respectively.

	SNC	S_1	S_2	...	S_n
SNC	$token$, 0	0	0		0
S_1					
S_2					
...					
S_n					

(a)

	SNC	S_1	S_2	...	S_n
SNC	1	0	0		0
S_1	$p_{i,1}$				
S_2	$p_{i,2}$				
...					
S_n	$p_{i,n}$				

(b)

Figure 7.2: SOTAM Simulation of the n-Parameter TAM Command C_i: Phase I

In phase II of the simulation the $token$ right is passed, in turn, from $[SNC, SNC]$ to $[S_1, S_1]$ to $[S_2, S_2]$, and so on to $[S_n, S_n]$. Each passage requires a total of four SOTAM commands based on the two-column synchronization protocol of section 7.1.1.[1] The details of the protocol are shown here only for the transfer of the token from column S_j to S_{j+1}. The SOTAM commands for transferring the token from SNC to S_1, and from S_n to SNC are not explicitly shown, since they are so similar.

Let $\pi(S_1, S_2, \ldots, S_n, SNC)$ represent the following predicate.

$$p_{i,1} \in [S_1, SNC] \wedge p_{i,2} \in [S_2, SNC] \wedge \ldots \wedge p_{i,n} \in [S_n, SNC]$$

The changes in column S_j in the original TAM command are carried out by a SOTAM command whose condition tests for $\pi(S_1, S_2, \ldots, S_n, SNC)$, and the presence of the

[1] One difference in detail is that the connection between consecutive columns is achieved by means of the $p_{i,j}$ rights in the SNC column, rather than by the $next$ right in the $[S_j, S_{j+1}]$ cell.

token in $[S_j, S_j]$. The SOTAM command which carries out the changes in column S_j also removes *token* from $[S_j, S_j]$, and replaces 0 in $[SNC, S_j]$ by 1 signifying that the token can be moved to the next column. The SOTAM command for simulating changes by the TAM command in column j is shown below.

command C_i-II-j-1$(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$
  if $\pi(S_1, S_2, \ldots, S_n, SNC) \wedge token \in [S_j, S_j]$ **then**
    enter in/delete from column S_j;
    delete *token* **from** $[S_j, S_j]$;
    delete 0 **from** $[SNC, S_j]$;
    enter 1 **in** $[SNC, S_j]$;
end

The two-column synchronization protocol, begun above, is then completed to move *token* to $[S_{j+1}, S_{j+1}]$, using the following commands.

command C_i-II-j-2$(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$
  if $1 \in [SNC, S_j] \wedge \pi(S_1, S_2, \ldots, S_n, SNC)$ **then**
    delete 0 **from** $[SNC, S_{j+1}]$;
    enter 2 **in** $[SNC, S_{j+1}]$;
end

command C_i-II-j-3$(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$
  if $1 \in [SNC, S_j] \wedge 2 \in [SNC, S_{j+1}] \wedge \pi(S_1, S_2, \ldots, S_n, SNC)$ **then**
    delete 1 **from** $[SNC, S_j]$;
    enter 0 **in** $[SNC, S_j]$;
end

command C_i-II-j-4$(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$

if $0 \in [SNC, S_j] \wedge 2 \in [SNC, S_{j+1}] \wedge \pi(S_1, S_2, \ldots, S_n, SNC)$ **then**

delete 2 **from** $[SNC, S_{j+1}]$;

enter 0 **in** $[SNC, S_{j+1}]$;

enter *token* **in** $[S_{j+1}, S_{j+1}]$;

end

In this manner, the simulation of the TAM command proceeds on a column-by-column basis, as depicted in figure 7.3.

Finally, when the token is passed from $[S_n, S_n]$ back to $[SNC, SNC]$, the two-column synchronization passes it back as the *token′* right, instead of *token*, as shown in figure 7.4(a). At this point the SNC column is cleaned out to the state of figure 7.4(b) using the following SOTAM command.

command C_i-$III(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$

if $\pi(S_1, S_2, \ldots, S_n, SNC) \wedge token' \in [SNC, SNC]$ **then**

delete $p_{i,1}$ **from** $[S_1, SNC]$;

delete $p_{i,2}$ **from** $[S_2, SNC]$;

...

delete $p_{i,n}$ **from** $[S_n, SNC]$;

delete *token′* **from** $[SNC, SNC]$;

enter *token* **in** $[SNC, SNC]$;

end

The SOTAM system is now ready to simulate another TAM command.

Note that a n-parameter TAM command requires $4(n + 1) + 1$ (or $4n + 5$) SOTAM commands in this construction. The SOTAM simulation operates on $n + 1$

	SNC	S_1	S_2	...	S_n
SNC	$token$, 0	0	0		0
S_1	$p_{i,1}$				
S_2	$p_{i,2}$				
...					
S_n	$p_{i,n}$				

(a)

	SNC	S_1	S_2	...	S_n
SNC	0	0	0		0
S_1	$p_{i,1}$	$token$			
S_2	$p_{i,2}$				
...					
S_n	$p_{i,n}$				

(b)

	SNC	S_1	S_2	...	S_n
SNC	0	0	0		0
S_1	$p_{i,1}$				
S_2	$p_{i,2}$		$token$		
...					
S_n	$p_{i,n}$				

(c)

	SNC	S_1	S_2	...	S_n
SNC	0	0	0		0
S_1	$p_{i,1}$				
S_2	$p_{i,2}$				
...					
S_n	$p_{i,n}$				$token$

(d)

Figure 7.3: SOTAM Simulation of the n-Parameter TAM Command C_i: Phase II

	SNC	S_1	S_2	...	S_n
SNC	$token'$, 0	0	0		0
S_1	$p_{i,1}$				
S_2	$p_{i,2}$				
...					
S_n	$p_{i,n}$				

(a)

	SNC	S_1	S_2	...	S_n
SNC	$token$, 0	0	0		0
S_1					
S_2					
...					
S_n					

(b)

Figure 7.4: SOTAM Simulation of the n-Parameter TAM Command C_i: Phase III

columns consisting of the n columns in the TAM command, and the SNC column. The modification of each column is bundled with the first step of the two-column synchronization protocol, giving us $4(n+1)$ commands. One command is required to clean out the SNC column at the end. Various optimizations are possible. Significantly, the SOTAM simulation is linear in the size of the TAM command being simulation.

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

Theorem 7 For every TAM system A the construction outlined above produces an equivalent SOTAM system B.

Proof: In order to prove that the construction outlined above produces an equivalent SOTAM system B, it is enough to show that SOTAM system B can simulate TAM

system A. In order to show this, we need to show that definition 27 is satisfied. Every command of an TAM system is mapped to a single partial order of commands (the partial order corresponds to the succesfull execution) and also satisfying condition 1 of definition 27. The construction ensures that conditions 2 to 5 are satisfied by simulating a single TAM command at a time and by ensuring that once a simulation of a command has started, the simulation will proceed to completion before simulation of another TAM command can begin. Construction also ensures by modifying each column in phase II in such a way that condition 6 of definiton 27 is satisfied at both intermediate and completion states. □

7.1.3 Expressive Power With Create and Destroy

We now consider TAM with create and destroy operations. There are several ways in which the construction of section 7.1.2 can be extended to allow for creation and destruction. One way is to use the construction of section 6.1.2. The same construction can be used with the minor modification that whenever a subject S_i is created a right 0 is also entered in $[SNC, S_i]$.

In any case we repeat the construction of 6.1.2 here to prove that TAM is equivalent to SOTAM.

The primitive TAM operation "**create subject** S_i" introduces an empty row and column in the access matrix for the newly created subject S_i. A SOTAM command which has this primitive operation in its body is quite restricted in what it can do, since all **enter** and **delete** operations will be confined to the new column S_i. We will therefore simulate creation in SOTAM in two steps.

- First we will allow unconditional creation of subjects to occur. However, the created subject will be dormant, indicated by the *dormant* right in the diago-

nal cell $[S_i, S_i]$. The unconditional creation also introduces the 0 right in the $[SNC, S_i]$ cell. We view the unconditional creation as occurring on demand as needed.

- Dormant subjects will be brought to life by replacing *dormant* in the diagonal cell by *alive*.

A dormant subject cannot be a parameter in any TAM command. This will be ensured by modifying the TAM system so that each TAM command tests for the *alive* right in the diagonal cells for every parameter in the command.

Consider a TAM command which requires m pre-existing subjects or objects, $S_1, \ldots, S_m$, and creates $n \Leftrightarrow m$ subjects or objects, $S_{m+1}, \ldots, S_n$. We will modify the given TAM command as follows. Let $\alpha(S_1, \ldots, S_m)$ be the given condition in this command. This condition will be supplemented by the tests $alive \in [S_i, S_i]$, for $i = 1 \ldots m$. It will be further supplemented by the tests $dormant \in [S_i, S_i]$, for $i = m+1 \ldots n$. All create operations in the body of the original TAM command will be discarded. Instead the dormant subjects will be made alive by the operations

enter *alive* **in** $[S_j, S_j]$;
delete *dormant* **from** $[S_j, S_j]$

for $j = m+1 \ldots n$.

The **destroy** operation can be similarly removed from the body of the given TAM commands, and relegated to a background "garbage collection" activity. To do this every "**destroy** S_j" primitive operation is replaced in the TAM command by the following operations

enter *dead* **in** $[S_j, S_j]$;
delete *alive* **from** $[S_j, S_j]$;

The meaning of $dead \in [S_j, S_j]$ is that S_j has effectively been destroyed, since it cannot function as a parameter in any TAM command. The background garbage collection can be done by introducing the following command for every type s.

command expunge($S : s$)
  if $dead \in [S, S]$ **then**
    destroy subject S;
end

In this manner the original TAM command has been reduced to one which only has **enter** and **delete** operations in its body. The construction of the SOTAM simulation can now proceed as in section 7.1.2.

7.2 Expressive Power of Unary-TAM and KTAM

This section compares the expressive power of UTAM with TAM. We argue in this section that UTAM is not strongly equivalent to TAM in terms of expressive power and is weakly equivalent to TAM in terms of expressive power. In section 7.2.1 we discuss why the construction of section 6.2 cannot be used to prove the equivalence of UTAM and TAM and it then discusses how the construction can be extended to prove the weak equivalence of TAM and UTAM. In the same section we also conjecture that UTAM and TAM are not strongly equivalent.

7.2.1 Weak Equivalence of TAM and UTAM

In this section, we discuss why the construction of section 6.2 cannot be used to prove the equivalence of UTAM and TAM and it then discusses how the construction can be extended to prove the weak equivalence of TAM and UTAM.

A TAM command in general tests multiple cells and modifies multiple cells. Consider a typical TAM command C_i given below.

command C_i $(S_1 : s_1, S_2 : s_2, \ldots, O_1 : o_1, O_2 : o_2, \ldots, O_n : o_n)$
  if $\alpha(S_1, S_2, \ldots, S_n, O_1, \ldots, O_n)$ **then**
  operations in multiple cells
end

The command's predicate $\alpha(S_1, S_2, \ldots, S_n, O_1, \ldots, O_n)$ tests for presence of rights in say m cells and modifies n cells. This command is similar to the ATAM command C_i given in section 6.2 except that the predicate cannot test for absence of rights.

The construction of section 6.2 simulates each ATAM command by multiple commands in U-ATAM. These commands in U-ATAM are divided into four phases. The command in phase I ensures that once phase I of the simulation of C_i has started, the simulation of another ATAM command cannot begin until the simulation of C_i is complete. The commands in the second phase test for the predicate of the ATAM command. If the predicate is true, they create some unique subject indicating that the predicate is true. In the third phase the body of the ATAM command is executed and in the final phase, the right *token* is entered back in $[SNC, SNC]$ If the predicate is false, a different unique subject is created. In the third and final phases, if the predicate is false, then all the bookkeeping rights are removed and the right *token* is entered back in $[SNC, SNC]$

The same construction cannot be used to prove the equivalence of TAM and UTAM. This is due to the fact that commands in phase II do check for absence of rights to check if the predicate is false. Since UTAM cannot test for absence of rights, these commands cannot be used. If the same construction is used with only commands

concerned with the predicate of the TAM command being true (omitting the phase III commands which deal with predicate being false), then the UTAM system can get into deadlocks. For example in the UTAM system, the phase I command deletes *token* from $[SNC, SNC]$ to simulate some command α. If the predicate of α is true, then the simulation of α proceeds to the final phase where it would enter a right *token* back in $[SNC, SNC]$ to indicate that the simulation of α is complete and that simulation of some other command can now proceed. If the predicate of the command α is false, then the UTAM system gets deadlocked as it cannot check for absence of rights. Then UTAM system would not be able to enter right *token* back in $[SNC, SNC]$. It seems like UTAM system cannot simulate the TAM system due to definition 27. We now prove how the same construction can be modified to prove that UTAM and TAM are weakly equivalent.

In chapter 4, the construction of 4.1 proves that TAM and ATAM are weakly equivalent and it gives a construction which represents absence of a right in a cell by testing for presence of a complementary right. The same construction can be used to test for absence of a right by testing for presence of a complementary right.

Hence the construction of 6.2 can be augmented with the construction of section 4.1 to prove the weak equivalence of TAM and UTAM.

7.2.2 Strong Non-Equivalence Conjecture

We just discussed that the construction of section 6.2 can only be used to prove the equivalence of TAM and UTAM if UTAM does have the ability to test for absence of rights. Since, it has been proved in chapter 4 that a model which cannot test for absence of rights, can only acquire the capability with respect to weak equivalence, we conjecture that UTAM (and KTAM) and TAM are not strongly equivalent.

Chapter 8

Conclusion

This chapter lists the contributions of this thesis and presents an insight into future directions. The contributions of this thesis are presented in section 8.1 and section 8.2 gives the future research directions.

8.1 Contributions

In this thesis we compare the expressive power of Typed Access Matrix Model (TAM), Augmented TAM (ATAM) and various other models. We first define a formalism to compare the relationship between expressive power of two models. This is significant as there has not been any formalism to compare the expressive power of two models. Our formalism defines two notions of equivalence: *Strong* and *Weak*. Our formalism helps in proving whether two models are equivalent (strongly or weakly or both) or not equivalent (both strongly and weakly).

We then address the impact of adding testing for absence of rights in access control models (on expressive power) by comparing the expressive power of TAM and ATAM. We prove that adding for testing for absence of rights does actually increase the expressive power of access control models. We illustrate the practical need for testing for absence of rights by showing that implementing transaction expressions does require the ability to test for absence of rights in access control models. In particular, dynamic separation of duties requires this ability.

Expressive Power of TAM and ATAM
TAM $\not\equiv_{strongly}$ ATAM
TAM $\equiv_{weakly}$ ATAM
Expressive Power of ATAM and its Variations
ATAM $\equiv_{strongly}$ SO-ATAM
ATAM $\equiv_{strongly}$ U-ATAM
Expressive Power of TAM and its variations
TAM $\equiv_{strongly}$ SOTAM
TAM $\equiv_{weakly}$ UTAM
TAM $\not\equiv_{strongly}$ UTAM (conjecture)
TAM $\not\equiv_{strongly}$ KTAM (conjecture)

Table 8.1: Summarized Results

We also define various simpler models by posing restrictions on Typed Access Matrix Model (TAM) and Augmented TAM (ATAM). We compare the expressive power of these simpler models with TAM and ATAM. The results are summarized again in table 8.1.

The results in table 8.1 indicate that all simple models have the most general expressive power of the models they are derived from. They also indicate that simplification of models can be carried to a point, beyond which they loose their expressive power.

The expressive power results also have an impact on both safety and implementation issues. The effect of these results on implementation is positive in the sense that all the simpler models defined will have an easier implementation than the models they are defined from. But, the effect of these results on safety is not optimistic. This is due the fact that these results indicate that safety analysis of these simpler models is as difficult as the most general model they are defined from.

8.2 Future Research

Based on the research work in this thesis, we propose the following future research directions.

8.2.1 Better Simulations

Most of the constructions used in this thesis (to prove the equivalence of two models) are not efficient in the sense that they would ensure that only one command is being simulated at a time. For example, consider the construction of 6.1. It ensures that during the A-SOTAM simulation of an ATAM system, only one ATAM command is being simulated at a time. Once the ATAM command is simulated, then only would it allow simulation of another ATAM command. We would like to make these simulations more efficient by allowing concurrent execution of the simulating commands. We would like to see, if the traditional concurrency mechanisms used in databases and operating systems can be applied.

8.2.2 Safety Issues

In this thesis we proved that very simple models do have the most general expressive power of the models they are derived from. This leads to the fact that the safety analysis of these simple models is as difficult as the most general model they are derived from. Rather than finding efficient safety results for these models, it is better to consider systems independently and then evaluate their safety features.

8.2.3 Implementation Issues

In this thesis we proved that very simple models do have the most general expressive power of the models they are derived from. In future, we would like to address the

implementation of the various simpler models defined in this thesis.

Bibliography

Bibliography

[AELO90] M. Abrams, K. Eggers, L. LaPadula, and I. Olson. A generalized framework for access control: An informal description. In *13th NIST-NCSC National Computer Security Conference*, pages 135–143, 1990.

[AJP93] M. Abrams, S. Jajodia, and H. Podell, editors. *Information Security : An Integrated Collection of Essays*. IEEE Computer Society Press, 1993. To Appear.

[ALS92] P.E. Ammann, R.J. Lipton, and Ravi S. Sandhu. The expressive power of multi-parent creation in monotonic access control models. In *IEEE Computer Security Foundations Workshop*, pages 148–156, Franconia, NH, June 1992.

[AS90] P.E. Ammann and Ravi S. Sandhu. Extending the creation operation in the schematic protection model. In *Sixth Annual Computer Security Application Conference*, pages 340–348, Tucson, AZ, December 1990.

[AS91] P.E. Ammann and Ravi S. Sandhu. Safety analysis for the extended schematic protection model. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 87–97, Oakland, CA, May 1991.

[AS92a] P.E. Ammann and Ravi S. Sandhu. The extended schematic protection model. *The Journal Of Computer Security*, 1(3&4):335–384, 1992.

[AS92b] P.E. Ammann and Ravi S. Sandhu. Implementing transaction control expressions by checking for absence of access rights. In *Eighth Annual Computer Security Application Conference*, pages 131–140, San Antonio, TX, December 1992.

[AS93] P.E. Ammann and Ravi S. Sandhu. One-representative safety analysis in the non-monotonic transform model. Technical report, George Mason University, 1993.

[ASS91] P.E. Ammann, Ravi S. Sandhu, and G.S. Suri. A distributed implementation of the schematic protection model. In *Seventh Annual Computer Security Application Conference*, pages 152–164, San Antonio, TX, December 1991.

[Bis88] M. Bishop. Theft of information in the take-grant protection model. In *IEEE Computer Security Foundations Workshop*, pages 194–218, Franconia, NH, June 1988.

[Bud83] T.A. Budd. Safety in grammatical protection systems. *International Journal of Computer and Information Sciences*, 12(6):413–431, 1983.

[CW87] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings IEEE Computer Society Symposium on Security and Privacy*, pages 184–194, Oakland, CA, May 1987.

[For94] Warwick Ford. *Computer Communications Security.* Prentice-Hall, 1994.

[Gas88] M. Gasser. *Building a Secure Computer System.* Van Nostrand Reinhold, 1988.

[GD72] G.S. Graham and P.J. Denning. Protection – principles and practice. In *AFIPS Spring Joint Computer Conference*, pages 40:417–429, 1972.

[HR78] M.H. Harrison and W.L. Ruzzo. Monotonic protection systems. In DeMillo et al, editor, *Foundations of Secure Computations*, pages 337–365. Academic Press, 1978.

[HRU76] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[Jon93] Dirk Jonscher. Extending access controls with duties—realized by active mechanisms. In B. Thuraisingham and C.E. Landwehr, editors, *Database Security VI: Status and Prospects*, pages 91–111. North-Holland, 1993.

[Lam71] B.W. Lampson. Protection. In *5th Princeton Symposium on Information Science and Systems*, pages 437–443, 1971. Reprinted in *ACM Operating Systems Review* 8(1):18–24, 1974.

[LB78] R.J. Lipton and T.A. Budd. On classes of protection systems. In DeMillo et al, editor, *Foundations of Secure Computations*, pages 3281–296. Academic Press, 1978.

[LM82] A. Lockman and N. Minsky. Unidirectional transport of rights and take-grant control. *IEEE Transactions on Software Engineering*, SE-8(6):597–604, 1982.

[LS77] R.J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[McL90] J. McLean. Security models and information flow. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 180–187, Oakland, CA, May 1990.

[MS88] J.D. Moffett and M.S. Sloman. The source of authority for commercial access control. *IEEE Computer*, 21(2):59–69, 1988.

[Mur93] William H. Murray. Introduction to access controls. In Zella A. Ruthberg and Hal F. Tipton, editors, *Handbook of Information Security Management*, pages 515–523. Auerbach Publishers, 1993.

[Pit87] P. Pittelli. The bell-lapadula computer security model represented as a special case of the harrison-ruzzo-ullman model. In *NBS-NCSC National Computer Security Conference*, 1987.

[San88a] Ravi S. Sandhu. Expressive power of the schematic protection model. In *IEEE Computer Security Foundations Workshop*, pages 188–193, Franconia, NH, June 1988.

[San88b] Ravi S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, April 1988.

[San88c] Ravi S. Sandhu. Transaction control expressions for separation of duties. In *Fourth Annual Computer Security Application Conference*, pages 282–286, Orlando, FL, December 1988.

[San89a] Ravi S. Sandhu. The demand operation in the schematic protection model. *Information Processing Letters*, 32(4):213–219, September 1989.

[San89b] Ravi S. Sandhu. Transformation of access rights. In *Proceedings IEEE Computer Society Symposium on Security and Privacy*, pages 259–268, Oakland, CA, May 1989.

[San90] Ravi S. Sandhu. Mandatory controls for database integrity. In D.L. Spooner and C.E. Landwehr, editors, *Database Security III: Status and Prospects*, pages 143–150. North-Holland, 1990.

[San91] Ravi S. Sandhu. Separation of duties in computerized information systems. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 179–189. North-Holland, 1991.

[San92a] Ravi S. Sandhu. Expressive power of the schematic protection model. *The Journal Of Computer Security*, 1(1):59–98, 1992.

[San92b] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA, May 1992.

[San92c] Ravi S. Sandhu. Undecidability of safety for the schematic protection model with cyclic creates. *Journal of Computer and System Sciences*, 44(1):141–159, February 1992.

[San93] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, November 1993.

[San94] Ravi S. Sandhu. On five definitions of data integrity. In T. Keefe and C.E. Landwehr, editors, *Database Security VII: Status and Prospects*, pages 257–267. North-Holland, 1994.

[SG93a] Ravi S. Sandhu and S. Ganta. Expressive power of the single-object typed access matrix model. In *Proceedings of the Ninth Annual Computer Security Application Conference*, pages 184–194, Orlando, FL, December 1993.

[SG93b] Ravi S. Sandhu and S. Ganta. On testing for absence of rights in access control models. In *Proceedings of the Computer Security Foundations WorkshopVI*, pages 109–120, Franconia, NH, June 1993.

[SG93c] Ravi S. Sandhu and S. Ganta. Recognition and approximation of NTrees based on binary refinement. Technical report, isse-tr-93-103, George Mason University, 1993.

[SG94a] Ravi S. Sandhu and S. Ganta. On the expressive power of the unary transformation model. In *Proceedings of the third European Symposium on Research in Computer Security*, pages 301–318, Brighton, UK, November 1994.

[SG94b] Ravi S. Sandhu and S. Ganta. On the minimality of testing for rights in transformation models. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 230–241, Oakland, CA, May 1994.

[SJ90] Ravi S. Sandhu and S. Jajodia. Integrity mechanisms in database management systems. In *NIST-NCSC National Computer Security Conference*, pages 526–540, 1990.

[Sny81] L. Snyder. Theft and conspiracy in the take-grant model. *Journal of Computer and System Sciences*, 23(3):337–347, 1981.

[SS91] Ravi S. Sandhu and G.S. Suri. A distributed implementation of the transform model. In *NIST-NCSC National Computer Security Conference*, pages 177–187, Washington, D.C., October 1991.

[SS92a] Ravi S. Sandhu and G. Suri. Non-monotonic transformations of access rights. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 148–161, Oakland, CA, May 1992.

[SS92b] Ravi S. Sandhu and G.S. Suri. Implementation considerations for the typed access matrix model in a distributed environment. pages 221–235, Baltimore, MD, October 1992.

[SS94] Ravi S. Sandhu and Pierangela Samarati. Access control: Principles and practice. *IEEE Communications*, 32(9):40–48, 1994.

Vita

Srinivas V. Ganta was born on September 21, 1968, in India and is an Indian citizen. He received the B.S in Electrical Engineering from Jawaharlal Nehru Technological University, India, in 1989, the M.S in Information and Software Systems Engineering from George Mason University, Fairfax, Virginia, in 1991. From 1990 to 1996 he was actively involved in research in Information Security. During 1994-95 he received a doctoral fellowship to continue his PhD studies and at present he is a technical staff member at SETA Corporation. His current work involves defining and prototyping products based on Novell NetWare (release 4.1), Oracle (version 7), Windows NT to apply Role-Based Access Control (RBAC) in a commercial environment.

Permanent address: 670 G, R.E. Quarters, Caltex Road,

Vijayawada, AP, India-520001.

This dissertation was typeset with LaTeX[‡] by the author.

[‡] LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.