PRACTICAL TAKE-GRANT SYSTEMS: DO THEY EXIST?

A Thesis

Submitted to the Faculty

of

Purdue University

by

Matt Bishop

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 1984

to my mother, Linda Allen

VITA

ACKNOWLEDGEMENTS

This thesis could not have been written without the support and advice of Dorothy Denning and Lawrence Snyder. Larry first interested me in the Take–Grant Model, and Dorothy suggested studying the problem which led to this thesis. She also encouraged me when my spirits flagged. Both have had a very major influence on my graduate studies, and both were very cooperative in dealing with my tight time schedules.

I also owe thanks to the other member of my committee, Peter Denning, for his time and effort, again under rather tight deadlines.

Several people at Purdue contributed to the writing of my thesis. Kevin Smallwood, Larry Petersen, and Dan Reed helped me find the macros written by Jeff Brumfield to aid formatting this thesis. Subhash Agrawal encouraged my writing several utilities which made the mechanics of keeping numberings consistent very easy. The systems programmers on the staff of the Purdue University Computer Center (in particular, Jeff Schwab), the Physics Department (in particular, Mike DeMoney and Charles LaBrec), and the Department of Computer Sciences (in particular, Steve Stone and Kevin Smallwood), were quite tolerant of my (open) attempts to break system security, and discussed many security problems with me.

Many friends have made my life at Purdue enjoyable. Special thanks go to Subhash Agrawal, Sean Arthur, André Bondi, Bob Brown, Ken and Ellen Dickey, Rich Hyde, and Paul McNabb.

I completed this thesis *in absentia* from Purdue, and would like to thank everyone at Megatest Corporation for all their help. They were very understanding in letting me use Megatest's computer facilities to type and print this thesis. I would especially like to thank Dawn Maneval, Larry Mongin, and Karen Van Dusen for helping me tame the laser printer.

Research leading to this thesis was made possible by support from the National Science Foundation.

TABLE OF CONTENTS

Page

ABSTRACT

Bishop, Matt, Ph.D., Purdue University, May 1984. Practical Take-Grant Systems: Do They Exist? Major Professor: Dorothy Denning.

The Take-Grant Protection Model is a theoretical model of computer safety. This thesis considers whether this model can be applied to non-theoretical computer systems. First, a model of hierarchical systems is presented; then, some new results are presented, and three extensions are proposed. These new results and extensions are used to model reference monitors and examine an existing computer system for security weaknesses; methods of fixing these weaknesses are also derived. The thesis concludes that, with the proper extensions, the Take-Grant Protection Model can indeed be used to examine existing systems for security flaws.

Chapter I

Introduction

I.1. Statement of the Problem

In the past, studies have tended to focus on the design and implementation of secure computing systems; this began with the concept of a reference monitor [ANDE72] and grew into such projects as the MITRE security kernel for the PDP-11/45† [MILL76, SCHI75], the UCLA Data Secure UNIX‡ system for the PDP-11/45 [POPE79], the various flavors of the Kernelized Secure Operating System (KSOS) [BERS79, BROA76], and the Provably Secure Operating System (PSOS) [FEIE79]. In addition, ways to test the security of existing systems have been explored (for example, [LIND75]). The aim of this thesis is to combine the two approaches. The thesis proposes a model for proving (or disproving) the security of a system, for identifying specific security flaws when the system is not secure, and for determining what must be done to correct these flaws.

The model is based on the *Take–Grant Protection Model*, which represents a computer system as a graph showing active and passive objects and authorities in the system. Changes in the system are reflected by graph rewriting rules, which add objects, and add and delete authorities. We have extended this model to include information flow [BISH79] and shall use this extension

†PDP is a Trademark of Digital Equipment Corporation.
‡UNIX is a Trademark of Bell Laboratories.

because it has been explored and many useful properties about it are known.

One comment about the model and the extensions used here: The graph rewriting rules we use are merely one possible set. Other rules are possible; and in a later chapter, we shall change the rewriting rules in order to examine the security of the UNIX operating system

The conditions for a computer system to be secure can be stated without tying the statement to any particular set of graph rewriting rules. The introduction of new rules requires that we determine conditions under which information and authorities can be transferred. To abstract these from the rules, we define *predicates* the truth or falsity of which are the basis for defining a secure system. The formal definitions and necessary and sufficient conditions for the predicates to hold do depend on the set of rewriting rules chosen, but the conditions for security can be stated simply in terms of the predicates.

The thesis is organized into several topics. The remainder of this chapter is spent reviewing the Take–Grant Protection Model as it now exists. From that point on, new results are presented. In Chapter II we explore the concept of hierarchies and how they may be modelled in the Take–Grant Protection Model. Chapter III gives a new proof for a theorem about information flow, and then examines the idea of *theft of information*. Chapter IV extends the model even farther; in that chapter, we add the concept of *groups* or classes to the model. Chapter V gives some examples of the practical application of these extensions; we examine reference monitors and several security loopholes in an existing system, demonstrating that the extensions can be used in practise. Finally, Chapter VI summarizes what has been done, and suggests areas for future research.

I.2. The Take–Grant Model

Now that we have stated the problem, let us describe the tool we shall use. It is called the *Take–Grant Protection Model* and was first presented as a theoretical model in [JONE76]. This section will describe what the model is, and cite several important results. The reader interested in the proofs of these can consult the referenced papers.

Let a finite, directed graph called a *protection graph* represent a system to be modelled. A protection graph has two distinct kinds of vertices, called *subjects* and *objects*. Subjects are the active vertices, and (for example) can represent users; they can pass information and authority by invoking *graph rewriting rules*. Objects, on the other hand, are completely passive; they can (for example) represent files, and do nothing.

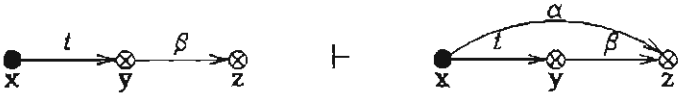
In protection graphs, the subjects are represented by ● and objects by ○. Vertices which may be either subjects or objects are represented by ⊗. Pictures are very often used to show the effects of applying a graph rewriting rule on the graph; the symbol $\vdash$ is used to mean that the graph following it is produced by the action of the graph rewriting rule on the graph preceding it. The rewriting rule itself is often written after the derived graph. The symbol $\vdash^*$ represents several rule applications. The term *witness* means a sequence of graph rewriting rules which produce the predicate or condition being witnessed, and a witness is often demonstrated by listing the graph rewriting rules that make up the witness (usually with pictures.)

The edges of a protection graph are labelled with subsets of a finite set R of rights. Suppose that $\{r,w,t,g\} \subseteq R$, where r, w, t, and g represent *r*ead, *w*rite, *t*ake, and *g*rant rights, respectively. When written as labels on a graph, the set braces are normally omitted.

The Take–Grant Model permits users with certain rights to transfer rights from one vertex to another. The rules governing the transfer of rights are called *de jure* rules [BISH79] and are as follows [JONE76]:

RULE R1.1: **take**

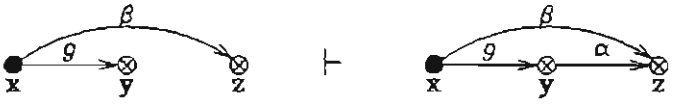
Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $t \in \gamma$, an edge from **y** to **z** labelled β, and $\alpha \subseteq \beta$. Then the *take* rule defines a new graph G_1 by adding an edge to the protection graph from **x** to **z** labelled α. Graphically,



The rule is written: **x** takes (α to **z**) from **y**.

RULE R1.2: **grant**

Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** be a subject. Let there be an edge from **x** to **y** labelled γ with $g \in \gamma$, an edge from **x** to **z** labelled β, and $\alpha \subseteq \beta$. Then the *grant* rule defines a new graph G_1 by adding an edge to the protection graph from **y** to **z** labelled α. Graphically,



The rule is written: **x** grants (α to **z**) to **y**.

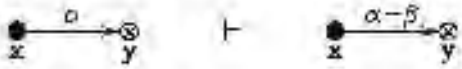
RULE R1.3: **create**

Let **x** be any subject in a protection graph G_0 and let α be a subset of R. *Create* defines a new graph G_1 by adding a new vertex **y** to the graph and an edge from **x** to **y** labelled α. Graphically,

The rule is written: **x** creates (α to new) vertex **y**.

RULE R1.4: *remove*

Let **x** and **y** be any distinct vertices in a protection graph G_1 such that **x** is a subject. Let there be an explicit edge from **x** to **y** labelled α, and let β be any subset of R. Then *remove* defines a new graph G_2 by deleting the β labels from α. If α becomes empty as a result, the edge itself is deleted. Graphically,

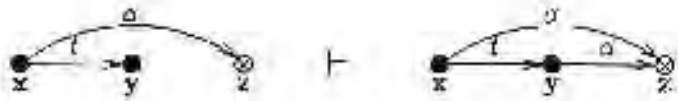


The rule is written: **x** removes (β to) **y**.

The edges which appear in the above graphs are called *explicit* because they represent authority known to the protection system.

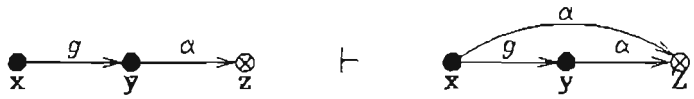
Note that there is a duality between the take and grant rules when the edge labelled t or g is between two subjects. Specifically, with the cooperation of both subjects, rights can be transmitted backwards along the edges. The following two lemmas demonstrate this:

LEMMA 1.1: Let there be an edge labelled t from one subject **x** to another subject **y**, and let **x** have α rights over a vertex **z**. Then **y** may obtain α rights over **z**. Graphically,



PROOF: See [JONES75]. ■

LEMMA 1.2: Let there be an edge labelled g from one subject **x** to another subject **y**, and let **y** have α rights over a vertex **z**. Then **x** may obtain α rights over **z**. Graphically,



PROOF: See [JONE76]. ■

As a result, when considering the transfer of authority between subjects, neither direction nor label of the edge is important, so long as the label is in the set $\{t,g\}$.

In order to state the main theorem concerning the transfer of authority between subjects, we need some preliminary definitions ([JONE76], [LIPT77]):

DEFINITION D1.1: A *tg–path* is a nonempty sequence $v_0, \ldots, v_k$ of distinct vertices such that for all i, $0 \le i < k$, v_i is connected to v_{i+1} by an edge (in either direction) with a label containing t or g.

DEFINITION D1.2: Vertices are *tg–connected* if there is a tg–path between them.

DEFINITION D1.3: An *island* is a maximal tg–connected subject–only subgraph.

Any right that one vertex in an island has can be obtained by any other vertex in that island. In other words, an island is a maximal set of subject–only vertices which possess common rights.

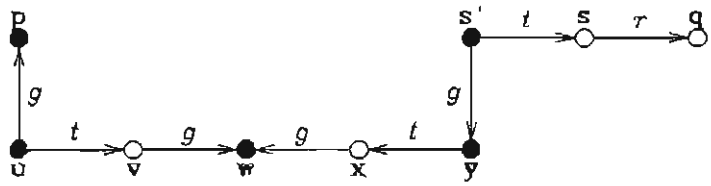
With each tg–path, associate one or more words over the alphabet $\{ \vec{t}, \overleftarrow{t}, \vec{g}, \overleftarrow{g} \}$ in the obvious way. If the path has length 0, then the associated word is the null word ν.

DEFINITION D1.4: A vertex v_0 *initially spans* to v_k if v_0 is a subject and there is a tg–path between v_0 and v_k with associated word in $\{ \vec{t}, \vec{g} \} \cup \{\nu\}$.

DEFINITION D1.5: A vertex v_0 *terminally spans* to v_k if v_0 is a subject and there is a tg–path between v_0 and v_k with associated word in $\{ \vec{t}^* \}$.

DEFINITION D1.6: A *bridge* is a tg–path with v_0 and v_k both subjects and the path's associated word in $\{ \vec{t}^*, \vec{t}^*, \vec{t}^* g \vec{t}^*, \vec{t}^* \overleftarrow{g} \vec{t}^* \}$.

An initial span is a tg–path along which the first vertex in the path can transmit authority; a terminal span is a tg–path along which the first vertex in the path can acquire authority. As a note, a bridge is said to be *directed away from* v_0. The following diagram illustrates these terms:



islands:	I_1={p,u},I_2={w},I_3={y,s'}
bridges:	u,v,w and w,x,y
initial span:	p with associated word: ν
terminal span:	s',s with associated word: $\vec{t}$

The following predicate formally defines the notion of *transferring authority*:

DEFINITION D1.7: The predicate *can•share*(α, **x**, **y**, G_0) is true for a right α and two vertices **x** and **y** if and only if there exist protection graphs $G_1, \ldots, G_n$ such that $G_0 \vdash^* G_n$ using only *de jure* rules, and in G_n there is an edge from **x** to **y** labelled α.

In short, if **x** can acquire α rights to **y**, then can•share(α, **x**, **y**, G_0) is true. The theorem which establishes necessary and sufficient conditions for this predicate to hold is:

THEOREM 1.3: The predicate can•share(α, **x**, **y**, G_0) is true if and only if there is an edge from **x** to **y** in G_0 labelled α, or if the following hold simultaneously:

C1.1. there is a vertex **s** $\in G_0$ with an **s**-to-**y** edge labelled α;

C1.2. there exist subject vertices **p'** and **s'** such that

a. **p'** initially spans to **x**, and

b. **s'** terminally spans to **s**;

C1.3. there exist islands $I_1, \ldots, I_v$ such that **p'** is in I_1, **s'** is in I_v, and there is a bridge from I_j to I_{j+1} $(1 \leq j < v)$.

PROOF: See [LIPT77]. ■

The *de jure* rules control the transfer of authority only; they say nothing about the transfer of information. The two are clearly different; for example, if a user is shown a document containing information which he does not have authority to read, the information has been transfered to the user. The *de jure* rules do not model cases like this. Instead, we use a different set of rules, called *de facto* rules, to derive paths along which information may flow.

In order to describe transfers of information, we cannot use explicit edges, because no change in authority occurs. Still, some indication of the paths along which information can be passed is necessary. Hence, we use a dashed line, labelled by r, to represent the path of a potential *de facto* transfer. Such an edge is called an *implicit* edge. Notice that implicit edges cannot be manipulated by the *de jure* rules, since the *de jure* rules can affect only authorities recorded in the protection system, and implicit edges do not represent such authority.

A protection graph records all authorities as explicit edges, so when a *de jure* rule is used to add a new edge, an actual transfer of authority has taken place. But when a *de facto* rule is used, a path along which information can be transferred is exhibited; the actual transfer may, or may not, have occurred. It is impossible to tell this from the graph, because the graph records authorities and *not* information. For the purposes of this model, however, we shall assume

that if it is possible for information to be transferred from one vertex to another, such a transfer has in fact occurred.

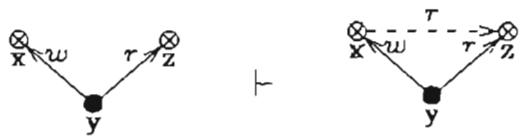
One set of proposed *de facto* rules was introduced in [BISH79] to model the transfer of information. Although these are not the only rules possible, their effects have been explored, and so we shall use them.

RULE R1.5: **post**

Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0 and let **x** and **z** be subjects. Let there be an edge from **x** to **y** labelled α, where $r \in \alpha$, and an edge from **z** to **y** labelled β, where $w \in \beta$. Then the *post* rule defines a new graph G_1 with an implicit edge from **x** to **z** labelled $\{r\}$. Graphically,

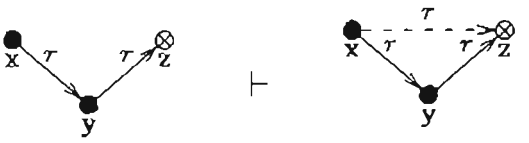
RULE R1.6: **pass**

Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **y** be a subject. Let there be an edge from **y** to **x** labelled α, where $w \in \alpha$, and an edge from **y** to **z** labelled β, where $r \in \beta$. Then the *pass* rule defines a new graph G_1 with an implicit edge from **x** to **z** labelled $\{r\}$. Graphically,



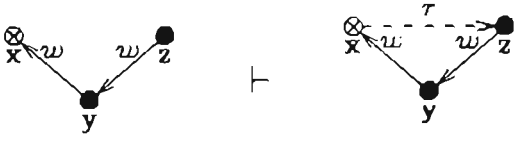
RULE R1.7: **spy**

Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **x** and **y** be subjects. Let there be an edge from **x** to **y** labelled α, where $r \in \alpha$, and an edge from **y** to **z** labelled β, where $r \in \beta$. Then the *spy* rule defines a new graph G_1 with an implicit edge from **x** to **z** labelled $\{r\}$. Graphically,



RULE R1.8: find

Let **x**, **y**, and **z** be three distinct vertices in a protection graph G_0, and let **y** and **z** be subjects. Let there be an edge from **y** to **x** labelled α, where $w \in \alpha$, and an edge from **z** to **y** labelled β, where $w \in \beta$. Then the *find* rule defines a new graph G_1 with an implicit edge from **x** to **z** labelled $\{r\}$. Graphically,



Note that these rules add implicit and not explicit edges. Further, as these rules model information flow, they *can* be used when either (or both) of the edges between **x** and **y**, or **y** and **z**, are implicit.

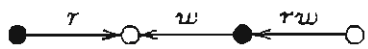
Now, consider the conditions necessary for a potential *de facto* transfer to exist in a graph.

DEFINITION D1.8: The predicate *can•know•f*(**x**, **y**, G_0) is true if and only if there exists a sequence of graphs $G_1, \ldots, G_n$ $(0 \leq n)$, such that $G_i \vdash^a G_{i+1}$ $(0 \leq i < n)$ by one of the *de facto* rules and in G_n either a **x**-to-**y** edge labelled r exists or a **y**-to-**x** edge labelled w exists and if the edge is explicit, its source is a subject.

Intuitively, can•know•f(**x**, **y**, G_0) is true if and only if **x** has the authority to read **y**, **y** has the authority to write to **x**, or an implicit edge from **x** to **y** can be added by means of the *de facto* rules. Note the duality of read and write. If **x** can write to **y**, then **y** effectively can read **x**. All **x** has to do is write to **y** any information that **y** wants to see. This duality will play an important role in later results.

DEFINITION D1.9: An *rw–path* is a nonempty sequence $v_0, \ldots, v_k$ of distinct vertices such that for all i, $0 \le i < k$, v_i is connected to v_{i+1} by an edge (in either direction) with a label containing an r or a w.

With each rw–path, associate one or more words over the alphabet $\{ \vec{r}, \overleftarrow{r}, \vec{w}, \overleftarrow{w} \}$ in the obvious way; for instance, the protection graph



has associated $\vec{r}\,\vec{w}\,\overleftarrow{r}$ and $\vec{r}\,\vec{w}\,\overleftarrow{w}$. If the path has length 0, then the associated word is the null word ν.

DEFINITION D1.10: An rw–path $v_0, \ldots, v_k$, $k \ge 1$, is an *admissible rw–path* if and only if:

C1.4. it has an associated word $\alpha_1 \alpha_2 \cdots \alpha_k$ in the regular language $(\vec{r} \cup \vec{w})^*$; and

C1.5. if $\alpha_i = \vec{r}$ then v_{i-1} is a subject and if $\alpha_i = \vec{w}$ then v_i is a subject.

Note that there cannot be two consecutive objects on an rw–admissible path.

THEOREM 1.4: Let **x** and **y** be vertices in a protection graph G_0. Then can•know•f(**x**, **y**, G_0) is true if and only if there is an admissible rw–path between **x** and **y**.

PROOF: See [BISH79]. ∎

This can be extended to include both *de jure* and *de facto* rules.

DEFINITION D1.11: can•know(**x**, **y**, G_0) is true if and only if there is a sequence of protection graphs $G_1, \ldots, G_n$ such that $G_0 \vdash^* G_n$ and in G_n either a **x**-to-**y** edge labelled r exists, or a **y**-to-**x** edge labelled w exists and, if the edge is explicit, its source is a subject.

This is merely can•know•f($\mathbf{x}$, $\mathbf{y}$, G_0) without the restriction on the types of rules used.

DEFINITION D1.12: An *rwtg–path* is a nonempty sequence $\mathbf{v}_0, \ldots, \mathbf{v}_k$ of distinct vertices such that for all i, $0 \leq i < k$, $\mathbf{v}_i$ is connected to $\mathbf{v}_{i+1}$ by an edge (in either direction) with a label containing a t, g, r, or a w.

With each rwtg–path, associate one or more words over the alphabet $\{ \vec{t}, \overleftarrow{t}, \vec{g}, \overleftarrow{g}, \vec{r}, \overleftarrow{r}, \vec{w}, \overleftarrow{w} \}$ in the obvious way.

DEFINITION D1.13: The vertex v_0 *rw–initially spans* to $\mathbf{v}_k$ if $\mathbf{v}_0$ is a subject and there is an rwtg–path between $\mathbf{v}_0$ and $\mathbf{v}_k$ with associated word in $\{ \vec{t}^* \vec{w} \}$.

DEFINITION D1.14: A vertex $\mathbf{v}_0$ *rw–terminally spans* to $\mathbf{v}_k$ if $\mathbf{v}_0$ is a subject and there is an rwtg–path between $\mathbf{v}_0$ and $\mathbf{v}_k$ with associated word in $\{ \vec{t}^* \vec{r} \}$.

DEFINITION D1.15: A *bridge* is an rwtg–path with associated word in the regular language

$$B = \{ \overleftarrow{t}^* \cup \vec{t}^* \cup \overleftarrow{t}^* \vec{g} \vec{t}^* \cup \vec{t}^* \overleftarrow{g} \overleftarrow{t}^* \}$$

(Note that this is the same as the definition given earlier in this section.) A *connection* is an rwtg–path with associated word in the regular language

$$C = \{ \vec{t}^* \vec{r} \cup \vec{w} \overleftarrow{t}^* \cup \vec{t}^* \vec{r} \vec{w} \overleftarrow{t}^* \}$$

The next result characterizes the set of graphs for which can•know is true:

THEOREM 1.5: can•know($\mathbf{x}$, $\mathbf{y}$, G_0) is true if and only if there exists a sequence of subjects $u_1, \ldots, u_n$ in G_0 ($1 \leq n$) such that the following conditions hold:

C1.6 $\mathbf{x} = \mathbf{u}_1$ or $\mathbf{u}_1$ rw–initially spans to $\mathbf{x}$,

C1.7 $\mathbf{y} = \mathbf{u}_n$ or $\mathbf{u}_n$ rw–terminally spans to $\mathbf{y}$,

C1.8 for all i, $1 \leq i < n$, there is an rwtg–path between $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ with an associated word in $B \cup C$.

PROOF: See [BISH79]. ■

Now that we have seen the basic model, let us consider the two questions which this thesis will attempt to answer.

1.3. Introduction to Hierarchies

A *hierarchy* is "a graded or ranked series" [WEBS73]. When used in the context of security, it means that there are gradations of access to certain information. The most famous example of this is the *classification system* used by the federal government today, where documents are labelled UNCLASSIFIED (the least secret level), CONFIDENTIAL, SECRET, or TOP SECRET (the most secret level), depending on their content. Each person who may need access to these documents is given a *security clearance* indicating the most secret level of documents which he may see; for example, a person with SECRET clearance may read documents labelled UNCLASSIFIED, CONFIDENTIAL, or SECRET, but not documents labelled TOP SECRET.

Such systems, rather obviously, are meant to protect information, and for this reason, people are not to reveal information to those with a lower security clearance. How can this be done, assuming that no-one in the hierarchy can be trusted? The problem was first approached mathematically in [BELL74] in the context of operating systems. Often, computer systems are designed with several levels of security; two examples of this are the THE operating system [DIJK68] and PSOS. The first application of the Take–Grant Protection Model [WU80] was to hierarchical systems in general, not just operating systems, and we shall extend those results as well as explore

some other means of ensuring a secure hierarchy.

The remainder of this thesis will explore the use of the Take–Grant model in a practical setting.

1.4. Take–Grant Modelling of Systems

It has long been believed that the Take–Grant Protection model has little if any practical use; indeed, one criticism of it is that one could not represent hierarchies using that model. Considering the model was introduced to explore theoretical matters, this belief is not surprising.

This requires some explanation. The terms *security* and *safety* are often used interchangeably; in fact, they are not synonyms. The term "safe" applies to an abstract model; its initial state is called "safe" if it is not possible to reach a new state in which a right can be transferred. The term "secure" applies to a nonabstract system; it requires not only that the abstract model of the system be safe, but also that the nonabstract system correctly implement the abstract model. Harrison, Ruzzo, and Ullman showed that in general, it cannot be determined whether or not a system is safe:

THEOREM 1.6: It is undecidable whether a given state of a protection system is safe for a given generic right.

PROOF: See [HARR76]. ■

The Take–Grant model describes a simpler type of system, called a *mono–operational system* (because each command performs a single primitive operation. For such systems,

THEOREM 1.7: there is an algorithm that decides whether a given mono-operational system and initial state is safe for a generic right.

PROOF: See [HARR76]. ■

In fact, safety in the Take–Grant system is not only decidable even if the number of objects which can be created is unbounded, but it is decidable in time linear in the size of the graph.

The first attempt to use the Take–Grant Protection System to model a computer system was presented in [JONE78], where it was used to model a security flaw in Multics. Unfortunately, the modelling was done *ad hoc*; rather than present a definition of security and then test the flaw against it, the flaw was simply modelled. This thesis takes a slightly different approach.

In general, it is hard to use the Take–Grant Protection System to model a computer system because it has no concept of *groups* in it. With most computer systems, one process or one user will perform actions that affect many files and processes, rather than just one. The Take–Grant rules, however, require that one actor operate on one target at a time. It is this point which this thesis considers. First, we extend the model to include the concept of *theft of information*; then we introduce graph rewriting rules which allow actors to act upon more than one target, and which allow one actor to affect many other objects. Finally, we define *security* in terms of the theoretical results and show how these results can be applied to a computing system.

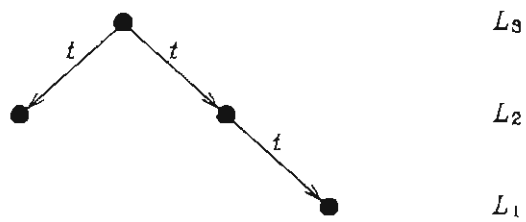
Ideally, one tests models of abstract systems, fixing the theoretical flaws which appear, until the model is safe; then, and only then, is the model implemented. In this thesis, however, we are trying to test existing systems for security flaws; hence, we shall go from an existing system to the abstract model. Hence, the term *secure* will mean not just the safety of the abstract model, but also that the abstraction captures the essential details of the system we are examining for flaws.

Chapter II

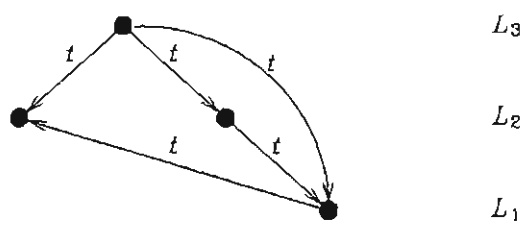
Hierarchical Take–Grant Protection Systems

II.1. Introduction

The problem of secure hierarchies was discussed briefly in section I.3. In previous works using the *Take-Grant Protection Model* to model hierarchies, there has always been an underlying assumption that at least some of the vertices were honest. For example, [WU80] presents a model of hierarchical protection systems involving only take and grant rules; but if two directly-connected subjects conspire to breach the security of the system, they can easily do so by using the two lemmas 1.1 and 1.2. For example, this situation



can be breached by applying the take rule and then lemma 1.1:



Previous models have also discussed at great length the transfer of rights: under what circumstances can it occur assuming the subjects are honest, what rights can a subject steal from a higher-level one, and the like. But little has been said about the transfer of information.

Briefly, in the model of hierarchical protection systems developed here, we are concerned with preventing transfers of information to subjects with a lower security level than the information has, as well as preventing the lower-level subject from obtaining authority to read the information. Because of the duality of reading and writing, subjects must also be prevented from writing to lower-level ones (otherwise, a dishonest high-level subject could just read high-level information and then write it to a lower-level subject, thereby breaching security). This raises some interesting questions. Is is possible to prevent information from being transfered using the graph rewriting rules for transferring authority and information without modification? If so, would restricting these rules provide any additional benefits? What kinds of restrictions should be considered? We shall try to answer these questions, among others.

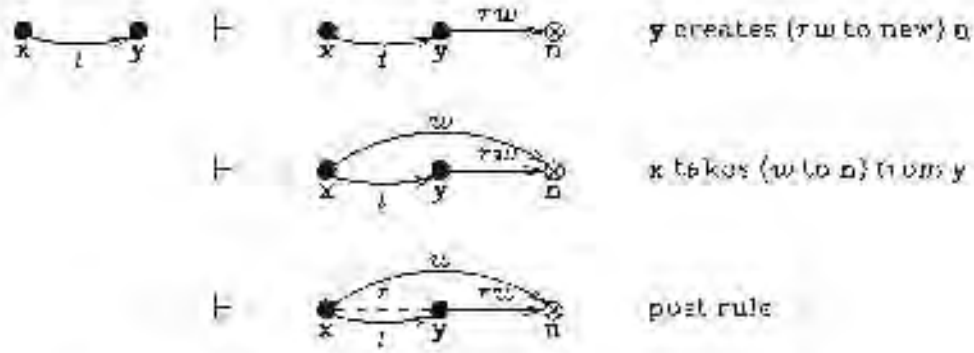
The strength of proposed restrictions will be considered as well. Let T be a set of graphs with some (arbitrary) property, and let R be a set of graph rewriting rules. Then R will be called *sound* if applying any finite sequence of those rewriting rules to a graph $t \in T$ will produce a graph $t' \in T$. Let $s, t \in T$ be any two graphs such that $s \vdash^* t$ using the rules in R; Then, if $s \vdash^* t$ using only elements of a subset $R' \subset R$, the subset R' is said to be *complete*. (Note that *subset* includes restricted rewriting rules as well.) A graph is said to be *secure* if there is no finite sequence of rewriting rules that will enable a subject to obtain information at a higher security level than the subject. The soundness and completeness of restrictions with respect to the property *secure* will be discussed.

We shall use one result in this chapter not presented earlier.

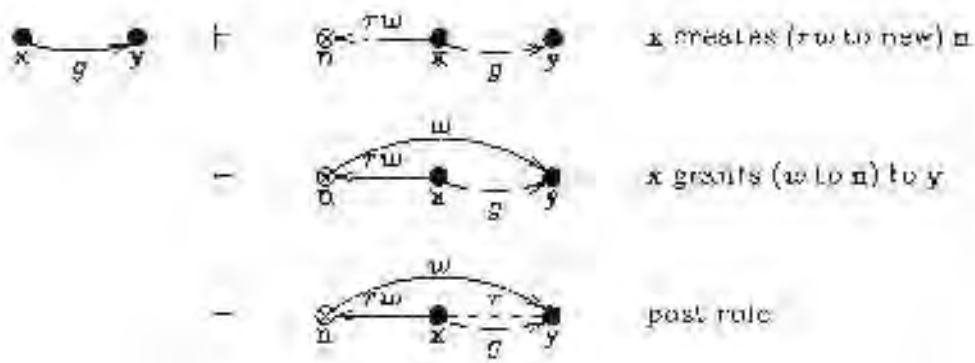
LEMMA 2.1: Let G_0 be a protection graph, and let x and y be vertices in the same island. Then can•know(x, y, G_0) and can•know(y, x, G_0) are both true.

PROOF: There are four cases to be considered:

Case 1:



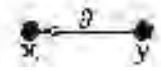
Case 2:



Case 3:

see case 2

Case 4:



see case 1

In all cases, an implicit read edge from **x** to **y**, and one from **y** to **x**, can be obtained. ■

II.2. A De Facto Hierarchical Protection System

DEFINITION D2.1: An *rw–level* is a set of vertices in a protection graph G_0 such that for every pair of vertices **x** and **y** in the set, can•know•f(**x**, **y**, G_0) is true.

Intuitively, vertices **x** and **y** are on the same rw–level if they have *de facto* access to exactly the same information, and can communicate with each other.

PROPOSITION 2.2: Let G_0 be a protection graph, and let **x** and **y** be any two vertices in G_0. Then **x** and **y** are in the same rw–level if and only if there is an rw–admissible path between them.

PROOF: Immediate by theorem 1.4 ■

DEFINITION D2.2: Two subject vertices **x** and **y** are said to be *rw–joined* if can•know•f(**x**, **y**, G_0) is true and can•know•f(**y**, **x**, G_0) is false.

Intuitively, this definition says that if a vertex **x** can obtain any information that another vertex **y** can, but not conversely, then **x** and **y** are rw–joined.

Those two ideas can be combined into one relation. In a protection graph G_0, say that a vertex **x** is *higher* than a second vertex **y** if, for any path from **x** to **y**, there exist vertices **a** and **b** such that can•know•f(**x**, **a**, G_0) and can•know•f(**b**, **y**, G_0) are true, and **a** is rw–joined to **b**. In this case, **y** is said to be *lower* than **x**. (We write this as **x**>**y**.) Two vertices **x** and **y** are said to be *at the same height* (or *level*) if **x** and **y** are at the same rw–level. Analogously, one rw–level L_2 is said to be *higher* than another rw–level L_1 if for every vertex $\mathbf{l}_2$ in L_2 and for every vertex $\mathbf{l}_1$ in L_1, $\mathbf{l}_2 > \mathbf{l}_1$. Again, in this case L_1 is said to be *lower* than L_2.

Informally, an rw–level is a security classification; and the higher an rw–level is, the more highly classified the security level is.

In what follows, unqualified statements about the relation *higher* apply to both the relation between vertices and the relation between rw–levels. When the distinction between the two relations is important, it will be explicitly stated, or be obvious from the context.

LEMMA 2.3: Let L_1 and L_2 be two different rw–levels in a protection graph G_0 such that L_2 is higher than L_1. Let l_1 be a vertex in L_1 and l_2 be a vertex in L_2. Then can•know•f(l_2, l_1, G_0) is true and can•know•f(l_1, l_2, G_0) is false.

PROOF: As L_2 is higher than L_1, there exist vertices **x** in L_2 and **y** in L_1 such that **x** is higher than **y**. By definition, can•know•f(l_2, **x**, G_0) and can•know•f(**y**, l_1, G_0) are both true. By the definition of higher, can•know•f(**x**, **y**, G_0) is true, so two applications of the *spy* rule show that can•know•f(l_2, l_1, G_0) is true. Now, let **x** be any vertex in L_1 and let **y** be any vertex in L_1 such that **x** and **y** are rw–connected. As L_2 is higher than L_1, the path from **y** to **x** is not an rw–admissible path; hence, can•know•f(**y**, **x**, G_0) is false. As any path from l_1 to l_2 must pass through an edge from L_1 to L_2, by proposition 2.2, no such edge is rw–admissible, so there is no rw–admissible path from l_1 to l_2. Hence, by theorem 1.4, can•know•f(l_1, l_2, G_0) is false. ■

Lemma 2.3 says that if one rw–level is higher than another, vertices in the higher level can obtain information from vertices in the lower level. However, no vertex in the lower level can access information which is available only to vertices in the higher level only. In effect, this is a hierarchical classification system of two levels. Generalizing this result to an arbitrary number of levels gives

THEOREM 2.4: Let $L_1, \ldots, L_n$ be rw-levels in a protection graph G_0 such that for $i=2, \ldots, n$, $L_i > L_{i-1}$. Let l_k be a vertex in L_k and l_j be a vertex in L_j, where $j<k$. Then can•know•f(l_j, l_k, G_0) is true and can•know•f(l_k, l_j, G_0) is false.

PROOF: By induction on $k-j$

BASIS: When $k-j=1$, the claim holds by Lemma 2.3.

INDUCTION HYPOTHESIS: For $k-j=1, \ldots, k-1$, the claim holds.

INDUCTION STEP: Let $i-j=k$. By the hypothesis, for any vertex l_{k-1} in L_{k-1}, can•know•f(l_k, l_{k-1}, G_0) is true, and can•know•f(l_{k-1}, l_k, G_0) is false. By lemma 2.3 can•know•f(l_{k-1}, l_j, G_0) is true, whence can•know•f(l_k, l_j, G_0) is true. Also by lemma 2.3, there is no vertex l_k in L_k such that can•know•f(l_j, l_{k-1}, G_0) is true, whence can•know•f(l_j, l_k, G_0) must be false. ■

The arrangement of rw-levels described in theorem 2.4 will be called a *structure*, indeed, theorem 2.4 provides the Take-Grant Protection Model with the structure needed to model a hierarchical classification system. For example, consider the linear classification scheme

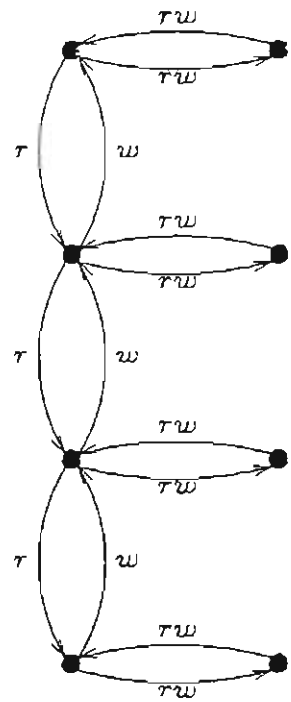
L_4

L_3

L_2

L_1

This classification scheme has 4 levels, namely L_4, L_3, L_2, and L_1 (in descending order). This classification hierarchy is easily modeled by a structure of this kind:



However, the structure provided can be used to model more complicated hierarchical systems such as those the levels of which are *not* linear. Recall that a *partial ordering* is an ordering which is transitive and irreflexive [ENDE77].

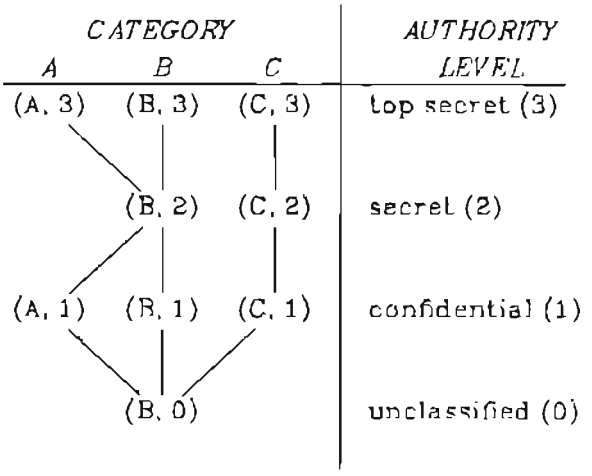
PROPOSITION 2.5: The relation *higher* is a partial ordering.

PROOF: First, consider the relation with respect to vertices. For transitivity, merely apply the *spy* rule; for irreflexivity, merely note that rw–joined vertices are on different rw–levels. The proof for the relation with respect to rw–levels follows immediately by definition. ■

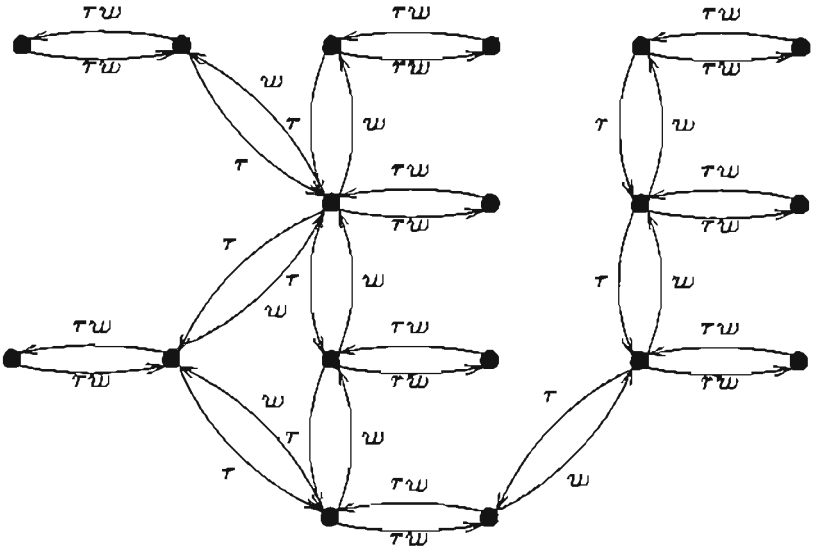
This theorem has two immediate consequences. The first is obvious; if $L_i \leq L_j \leq L_i$, then $L_i = L_j$. This emphasizes the hierarchical nature of the structure described in theorem 2.4. Secondly, at most one of the relations $L_i < L_j$, $L_i = L_j$, and $L_i > L_j$ is true. Note that *none* need be true, which means that there can be rw–levels which are incomparable. Thus, while there is at least one minimal

and one maximal level (with respect to the *highest* relation) in any such structure, there need not be a lowest or a highest level.

This enables a classification system which uses a partial, rather than a linear, ordering to be modelled. One such classification system is the *Military Classification System* (eg., see [DENN76, DENN82]):



Such a configuration is easily translated into this model:



each security level L_i representing an authority level and a category.) In particular, note that while two subjects may have the same security classification, the model makes no assumptions about their being able to communicate with each other.

This representation of hierarchical classification systems has an extremely important effect. Recall that the *de facto* rules *spy*, *post*, and *find* require the cooperation of *two* subjects. Thus, in many cases, if two subjects conspired to breach the security of a hierarchical system, they could do so. Specifically, suppose a subject were willing to violate security. Under other proposed representations, this would mean that all inferior subjects would be suspect, because the superior could conspire with them to breach security. But by theorem 2.4, in this representation such a conspiracy is impossible. Preventing any breaches in security, rather than trusting any subject to be honest, makes this model so useful.

Thus far we have only been concerned about subjects. It is reasonable to assume that if people (subjects) have a certain security classification, some information also has that classification. As documents are inactive, they are to be represented by object vertices; it is therefore necessary to define what the security level of an object vertex is, and show that under such a definition information cannot flow from a higher level to a lower.

DEFINITION D2.3: An object vertex **v** is said to belong to the *lowest* rw–level in which a subject vertex has either read or write access to it.

Intuitively, this says that a document has the same security level as the lowest level of users with access to it. The central result of this section can now be proven:

THEOREM 2.6: Let G_0 be a protection graph, and let $L_1, \ldots, L_n$ be rw–levels such that for $i = 2, \ldots, n$, $L_i > L_{i-1}$. Let l_j be an object vertex in L_j. Then for any subject vertex l_i in L_i, $i < j$, can•know•f(l_i, l_j, G_0) is false.

PROOF: In order that can•know•f(l_i, l_j, G_0) be true, there must be an rw–admissible path from l_i to l_j by theorem 1.4. But as l_j is in a higher rw–level than l_i, there is no such path (as there is no rw–admissible path from $\mathbf{x}$ to any vertex in L_k, where $k < i$). Hence can•know•f(l_i, l_j, G_0) is false. ■

This theorem states that unless a user has a security level equal to or greater than that of a certain document, he cannot obtain access to it regardless of how many (possibly corrupt) people *do* have access to it.

We have seen how to model hierarchical sturctures where security is concerned only with the passing of information. Now, let us consider this in conjunction with the passing of authority.

II.3. Combining De Facto and De Jure Rules

In order to include *de jure* rules in the model of a hierarchical protection graph, some of the definitions of the last section must be generalized.

DEFINITION D2.4: An *rwtg–level* is a set of subjects in a protection graph G_0 such that for every pair of vertices $\mathbf{x}$ and $\mathbf{y}$ in the set, can•know($\mathbf{x}$, $\mathbf{y}$, G_0) and can•know($\mathbf{y}$, $\mathbf{x}$, G_0) are true.

The notion of *higher* discussed in the previous section can also be extended in the obvious way to include rwtg–levels and vertices in those levels. In particular, the relation is still a partial order. A protection graph G_0 is *secure* if for all vertices $\mathbf{x}$ and $\mathbf{y}$ such that $\mathbf{x}$ is lower than $\mathbf{y}$, can•know($\mathbf{x}$, $\mathbf{y}$, G_0) is false.

These definitions require that information cannot flow from a higher to a lower level. The notion of an rwtg-level merely combines the concept of an island with that of an rw-level, and coincides with the notion of a security classification level. More formally:

LEMMA 2.13: Every island is contained in exactly one rwtg-level.

PROOF: By lemma 2.1 and the definition of an rwtg-level, an island is itself an rwtg-level; hence, an island is contained in *at least* one rwtg-level. So, assume that an island has vertices l_1 and l_2 in distinct rwtg-levels L_1 and L_2, respectively. By definition of island, can•know(l_1, l_2, G_0) is true; hence, by transitivity of can•know and the definition of rwtg-level, l_1 and l_2 are in the same rwtg-level, contradicting the assumption that L_1 and L_2 are distinct. Hence, each island is contained in *at most* one rwtg-level. This proves the theorem. ■

In what follows, the definition of rw-joined is exactly the same as in II.2.

THEOREM 2.7: A protection graph G_0 is secure if and only if there are no bridges or connections between rwtg-levels.

PROOF: Let l_i and l_j be two vertices in two distinct rwtg-levels L_i and L_j (respectively) such that $L_i > L_j$. We must show that can•know(l_j, l_i, G_0) is false if and only if there are no bridges or connections between L_i and L_j.

($\Leftarrow$) Assume can•know(l_j, l_i, G_0) is false. Then apply theorem 1.5. As C1.6 and C1.7 are trivially true for any rwtg-level, C1.8 must be false (or else can•know(l_j, l_i, G_0) would be true, contradicting assumption). The result follows immediately from the definition of island and theorem 2.13.

($\Rightarrow$) Assume now there are no bridges or connections between L_i and L_j. It must be shown that can•know(l_j, l_i, G_0) is false. Note first that by hypothesis there is no connection between l_j and l_i. So consider condition C1.8 of Theorem 1.5. As there are no bridges between rwtg-levels, there are no bridges between

islands; hence condition C1.3 is false, so can•know$(I_y, I_{\{\}}, G_\beta)$ is false.

This proves the theorem. ■

Notice that this theorem agrees quite nicely with theorem 1.5. The nature of an rwtg–level ensures that vertices are rw–initially and rw–terminally connected to all other vertices in the rwtg–level. The remaining conditions in theorem 1.5 are that no bridges or connections exist between the two vertices, and hence the two rwtg–levels, involved. So this result makes sense.

This result states that the weakest way to ensure that information flows only from lower levels to higher ones in G_0 is to prevent the transfer of any right from an rwtg–level to a lower rwtg–level. This is probably overly restrictive; it may be necessary to allow some rights (but not read or write) to be transferred from one rwtg–level to another, without regard to which rwtg–level is the higher. So let us consider the kinds of restrictions possible.

There are three basic types of restrictions possible:

R2.1 restrict the directions of the take and grant edges between rwtg–levels,

R2.2. restrict the application of the take and grant edges between rwtg–levels, or

R2.3. combine the above two restrictions.

Consider these restrictions one at a time. In the following definitions, let T be a set of graphs with some (arbitrary) property, and let R be a set of graph rewriting rules.

DEFINITION D2.6: R is *sound* if applying any finite sequence of elements of R to some $t \subset T$ produces a graph $t' \subset T$.

DEFINITION D2.7: Let $s, t \in T$, and $s \vdash^* t$ using the rules in R. Then if for some $R' \subseteq R$, $s \vdash^* t$ using only elements of R', the subset R' is *complete*. (Note that *subset* includes restricted rewriting rules as well.)

Restrictions of Direction

Restrictions of direction are those restrictions which require the take or grant edge being used to manipulate rights to point in a certain direction. For example, if the take rule could only be applied to rights which a vertex has over a (lower-level) vertex, that restriction would be a restriction of direction.

Restrictions of direction are sound but not complete. To show soundness, merely note that by theorem 1.5, as there are no bridges or connections in the original protection graph between rwtg-levels one of which is higher than the other, it will be impossible to pass rights of *any* kind between the two levels, so restricting the directions in which a vertex can apply the *de jure* rules will not matter. Thus, as the original graph is secure, so will any other graph derived by these rules. However, the rules restricted in this way are not complete, because rights other than read and write cannot be passed from one rwtg-level to a lower level.

The above argument can be made more rigorous, and shows:

LEMMA 2.8: Restrictions of direction are sound but not complete.

Restrictions of Application

Restrictions of application are those restrictions which prevent the take or grant edge from manipulating certain rights. For example, if the take rule could only be applied to read rights, that restriction would be a restriction of application.

Restrictions of application, like restrictions of directions, are sound but but not complete. To show soundness, merely note that there are no bridges or connections in the original protection graph between rwtg–levels one of which is higher than the other. Thus, by theorem 1.5, it is impossible to pass rights of *any* kind between the two levels, and so limiting the kinds of rights which can be passed has no effect. Thus, as the original graph is secure, so will any other graph derived by these rules. However, the rules restricted in this way are not complete. For example, if the take rule is restricted so that it cannot act on read rights, this will prevent a higher–level vertex from taking read rights to a lower–level vertex.

The above argument can be made more rigorous, and shows:

LEMMA 2.9: Restrictions of application are sound but not complete.

Restrictions of Direction and Application

Both restrictions of application and restrictions of direction have one severe shortcoming: under them, the *de jure* rules are not complete. Hence, consider a combination of the first two restrictions. The object is to modify the

take and grant rules so that no explicit or implicit read edges go from a higher to a lower rwtg–level after a finite number of *de jure* and *de facto* rule applications. We propose the following restriction, in which a directed path begins at a *source* vertex and ends at a *target* vertex:

R2.4. No *de jure* rule may be applied if, as a result, either of the following connections would be completed:

a. $\vec{r}$ with the source vertex lower than the terminal vertex, or

b. $\vec{w}$ with the source vertex higher than the terminal vertex.

In more formal language, let R be a *de jure* rule, and let $G \vdash_R G'$. This is an invalid step in a derivation if there exist vertices $\mathbf{x}, \mathbf{y} \in G$ such that $\mathbf{x} > \mathbf{y}$, no edge from $\mathbf{x}$ to $\mathbf{y}$ has associated word $\vec{w}$ and no edge from $\mathbf{y}$ to $\mathbf{x}$ has associated word $\vec{r}$.

There is an intuitive basis for these restrictions. First, notice that the reason bridges are included in theorem 2.13 is that rights are transmitted over bridges, and so in the sequence of rule applications to move the right along the bridge, one of the three (forbidden) connections must occur. (A read edge from one vertex to a higher one violates restriction R2.4a, and a write edge from one vertex to a lower one violates restriction R2.4b. For the third possible connection, the object which is the target of the write edge belongs to the lower level, so it reduces to restriction R2.4a.) Thus, not the construction of bridges, but only the transmission of read or write rights along them, need be restricted.

In addition, there is no restriction on *any* rights other than read and write. In particular, other rights can be freely passed from one level to another. This is an advantage of these restrictions, and enables us to show the main result:

THEOREM 2.10: The *de jure* and *de facto* rules, under the restriction stated above, are both sound and complete.

PROOF: SOUNDNESS: This follows immediately from theorem 2.7, the nature of the restriction, and the fact that if read or write rights are passed along bridges, there will be a connection before the right is acquired by the target vertex.

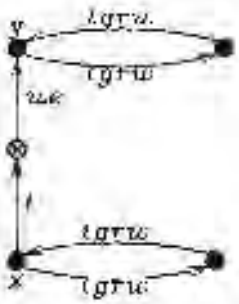
COMPLETENESS: Let $G_0, \ldots, G_n$ be a sequence of protection graphs such that G_0 and G_n are secure (but $G_1, \ldots, G_{n-1}$ need not be secure). Let $\rho_0, \ldots, \rho_{n-1}$ be a set of unrestricted *de jure* and *de facto* rules such that $G_0 \vdash_{\rho_0} G_1 \vdash_{\rho_1} \ldots G_{n-1} \vdash_{\rho_{n-1}} G_n$. Then we must find a sequence $\rho_1', \ldots, \rho_k'$ of rewriting rules such that $G_0 \vdash_{\rho_0'} G_1' \vdash_{\rho_1'} \ldots \ldots G_{k-1}' \vdash_{\rho_{k-1}'} G_k'$ and $G_k' = G_n$.

As G_0 and G_n are secure, no connections exist in either. If none of the restricted connections were made by any of $\rho_0, \ldots, \rho_{n-1}$, then we are done (just take this sequence to be the sequence satisfying the restrictions.) So suppose a connection was made by some rule application, say, ρ_{j_0}. Only an implicit read edge could have been added via this connection. Let $\rho_{j_1}, \ldots, \rho_{j_m}$ be the set of rules that manipulate this implicit edge. Clearly, none of these rules can change any explicit edges in the graph (by inspection of the *de facto* rules); and when the connection made by ρ_{j_0} is broken (as it must be, for there are no connections in G_n), all implicit edges made by the rules $\rho_{j_1}, \ldots, \rho_{j_m}$ will be deleted from the protection graph. Hence, the existence of the connection in the derivation has no effect on the final graph, because any edges added as a result of its existence will be deleted by the end of the derivation. Thus, we may delete $\rho_{j_0}, \ldots, \rho_{j_m}$ from the derivation. Repeat these deletions for every restricted connection formed during the derivation; as no restricted connection is

formed by the remaining sequence of rule applications, it is the one sought. ■

Note that a greater number of secure graphs may be formed by using the restricted rules than by using the unrestricted ones, because without the restriction the only way to prevent connections is to ban all bridges as well.

An example will emphasize this result. Consider the following protection graph:



In this graph, r stands for the right to *execute* a file. Under the unrestricted *de jure* and *de facto* rules, G_6 is not secure, because by using the take rule, **y** can obtain a write edge to **x**, and hence can pass information to another vertex in a lower rwtg-level. But under the restricted *de jure* and *de facto* rules, **y** cannot obtain a write edge to **x**, because completing the required connection would violate restriction R2.4b. Yet, notice that **y** can still obtain the right to execute **x**; that is not constrained by the restrictions.

The next question is the complexity of the result. How hard is it to test whether or not a given graph meets the restriction, and how hard is it to test whether or not application of a given rule violates the restriction? Clearly, to test for a violation, we need only look at each edge labelled r and each edge labelled w. Thus:

COROLLARY 2.11: Testing a graph for violation of the restriction may be done in time linear in the number of edges of the graph.

PROOF: Immediate from theorem 2.10. ■

COROLLARY 2.12: Determining whether or not an application of a *de jure* rule violates the restriction may be done in constant time.

PROOF: Immediate from theorem 2.10. ■

Chapter III

More New Results and Proofs in the Original Model

III.1. Introduction

In chapter II, we saw how hierarchies could be modelled using the Take–Grant Protection System. We shall now begin to consider how to model a computer system.

We wish to capture two types of theft in our model: thefts of rights and thefts of information. The former we can describe exactly, by the predicate can•steal; but the latter we cannot yet describe. This chapter proposes a new predicate, called *can•snoop*, which will describe thefts of information just as the predicate can•steal describes thefts of rights.

III.2 A New Proof of Can•Know

Before we do so, however, we will give an alternate proof of the necessary and sufficient conditions for can•know to be true. The proof given in [BISH79] is very complex and does not generalize readily to the other cases we shall touch upon in the next chapter. Because many of the results and proofs in the extensions to the model are quite similar to those of the original model, we wish to emphasize this parallel development between

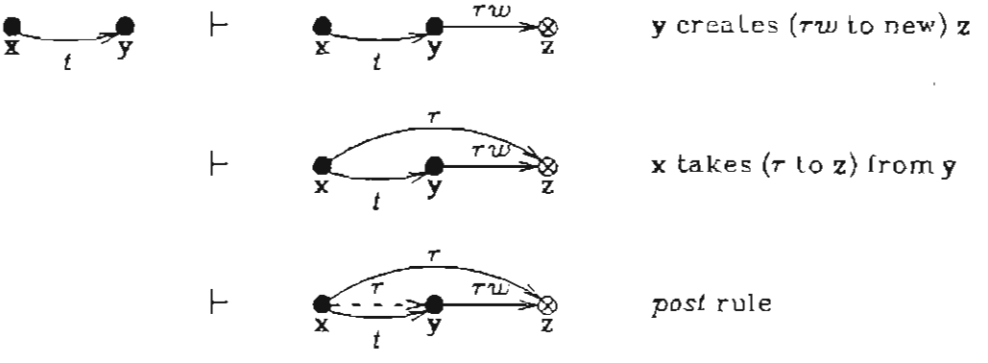
the original model and each of the extensions.

In the interests of clarity, we need the following two lemmas:

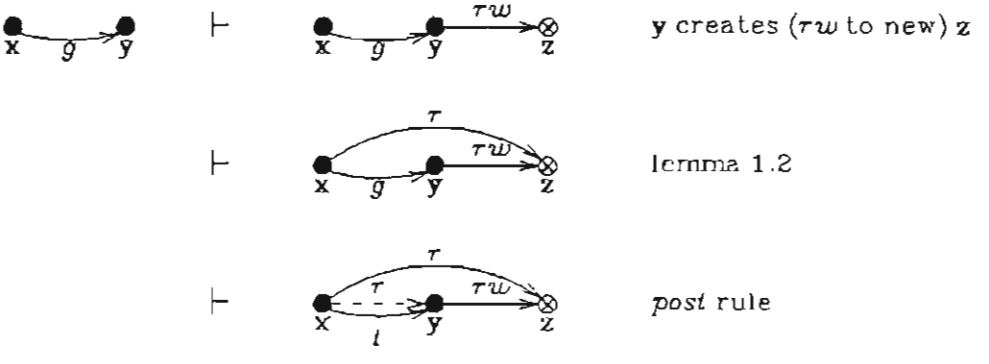
LEMMA 3.1: If there is a bridge from **x** to **y**, then **x** can obtain an implicit read edge to **y**.

PROOF: By the take rule and the definition of *bridge*, it suffices to prove the lemma for bridges of length 2 or less. Six cases arise.

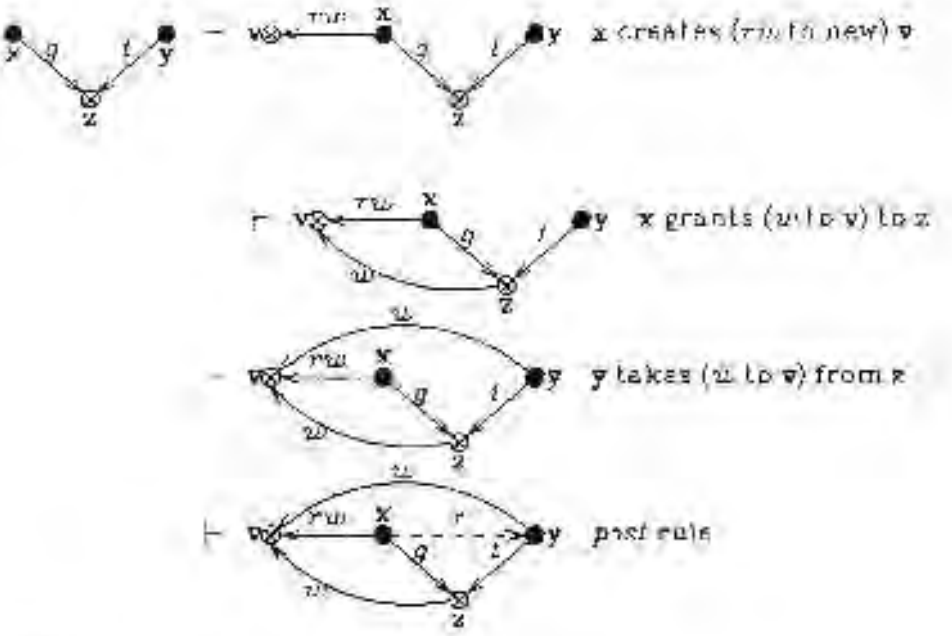
Case 1:



Case 2:

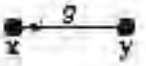


Case 3:



Case 4:

Case 5:



Case 6:

In all cases, x gets implicit read rights over y, proving the claim. ■

LEMMA 3.2: Let x and y be subjects with a bridge or connection between them. Then at least one of the following is true:

C3.1. an explicit read edge from **x** to **y** exists or may be added;

C3.2. an implicit read edge from **x** to **y** may be added; or

C3.3. an explicit write edge from **y** to **x** exists or may be added.

PROOF: If there is a bridge between **x** and **y**, case C3.1 holds by Lemma 3.1. So, suppose there is a connection from **x** to **y**. If the associated word is in $\vec{t}^*\vec{r}$, by using the take rule, **x** can obtain an explicit read edge to **y**, establishing case C3.1. If the associated word is in $\vec{w}\vec{t}^*$, by using the take rule, **y** can obtain an explicit write edge to **x**, establishing case C3.3. If the associated word is in $\vec{t}^*\vec{r}\vec{w}\vec{t}^*$, **x** and **y** can each apply the take rule until **x** obtains a read edge to a vertex to which **y** has a write edge (or vice versa); then **x** and **y** can use the post rule to add an implicit read edge from **x** to **y** (case C3.2.) ■

The last lemma is the most important, because it states a set of conditions which must be met before can•know is true. Hence, the theorem given next really only tries to establish when there will be a series of bridges or connections from a vertex **q** to another vertex **p** along which information can be sent. (Note condition C3.5, which was implied but not explicitly stated, in the earlier version of the theorem.)

THEOREM 3.3: Let **p** and **q** be vertices in a protection graph G_0. Then *can•know*(**p**, **q**, G_0) is true if, and only if, one of the following conditions holds:

C3.4. can•share(r, **p**, **q**, G_0) is true;

C3.5. can•share(w, **q**, **p**, G_0) is true;

C3.6. all of the following conditions hold:

a. There is a subject **p'** such that **p'** = **p** or **p'** is rw-initially connected to p;

b. There is a subject $\mathbf{q}'$ such that $\mathbf{q}' = \mathbf{q}$ or $\mathbf{q}'$ is rw-terminally connected to q;

c. There is a sequence of islands $\{l_j \mid 1 \leq j \leq m\}$ such that there is a bridge or connection from l_j to l_{j+1}, $1 \leq j \leq m$, and $\mathbf{p}' \in l_1$, and $\mathbf{q}' \in l_m$.

INFORMAL ARGUMENT: To prove the "if" part, note that conditions C3.4 and C3.5 imply by definition that can•know is true. For condition C3.6, part C3.6a says that $\mathbf{p}'$ can send any information it gets to $\mathbf{p}$, and part C3.6b says that $\mathbf{q}'$ can obtain any information it needs from $\mathbf{q}$. Part C3.6c simply says that $\mathbf{q}'$ can send the information to a vertex $\mathbf{z}_1$ in l_{m-1}, which can in turn send it to a vertex $\mathbf{z}_2$ in l_{m-2}, and so on, until a vertex $\mathbf{z}_{m-1}$ in l_1 gets the information. By the properties of an island, this means that $\mathbf{p}'$ can get the information. Putting all this together, can•know is true.

Going the other way involves considering the rule applications needed to produce a witness. We can require all *de facto* rules to be applied last, and examine the conditions needed for them to be applied. For example, in the pass rule, there is a vertex $\mathbf{y}$ for which can•share(w, $\mathbf{y}$, $\mathbf{p}$, G_n) and can•share(r, $\mathbf{y}$, $\mathbf{p}$, G_n) are true. From the conditions required for both can•share rules to hold, condition C3.6 of the theorem must be true. (In the formal proof, we will show this for the post and spy rules; the formal proof for the pass and find rules are left as an exercise for the reader.) And if no *de facto* rules are used to generate a witness to can•know, obviously one of conditions C3.4 or C3.5 must hold.

PROOF: ($\Leftarrow$) By definition of can•know, if either condition C3.4 or condition C3.5 holds, can•know is true. So assume neither condition C3.4 nor condition C3.5 holds.

Consider condition C3.6a. As $\mathbf{p}'$ is *rw*-initially connected to $\mathbf{p}$, it either has or can obtain (through the take rule) a write edge to $\mathbf{p}$. Similarly, by condition C3.6a, if $\mathbf{q}'$ is *rw*-terminally connected to $\mathbf{q}$, it either has or can obtain (using the take rule) a read edge to $\mathbf{q}$. Thus, it suffices to show can•know($\mathbf{p}'$, $\mathbf{q}'$, G_0) is true by condition C3.6c; merely apply the spy rule (if $\mathbf{q}' \neq \mathbf{q}$), and then the post rule (if $\mathbf{p}' \neq \mathbf{p}$), to obtain can•know($\mathbf{p}$, $\mathbf{q}$, G_0).

We show condition C3.6c implies can•know($\mathbf{p}'$, $\mathbf{q}'$, G_0) by inducting on m, the number of islands in that condition.

BASIS: Let $m = 1$. Then $\mathbf{p}'$ and $\mathbf{q}'$ are in the same island, whence by lemma 2.1, can•know($\mathbf{p}'$, $\mathbf{q}'$, G_0) is true.

INDUCTION HYPOTHESIS: For $m = 1, \ldots, k$, if condition C3.6c holds, can•know($\mathbf{p}'$, $\mathbf{q}'$, G_0) also holds.

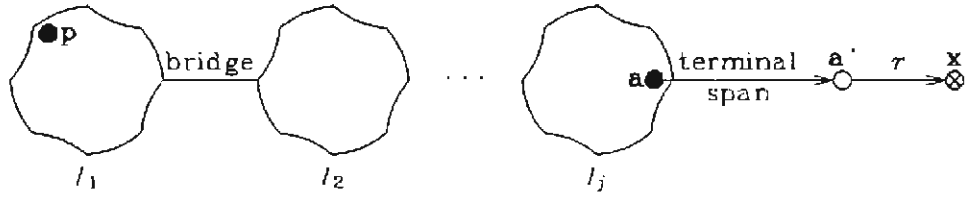
INDUCTION STEP: Let $m = k + 1$. Let $\mathbf{z}_k$ be the subject in I_k that bounds the bridge or connection between I_k and I_{k+1}; let $\mathbf{z}_{k+1}$ be $\mathbf{z}_k$'s counterpart in I_{k+1}. By lemma 3.2, can•know($\mathbf{z}_{k+1}$, $\mathbf{q}'$, G_0) holds; by lemma 3.1, this means can•know($\mathbf{z}_k$, $\mathbf{q}'$, G_0) holds; and by the induction hypothesis, can•know($\mathbf{p}'$, $\mathbf{z}_k$, G_0) holds; whence by the spy rule, can•know($\mathbf{p}'$, $\mathbf{q}'$, G_0) is true. This proves the induction hypothesis, and hence the "if" part of the theorem.

($\Rightarrow$) Now assume can•know($\mathbf{p}$, $\mathbf{q}$, G_0) is true, and consider a minimal set of rule applications ρ_i needed to produce a witness. Without loss of generality, we may reorder the ρ_i's so that all *de jure* rule applications precede any *de facto* rule applications, since *de facto* rule applications do not change the state of the protection graph. If no *de facto* rules are applied, the witness will end with either an explicit read edge from $\mathbf{p}$ to $\mathbf{q}$, in which case condition C3.4 holds, or an explicit write edge from $\mathbf{q}$ to $\mathbf{p}$, in which case condition C3.5 holds. So suppose that at least one *de facto* rule application is needed.

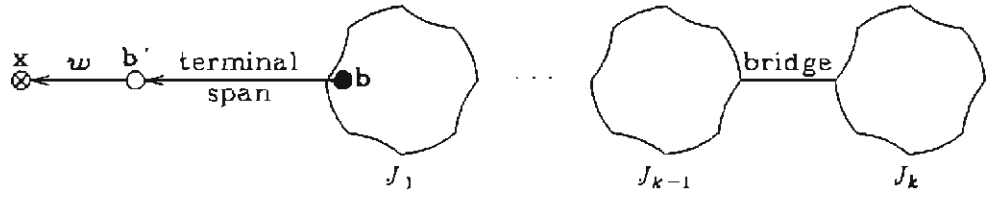
Induct on the number m of such *de facto* rule applications.

BASIS: Let $m = 1$. Each of the *de facto* rules must be considered. We will give the proof for the post rule; the other rules are treated similarly.

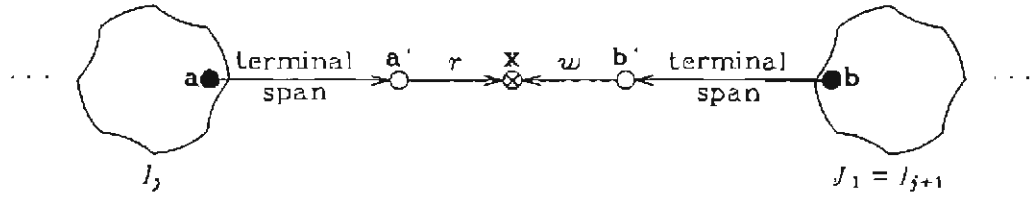
Consider the post rule. In order to apply this rule, there must be a vertex **x** such that can•share(r, **p**, **x**, G_0) and can•share(w, **q**, **y**, G_0) are true. By theorem 1.3, this means that there is a sequence of islands $I_1, \ldots, I_j$ with **p** $\in I_1$, and a vertex **a** $\in I_j$ which terminally spans to another vertex **a'**, which has a read edge to **x**:



Similarly, there is a sequence of islands $J_1, \ldots, J_k$, with **q** $\in J_k$, and a vertex **b** $\in J_1$ which terminally spans to another vertex **b'**, which has a write edge to **x**:



Now, combining these two facts, relabel the islands $J_1, \ldots, J_k$ as $I_{j+1}, \ldots, I_{j+k}$. Note that, as a terminal span has associated word in $\vec{t}^*$, we have a connection $\vec{t}^* r \vec{w} \overleftarrow{t}^*$ from **a** to **b**:

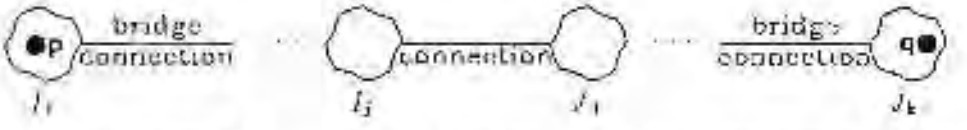


This is condition C3.6c. Taking $p' = p$ and $q' = q$, conditions C3.6a and C3.6b hold.

INDUCTION HYPOTHESIS: Let $m = 1, \ldots, k$. Then after n *de jacto* rule applications, if can•know(**p**, **q**, G_0) is true, conditions C3.6a, C3.6b, and C3.6c hold.

INDUCTION STEP: Let $n = k + 1$, and assume the $k+1$st rule applied is a spy rule (proofs for the other three rules are similar.)

As the spy rule is used, **p** is a subject, and there is a subject vertex **x** such that can•know(**p**, **x**, G_0) and can•know(**x**, **q**, G_0) are true. By the induction hypothesis, the first can•know ensures that there is a subject p' such that $p' = p$ or p' rw-initially spans to **p**, giving condition C3.6a; the second can•know ensures that there is a subject q' such that $q' = q$ or q' rw-terminally spans to **q**, giving condition C3.6b. By the induction hypothesis, condition C3.6c is assumed to hold for both can•know(**p**, **x**, G_0) and can•know(**x**, **q**, G_0). So, let $I_1, \ldots, I_j$ and $J_1, \ldots, J_k$ be the sets of islands for can•know(**p**, **x**, G_0) and can•know(**x**, **q**, G_0), respectively. Thus, the configuration is:



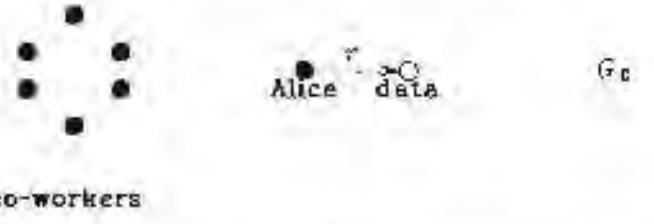
Again, relabel $J_1, \ldots, J_k$ to be $I_{j+1}, \ldots, I_{j+k}$; also, recall that an rw-terminal span between subjects is a connection. This establishes condition C3.6c, proving the induction hypothesis and the claim.

Hence, theorem 3.3 has been proven. ■

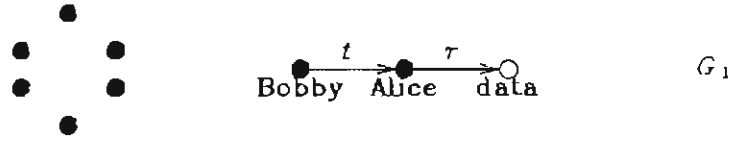
III.3. Definition and Proof of Can·Snoop

Up to this point, we have been considering cases where all vertices cooperate in sharing information, so all *de facto* rules may be applied with impunity. Suppose this is not true: suppose all vertices which have the right to read a vertex flatly refuse to pass the information along. Under what conditions can a vertex which does not have read rights over a second vertex obtain information from the second vertex?

An example will help show what the problem is. Suppose Alice works for a firm which has proprietary information that its competitors need desperately to see. Alice, who works with this information quite a bit, has the authority to read the documents containing the proprietary information whenever she likes, with the understanding she is not to pass this sensitive data to anyone else, including co-workers. The situation, in Take-Grant terms, is:

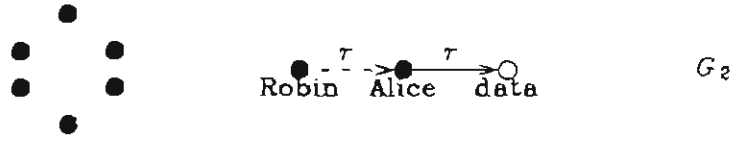


Any documents as sensitive as those which Alice consults must be kept under lock and key. Alice's company has a large vault, which is opened by a key that Alice has. One of her co-workers, Bobby, is not cleared to read these documents and does not have a key to the vault. While passing Alice's desk, he notes a key lying on top of it. Were Bobby to take that key, he would be "taking" Alice's right to read the documents, because she could no longer open the vault. In effect, he would have illicitly obtained the right to read those documents. He could also pass this information on to someone else. This is an example of Alice's sharing (albeit unknowingly) her right to read the documents.



co-workers

Because he is honest, Bobby does not take the key, but merely suggests to Alice that she be a bit more careful. (He is taken aback when she tells him it is her car key and not the key to the vault!) Later in the day, Alice takes a sensitive document out of the vault, goes back to her desk, and begins to read the document. Unfortunately, Robin, who sits directly behind Alice in her office, can see what Alice is reading just by looking over Alice's shoulder. In Take Grant terms, this situation is:



co-workers

By the spy rule, Robin can read anything Alice can (the Robin-to-Alice edge, being unauthorized, is implicit); hence, can•know(Robin, "proprietary data", G_2) is true as long as Robin can look over Alice's shoulder; if Alice read the document elsewhere, such as in the vault, Robin would no longer be able to read the document over Alice's shoulder, so the spy rule would not be applicable since there would be no (implicit or explicit) Robin-to-Alice edge. Notice the difference between can•know(Robin, "proprietary data", G_2) and can•know(Bobby, "proprietary data", G_1); in the latter case, the can•know is true whether or not Alice cooperates by (knowingly or unknowingly) allowing her shoulder to be looked over. The can•know predicate fails to capture this distinction.

We define a new predicate, called can•snoop. This predicate will be true if can•know is true and no-one who has any rights over the information being snooped for cooperates with the snooper. For example, can•snoop(Robin, "proprietary data", G_2) is false, since Alice has to pass the information to Robin (by letting Robin look over her shoulder, in this example), whereas can•snoop(Bobby, "proprietary data", G_1) is true, since Bobby could see the documents whether or not Alice cooperated, once Bobby had "taken" them.

More formally, we define:

DEFINITION D3.1: The predicate *can•snoop*(**p**, **q**, G_0) is true if, and only if, one of the following holds:

C3.7. can•steal(r, **p**, **q**, G_0) is true; or

C3.8. there exists a sequence of graphs and rule applications $G_0 \vdash_{\rho_1} \cdots \vdash_{\rho_n} G_n$ for which all of the following conditions hold:

a. there is no explicit edge from **p** to **q** labelled r in G_0;

b. there is an implicit edge from **p** to **q** labelled r in G_n;

c. neither **q** nor any vertex directly connected to **q** is an actor in a grant rule or a *de facto* rule resulting in a read edge with **q** as its target.

Before we state necessary and sufficient conditions for can•snoop to be true, let us examine the definition more closely. The predicate is rather clearly the *de facto* analogue of can•steal, just as can•know is the *de facto* analogue of can•share. If **p** can steal read rights to **q**, clearly no-one who owns those rights over **q** can prevent **p** from obtaining information from **q**. Similarly, if **p** has authority to read **q**, it would strain the meaning of what we are trying to define to say can•snoop(**p**, **q**, G_0) is true. In G_n, note that any read edge from **p** to **q** must be implicit, for if not, can•steal(r, **p**, **q**, G_0) would have been true, meeting

condition C3.7. And for the purposes of this discussion, we will assume that **q** will not cooperate (either wittingly or unwittingly) with any snooping; it would be equally reasonable to assume that **q** would cooperate, in which case what follows must be modified somewhat.

THEOREM 3.4: For distinct vertices **p** and **q** in a protection graph G_0 with explicit edges only, *can•snoop*(**p**, **q**, G_0) is true if and only if one of the following conditions holds:

C3.9. can•steal(r, **p**, **q**, G_0) is true; or,

C3.10. all of the following hold simultaneously:

a. there is no edge labelled r from **p** to **q** in G_0;

b. there is a subject vertex **p'** such that **p'** = **p** or **p'** $r\overrightarrow{w}$-initially spans to **p**;

c. if **q** is a subject, there is a vertex **q'** such that **q'** ≠ **q**, there is no edge labelled r from **q'** to **q** in G_0, and **q'** $r\overrightarrow{w}$-terminally spans to **q**; and

d. can•know(**p'**, **q'**, G_0) is true.

INFORMAL ARGUMENT: If can•snoop is true, and can•steal false, we have to show all parts of condition C3.10 are true. Condition C3.10a follows from the definition. By part C3.8b of the definition, can•know(**p**, **q**, G_0) is true, from which condition C3.10b springs. Also, by theorem 3.3, condition C3.6b, we have **q'**. Combining this with the definition, it becomes clear that although **q'** $r\overrightarrow{w}$-terminally spans to **q**, **q'** ≠ **q**, and there is no edge labelled r from **q'** to **q** in G_0. The proof that can•know(**p**, **q**, G_0) is true involves proving that the first rule to add a read edge with target **q** is a take rule, and working backwards.

Going from the conditions to can•snoop is trivial.

PROOF: ($\Rightarrow$) Let can•snoop(p, q, G_Γ) be true. If can•steal(r, p, q, G_0) holds, we are done, since part C3.7 of the definition is condition C3.9 of the theorem. So, assume can•steal(r, p, q, G_0) is false.

Part C3.8a of the definition gives condition C3.10a of the theorem.

By part C3.8b of the definition, there is an implicit read edge from p to q in G_n, whence by definition can•know(p, q, G_Γ) is true; so, condition C3.10b of this theorem results from condition C3.8a of theorem 3.3.

By condition C3.8b of theorem 3.3, there is a subject q' such that q' $\neq$ q or q' rw-terminally spans to q. If q is an object, we can take q' to be the q' in condition C3.10c of this theorem. If q is a subject, by part C3.8c of definition D3.1, it is not used in the sequence of rule applications witnessing can•snoop. Hence, in this case, q' $\neq$ q; choose q' in condition C3.10c of the theorem to be this q'. Thus, in either case, the q' in condition C3.10c of this theorem is the same as the q' we in condition C3.8b of theorem 3.3.

Assume q' and q are directly connected by an edge labelled r in G_0. Either can•share(t, p', q', G_0) is true [which means can•steal(r, p, q, G_0) is true, contradiction] or q' must actively participate in a grant, pass, or spy rule application [contradicting part C3.8c of the definition of can•steal.] In either case, there cannot be an edge labelled r from q' to q in G_Γ.

It remains to be shown that can•know(p', q', G_0) is true. Let $G_0 \vdash_{\rho_1} \cdots \vdash_{\rho_n} G_\Gamma$ be a minimum length derivation sequence, and let i be the least index such that $G_{i-1} \vdash_{\rho_i} G_i$, there is no explicit or implicit read edge from x to q in G_{i-1}, and there is an explicit or implicit read edge from x to q in G_{i-1}, where x is any vertex in G_{i-1}. That is, G_i is the first graph in which an edge labelled r with target q is added. Consider the rule ρ_i which caused this edge to be added.

ρ_i cannot be a grant rule since, by part C3.8c of definition D3.1, the owner of r rights to q will not will not grant them to anyone else. ρ_i cannot be a pass, spy, or find rule, since by part C3.8c of definition D3.1, the owner of r rights to q will not pass information from q to anyone else. ρ_i cannot be a post rule since by part C3.8c of definition D3.1, q will not pass information from itself to anyone else. As neither the create nor the remove rules add edges to existing vertices, ρ_i cannot be either. Hence, ρ_i must be a take rule.

We therefore have:

Recalling that can•know(p, q, G_0) is true, by theorem 3.3 we see that can•know(p', q, G_0) is true. Apply theorem 3.3 again. By this theorem, there is a subject q' such that $q' \neq q$ or q' rw-terminally spans to q. Noting that there is no direct edge labelled r from q' to q in G_0, we take $q' = x$ in theorem 3.3 and in this theorem, whence can•know(p', q', G_0) immediately follows.

($\Leftarrow$) If condition C3.9 holds, by part C3.7 of definition D3.1, can•snoop(p, q, G_0) is true.

So, assume condition C3.10 holds. Part C3.8a of definition D3.1 is the same as condition C3.10a of the theorem. By theorem 3.3, conditions C3.10b, C3.10c, and C3.10d of theorem 3.4 establish part C3.8b of definition D3.1.. And as $q' \neq q$ when q is a subject, part C3.8c of definition D3.1 is also true.

This completes the proof of theorem 3.4 ■

Chapter IV

Some Extensions to the Take–Grant Model

IV.1. Introduction

In its original form, the Take Grant model is a powerful theoretical tool; unfortunately, there is one major drawback in applying it to non-theoretical situations. The model deals with authorities between individual vertices, and in practise, this is rarely done. Most computer systems have the concept of a "protection group" designed into them, so that many users can share access to a set of files. To apply the Take Grant model to this situation would require that the graph rewriting rules be changed so that various combinations of takes and grants were always performed in a given sequence.

In this chapter, we will look at three ways in which the rules may be rewritten, and how doing so affects the theorems and proofs. The rules we shall introduce involve acting on *classes* or sets of vertices as well as individual vertices; in the next chapter, we shall apply many of our results to an operating system to show how, in practise, one would go about using the theory developed in this chapter.

DEFINITION D4.1: A *class* is a collection of vertices.

In what follows, all vertices belong to at least one class. More than one vertex may belong to the same class; similarly, one vertex may belong to many classes. Should there be a one to one correspondence between classes and

vertices, that is, if each vertex belongs to exactly one class and no class has more than one vertex, the situation reduces to that of the original model.

A word on notation is necessary. We shall examine three possible extensions: when one vertex acts on a class, when a class of vertices acts on one vertex, and when a class of vertices acts on another class. For brevity, we shall refer to these situations as the "one-many", the "many-one", and the "many-many" extensions to the Take Grant model, respectively. (For consistency, the model discussed in the preceding chapters ought to be called the "one-one" version, but because this term has not been used before, we will refer to that model as the "original" model.) The same predicate names as for the unextended model will be used, but we will indicate which graph rewriting rules are involved by placing one of the subscripts *om*, *mo*, or *mm* after the predicate to indicate that the rules used are those of the one-many, many-one, or many-many extension to the Take Grant model.

One point about the rules ought to be emphasized. We will change the graph rewriting rules (the *de jure* rules) only. The *de facto* rules do not add explicit edges, and so are not, strictly speaking, graph rewriting rules. More importantly, they are rules which an individual, rather than a class, can apply; information flows from one vertex to another, and not from one class of vertices to another. Hence, throughout this chapter, the *de facto* rules used are those used in the original model.

One word of warning is appropriate here. It will be very tempting to assume that of the three extensions presented here, it is only necessary to analyze one or two in depth, and then the others will "obviously" follow from it (or them.) There is a reason why this approach was not used in this thesis. Such a belief is not correct; while many of the lemmas, theorems, and

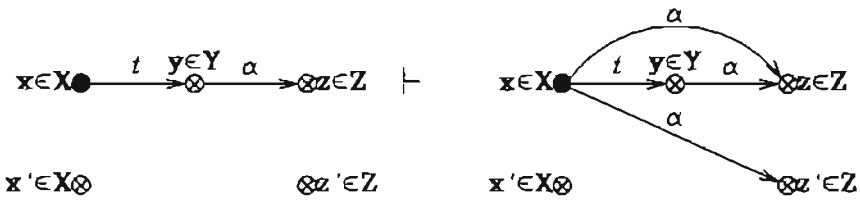
definitions resemble each other, they are by no means identical; and in cases where proofs are necessary, the proofs of two very similar claims will be quite different. (As an example, look at lemmas 4.2, 4.16, and 4.31, proving the formal definition of "terminal span" meets the intuitive requirements.) This is a result of the different graph rewriting rules used by each extension.

IV.2. The One—Many Extension

In this extension, we change the graph rewriting rules so that one vertex acts upon a class of vertices. The new rules are:

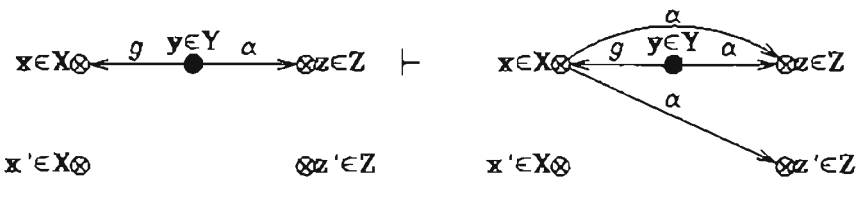
RULE R4.1: take$_{om}$

Let **x**, **y**, and **z** be vertices belonging to classes **X**, **Y**, and **Z**, respectively, and let **x** be a subject. Suppose **x** has $take_{om}$ rights over **y** and **y** has α rights over **z**. Then **x** obtains α rights over all vertices in **Z**. In pictures,



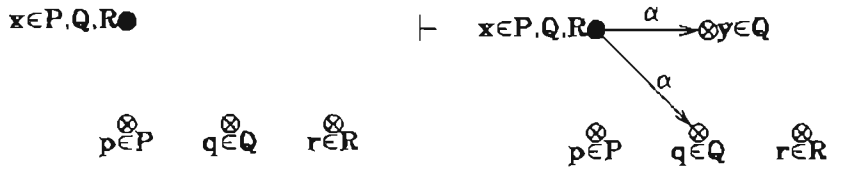
RULE R4.2: grant$_{om}$

Let **x**, **y**, and **z** be vertices belonging to classes **X**, **Y**, and **Z**, respectively, and let **y** be a subject. Suppose **y** has $grant_{om}$ rights over **x** and **y** has α rights over **z**. Then **x** obtains α rights over all vertices in **Z**. In pictures,



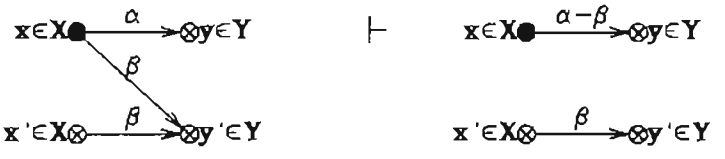
RULE R4.3: $\mathbf{create}_{om}$

Let **x** belong to a set of classes **S** and let **x** be a subject. Then **x** $create_{om}$ a new (subject or object) vertex **y** belonging to classes in a set **S'** $\subseteq$ **S**, with **x** having α rights over all vertices belonging to classes in **S'**. In pictures,



RULE R4.4: $\mathbf{remove}_{om}$

Let **x** and **y** be vertices belonging to classes **X** and **Y**, respectively, and let **x** be a subject. Suppose **x** has α rights over **y** and let $\beta \subseteq \alpha$. Then **x** $removes_{om}$ β rights over all vertices in **Y**. In pictures,

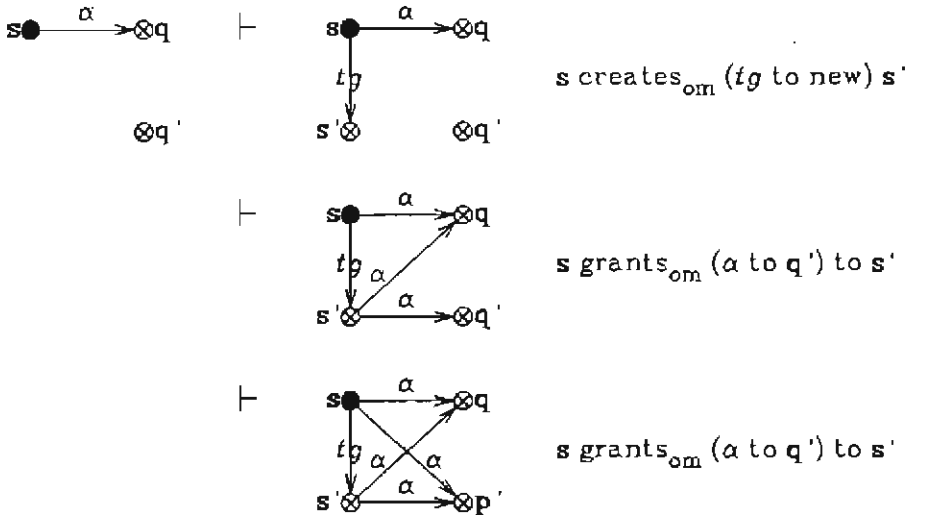


In what follows, as in the original model, we will not use the remove_{om} rule except where necessary, since it was included mainly for completeness; its presence rarely changes anything, because we (tacitly) assume that when an edge or right is added to the protection graph, it remains there.

Before we begin defining terms, there is one effect of these rules that will be seen over and over; basically, it says that if a subject has rights over one member of a class, it can obtain these rights over all members of that class. This is so useful that we state and prove it as a lemma:

LEMMA 4.1: Let **s** be a subject vertex, and let **q** and **q'** be vertices in the same class in a protection graph G_0. If **s** has α rights over **q**, then **s** can acquire α rights over **q'**.

PROOF: The following construction demonstrates the claim:



This completes the proof. ■

This lemma will have many consequences. The most interesting one is the effect on the transfer of rights from one vertex to another; it is sufficient for a vertex to obtain a right over any member of a class in order to gain that right over all members of that class. To take a more concrete example, think of files as objects, processes as subjects, and protection groups as classes. If a process can access a file, it can access all files in that protection group. Basically, that is all the lemma says.

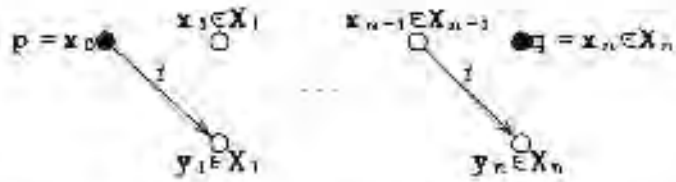
Our immediate goal is to determine necessary and sufficient conditions for the transfer of authority in this model. Let us now look at the analogue to initial and terminal spans and bridges.

Recall that a *terminal span* is a path along which rights can be obtained. That is, if **x** terminally spans to **y**, **x** can obtain any right **y** has. Under the four rules R4.1 – R4.4, it is clear that the terminal span need not go from **x** to **y** directly, but may go from **x** to any vertex in the same class as **y**; for, if the terminal span is of length 1, an appropriate application of the create$_{om}$ rule will

do this, whereas for longer bridges, repeated applications of the $take_{on}$ and $grant_{om}$ rules will be needed.

More formally,

DEFINITION D4.2: A subject vertex **p** *terminally*$_{om}$ spans to another (subject or object) vertex **q** if there exists a sequence of vertices $x_0, \ldots, x_n$, $n \geq 0$, such that $p = x_0$, $q = x_n$, and there are vertices $y_1, \ldots, y_n$ for which there is an edge labelled t from x_i to y_{i-1}, and y_{i+1} is in the same class as x_{i+1}. Pictorially,

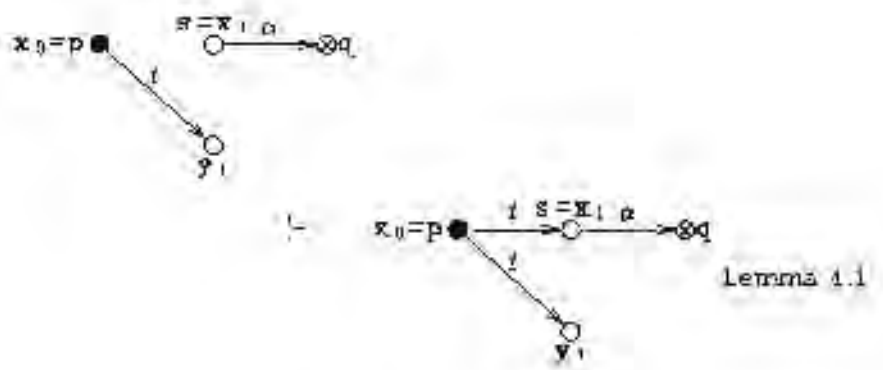


To see that this formal definition meets the intuitive one, we show:

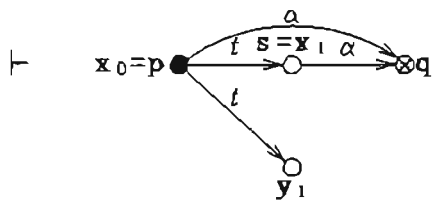
LEMMA 4.2: Let **p**, **s**, and **q** be vertices in a protection graph, let **s** have α rights over **q**, and let **p** terminally$_{om}$ span to **s**. Then **p** can obtain α rights over **q**.

PROOF: To prove this lemma, we induct on the length n of the terminal$_{om}$ span.

BASIS: $n = 1$. In this case, the following sequence of rule applications proves the lemma.



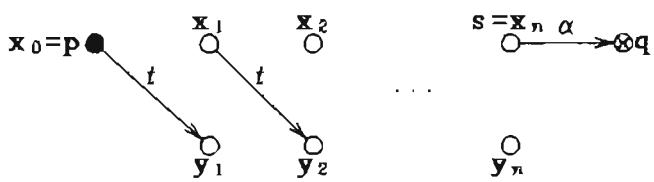
Lemma 4.1

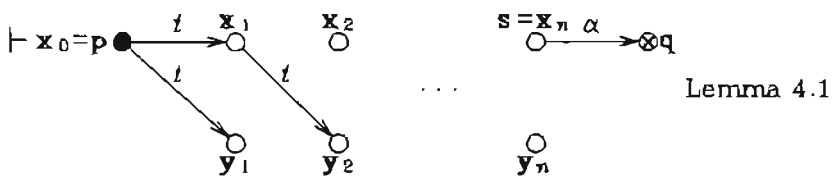


p takes$_{om}$ (α to **q**) from **s**

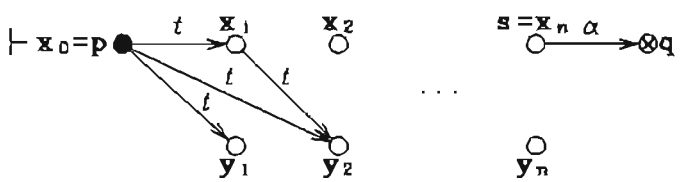
INDUCTION HYPOTHESIS: The claim is true for a terminal$_{om}$ span of length n, $n = 1, \ldots, k$.

INDUCTION STEP: Let $n = k + 1$. Then,





Lemma 4.1



p takes$_{om}$ (t_{om} to $\mathbf{y}_2$) from $\mathbf{x}_1$

Now, the terminal span is of length k, so the claim follows from the induction hypothesis. Hence, **p** can obtain α rights to **q**. ■

We can make this lemma more general:

COROLLARY 4.3: Let **q** and **q'** be vertices (not necessarily distinct) in the same class in a protection graph. Let the following two conditions hold:

C4.1. There is a vertex **s** with an α edge to **q**'.

C4.2. There is a subject vertex **s**' such that **s**' = **s** or **s**' terminally$_{om}$ spans to **s**.

Then **s**' can acquire α rights to **q**.

PROOF: If **s**' = **s**, then by lemma 4.1 we are done. If **q**' = **q**, the corollary is trivially true. Finally, by lemma 4.2, **s**' can acquire α rights over **q**'; then by lemma 4.1, **s**' can acquire α rights over **q**. ■

Going back to definitions, recall that an *initial span* is a path along which rights may be bestowed; that is, if **p** initially spans to **q**, **p** may pass any right it has to **q**. More formally,

DEFINITION D4.3: A subject vertex **p** *initially*$_{om}$ *spans* to a (subject or object) vertex **q** if there exists a sequence of vertices $\mathbf{x}_0, \ldots, \mathbf{x}_n$, $n > 0$, such that $\mathbf{p} = \mathbf{x}_0$, $\mathbf{q} = \mathbf{x}_n$, and there are vertices $\mathbf{y}_1, \ldots, \mathbf{y}_n$, $n > 0$, such that $\mathbf{y}_i$ and $\mathbf{x}_i$ are in the same class, there is an edge labelled g from $\mathbf{x}_{n-1}$ to $\mathbf{y}_n$, and for all $i \neq n$, there is an edge labelled t from $\mathbf{x}_{i-1}$ to $\mathbf{y}_i$.

As with terminal$_{om}$ spans, let us show that this definition meets the intuitive requirements.

LEMMA 4.4: Let the following condition hold:

C4.3. There is a subject vertex **p**' such that **p**' = **p** or **p**' initially$_{om}$ spans to **p**.

Then **p**' may pass to **p** any right it has.

PROOF: If **p**' = **p**, we are done. So, assume **p**' ≠ **p**; by corollary 4.2, taking **s**' = **p**', **q** = **p**, **q**' = $\mathbf{y}_n$, **s** = $\mathbf{x}_n$, and $\alpha = g$, the result follows. ■

Finally, we define the analogue of a bridge; recall that a bridge is a path along which rights may be transferred from one subject to another.

DEFINITION D4.4: Two subject vertices **p** and **q** are joined by a *bridge*$_{om}$ if one of the following conditions holds:

C4.4. Either of the two subjects terminally$_{om}$ spans to the other;

C4.5. Either of the two subjects initially$_{om}$ spans to the other;

C4.6. There is an object vertex **s** such that one of the subjects initially$_{om}$ spans to **s**, and the other terminally$_{om}$ spans to **s**.

Again, we show this formal definition meets the intuitive requirements.

LEMMA 4.5: Let **p** and **q** be subjects with a bridge between them. Then **p** can obtain any right **q** has.

PROOF: If case C4.4 applies, either **p** terminally$_{om}$ spans to **q** or vice versa; in either case, by definition D1.5 or by lemma 4.2, the claim is proven. If case C4.5 applies, either **p** initially$_{om}$ spans to **q** or vice versa; in either case, by definition D1.4 or by lemma 4.4, the claim is proven. If case C4.6 applies with **p** terminally$_{om}$ spanning to **s**, and **q** initially$_{om}$ spanning to **s**, by lemma 4.4 **q** can pass any right to **s**, which **p** can then acquire by lemma 4.2; if **p** initially$_{om}$ spans to **s**, and **q** terminally$_{om}$ spans to **s**, by the construction used to prove case 2 of lemma 3.1, we are done. ■

Note that, by lemma 4.1, if there is a bridge from a vertex in one class to a vertex in another class, either vertex can acquire rights over any member in the other class; therefore, we will often speak of a vertex and a class being *linked*$_{om}$. All this means is that there is a bridge from the vertex to a vertex in the class. The vertex linked$_{om}$ to the class, incidentally, will often be called a *link*$_{om}$ *vertex*.

We can now consider several classes with vertices linked by bridges.

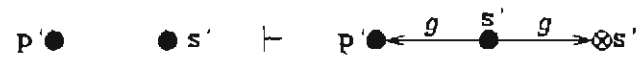
LEMMA 4.6: Let the following condition be true:

C4.7. There exists a sequence of classes $C_1, \ldots, C_m$ such that subjects $p' \in C_1$, $s' \in C_m$, and in each C_i there is a subject vertex c_i linked$_{om}$ to C_{i+1}, for $1 \leq i \leq m$.

Then p' can acquire any right s' has.

PROOF: We induct on the number m of classes.

BASIS: When $m = 1$, if $p' = s'$, we are done. If not,



and now s' can grant any right to p'.

INDUCTION HYPOTHESIS: For $m = 1, \ldots, k$, the claim holds for m classes.

INDUCTION STEP: Let $m = k + 1$, and consider c_2. By the induction hypothesis, c_2 can obtain any right s' has. By assumption, c_1 is a subject; hence, by lemma 4.1, c_1 can obtain any right c_2 has, so it can obtain any right s' has. And by using the create$_{om}$ rule as in the basis, c_1 can grant to p' any right it has. Hence, p' can acquire any right s' has. ■

Given lemmas 4.2, 4.4, and 4.6, we can characterize sharing completely. If conditions C4.1, C4.2, C4.3, and C4.7 hold, by lemma 4.2, s' can get α rights to q; by lemma 4.6, p' can get α rights to q from s'; and by lemma 4.4, p can be given α rights to q by p'. Hence, we define the predicate

DEFINITION D4.5: The predicate can•share$_{om}(\alpha, p, q, G_0)$ is true if and only if there is a finite sequence of the four om graph rewriting rules which, when applied to the protection graph G_0, result in an edge labelled α going from p to q.

The necessary and sufficient conditions for this predicate to be true are:

THEOREM 4.7: The predicate can•share(α, p, q, G_0) is true if and only if conditions C4.1, C4.2, C4.3, and C4.7 hold simultaneously.

PROOF: ($\Rightarrow$) Consider what happens if any one of the conditions fails. If C4.1 fails, no vertex can obtain α rights to q, since none of the graph rewriting rules add new incoming rights to a vertex; if C4.2 fails, no vertex in any class C_m can obtain α rights to q; if C4.3 fails, no vertex can grant_{om} p α rights to q; and if C4.7 fails, no vertex which can grant_{om} p rights will be able to obtain α rights to q. Hence, all four conditions must hold simultaneously for $\text{can•share}_{om}(\alpha, p, q, G_0)$ to be true.

($\Leftarrow$) Immediate from lemmas 4.2, 4.4, and 4.6. ■

Now that we have characterized sharing, let us think about stealing rights. Informally, we want the predicate $\text{can•steal}_{om}(\alpha, p, q, G_0)$ to be true whenever p can obtain α rights to q without the owner of that right granting it. However, suppose an owner grants to p the right to α a vertex q" in the same class as q. It is reasonable to bar this from taking place in a theft, because a vertex granting rights grants rights to a class, not to a vertex. So, if such a grant takes place, we shall not consider the action a theft.

More formally, we define the predicate

DEFINITION D4.6: Let p be a vertex and let q be a vertex in class Q. The predicate $\text{can•steal}_{om}(\alpha, p, q, G_0)$ is true if and only if all the following hold:

C4.8. there is no edge labelled α from p to q in G_0;

C4.9. there is a sequence $G_1, \ldots, G_n$ of protection graphs and $\rho_1, \ldots, \rho_n$ of rule applications such that

a. $G_0 \vdash_{\rho_1} \ldots \vdash_{\rho_n} G_n$,

b. there is an edge labelled α from $\mathbf{p}$ to $\mathbf{q}$ in G_n, and

c. if $\mathbf{s}$ has α rights over any $\mathbf{q}' \in \mathbf{Q}$ in G_0, no ρ_j has the form

$$\mathbf{s} \text{ grants}_{om} (\alpha \text{ to } \mathbf{q}') \text{ to } \mathbf{x}_j$$

for any $\mathbf{x}_j \in G_{i-1}$, $1 \leq i,j \leq n$.

Clearly, part C4.9c of the definition is the key part. We can modify the conditions necessary and sufficient for can•share$_{om}$ to be true to give conditions both necessary and sufficient for can•steal$_{om}$ to be true:

THEOREM 4.8: Let $\mathbf{p}$ and $\mathbf{q}$ be vertices in a protection graph G_0. Then can•steal$_{om}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true if and only if the following conditions all hold:

C4.10. there is no edge labelled α from $\mathbf{p}$ to $\mathbf{q}$ in G_0,

C4.11. there is a vertex $\mathbf{s}$ with an α edge to a vertex $\mathbf{q}'$ in the same class as $\mathbf{q}$,

C4.12. there is a subject vertex $\mathbf{p}'$ for which $\mathbf{p}' = \mathbf{p}$ or $\mathbf{p}'$ initially$_{om}$ spans to $\mathbf{p}$, and

C4.13. can•share$_{om}(t, \mathbf{p}', \mathbf{s}, G_0)$ is true.

INFORMAL ARGUMENT: To show the "only if" part, note C4.10 comes from the definition; as being true means can•share$_{om}$ is true, conditions C4.11 and C4.12 both hold. We can deduce C4.13 by showing the first rule to add an α edge to $\mathbf{q}'$ in $\mathbf{Q}$ must be a take$_{om}$ rule. Proving the "if" part merely requires that we show part C4.9c of the definition is met, which involves checking several cases.

PROOF: ($\Rightarrow$) Assume can•steal$_{om}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true. By part C4.9c of definition D4.6, condition C4.10 holds. By part C4.9b of the definition, can•share$_{om}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true, so condition C4.3 of theorem 4.7 gives condition C4.12 of this theorem, and condition C4.1 of theorem 4.7 gives

condition C4.11 of this theorem.

To prove condition C4.13 holds, consider the minimal length derivation sequence $G_0 \vdash_{\rho_1} \ldots \vdash_{\rho_n} G_n$ witnessing can•share$_{acm}(\alpha, p, q, G_0)$. Let i be the index for which G_i is the first graph in the derivation sequence where an edge labelled α to q' is added; that is, i is the first index for which $G_{i-1} \vdash_{\rho_i} G_i$, there is an edge labelled α from some vertex x to q' in G_i, but there is no edge labelled α from x to q' in G_{i-1}. Clearly, ρ_i is neither a create$_{acm}$ nor a remove$_{acm}$ rule. By part C4.9c of the definition, it cannot be a grant$_{acm}$ rule, since by our choice of i, all vertices with α rights to q' in G_{i-1} are also in G_0. Hence, ρ_i must be a take$_{acm}$ rule of the form

$$x \text{ takes}_{\alpha \text{ to } q'} \text{ from } s.$$

By C4.8, there is a subject vertex s' such that $s' = s$ or s' terminally$_{acm}$ spans to s. By C4.7, there are classes $C_1, \ldots, C_m$ such that $p' \subset C_1$ and $s' \subset C_m$.

Suppose s is an object. If s' is not in the same class as x, then our derivation sequence would not be minimal because there would be a shorter sequence not involving s'; hence, s' is in the same class as x, and terminally$_{acm}$ spans to x; thus, take $s' = x$. In this case, can•share$_{acm}(\alpha, p', s, G_0)$ holds.

Now suppose s is a subject; that is, $s' = s$. The vertex x is in some intermediate class C_j, $1 \le j \le m$. If $x \subset G_0$, all of C4.1, C4.2, C4.3, and C4.7 are satisfied, and we are done. So, let $x \notin G_0$. As $s \subset G_0$ and new labels on incoming edges cannot be added to extant classes, there must be a subject y in one class for which can•share$_{acm}(\alpha, y, s, G_0)$ is true. Note that any rule of the form

$$s \text{ grants}_{\alpha \text{ to } z} \text{ to } y$$

(where z is an arbitrary vertex) can be replaced by

$$\mathbf{x}\ \text{takes}_{om}\ (\alpha\ \text{to}\ \mathbf{z})\ \text{from}\ \mathbf{s}$$

$$\mathbf{x}\ \text{takes}_{om}\ (g\ \text{to}\ \mathbf{y})\ \text{from}\ \mathbf{s}$$

$$\mathbf{x}\ \text{grants}_{om}\ (\alpha\ \text{to}\ \mathbf{z})\ \text{to}\ \mathbf{s}$$

whenever $\mathbf{x} \neq \mathbf{y}$, and

$$\mathbf{x}\ \text{takes}_{om}\ (\alpha\ \text{to}\ \mathbf{z})\ \text{from}\ \mathbf{s}$$

whenever $\mathbf{x} = \mathbf{y}$. Thus, since $\mathbf{s}$ need never grant_{om} α to do the sharing, $\text{can•share}_{om}(t, \mathbf{y}, \mathbf{s}, G_0)$ is true. This satisfies C4.13 of this theorem.

($\Rightarrow$) Let the four conditions in the theorem hold. If $\mathbf{p}$ is a subject, it can take_{om} (α to $\mathbf{q}'$) from $\mathbf{s}$ since it gets the take_{om} right to $\mathbf{s}$, and hence get an α right to $\mathbf{q}$.

So, suppose $\mathbf{p}$ is an object. Then by condition C4.3, there must be a subject $\mathbf{p}'$ that initially_{om} spans to $\mathbf{p}$, and by condition C4.7, $\text{can•share}_{om}(t, \mathbf{p}', \mathbf{s}, G_0)$. If in G_0 $\mathbf{p}'$ does not have an α edge to $\mathbf{q}'$, then $\mathbf{p}'$ can take_{om} (α to $\mathbf{q}'$) from $\mathbf{s}$ and grant_{om} it to $\mathbf{p}$. If $\mathbf{p}'$ does have an α edge to $\mathbf{q}'$ in G_0, note simply that by conditions C4.3 and C4.7 there is a G_i such that $\mathbf{p}'$ has a take_{om} edge to $\mathbf{s}$ and a grant_{om} edge to $\mathbf{p}$, the following sequence enables $\mathbf{p}'$ to pass the right (α to $\mathbf{q}$) to $\mathbf{p}$ without ever granting_{om} it:

$$\mathbf{p}'\ \text{creates}_{om}\ (g\ \text{to new subject})\ \mathbf{z}$$

$$\mathbf{p}'\ \text{grants}_{om}\ (t\ \text{to}\ \mathbf{s})\ \text{to}\ \mathbf{z}$$

$$\mathbf{p}'\ \text{grants}_{om}\ (g\ \text{to}\ \mathbf{p})\ \text{to}\ \mathbf{z}$$

$$\mathbf{z}\ \text{takes}_{om}\ (\alpha\ \text{to}\ \mathbf{q})\ \text{from}\ \mathbf{s}$$

$$\mathbf{z}\ \text{grants}_{om}\ (\alpha\ \text{to}\ \mathbf{q})\ \text{to}\ \mathbf{p}$$

This is a witness for $\text{can•steal}_{om}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$, thus proving the theorem.

Thus far, we have dealt with the transfer of rights in a protection graph. Let us now consider the transfer of information. As in the original model, we shall define predicates analogous to can•know and can•snoop to test for information flow.

First, let us define conditions under which a vertex can grant$_{om}$, or take$_{om}$, read and write rights. Recall that an *rw*-initial path is a path along which information can be transmitted. We define an *rw*-initial$_{om}$ path to be the same under the om rules:

DEFINITION D4.7: A vertex **p** *rw-initially$_{om}$ spans* to **q** if there is a vertex **s** with a write edge to another vertex *q* " in the same class as **q**, and **p** terminally$_{om}$ spans to **s**, or **p** = **s**.

To see this meets the intuitive requirements, we now show:

LEMMA 4.9: Let the following condition be true:

C4.14. there is a subject **p**' which *rw*-initially$_{om}$ spans to **p**, or **p**' = **p**.

Then **p**' can acquire *w* rights to **p**.

PROOF: If **p**' ≠ **s**, as **p**' terminally$_{om}$ spans to **s**, **p**' can acquire *w* rights to **q**" by lemma 4.2. If **p**' = **s**, it already has those rights. As **q**" and **q** are in the same class, and **p**' is a subject, by lemma 4.1, **p**' can acquire *w* rights over **q**. ■

Also, recall that an *rw*-terminal span is a path along which information can be obtained. Again, we define an *rw*-terminal$_{om}$ span to be the same under the om rules:

DEFINITION D4.8: A vertex **p** *rw-terminally$_{om}$ spans* to **q** if there is a vertex **s** with a read edge to another vertex *q* " in the same class as **q**, and **p** terminally$_{om}$ spans to **s**, or **p** = **s**.

Once again, we must prove this definition meets the intuitive requirement:

LEMMA 4.10: Let the following condition hold:

C4.15. there is a subject **q'** which *rw*-terminally$_{om}$ spans to **q**, or **q'** = **q**.

Then **q'** can acquire read rights over **q**.

PROOF: If **q'** ≠ **s**, as **q'** terminally$_{om}$ spans to **s**, **q'** can acquire read rights to a vertex **q"** in the same class as **q**, by lemma 4.2. If **q'** = **s**, it has such rights to **q"**. As **q"** and **q** are in the same class, and **q'** is a subject, by lemma 4.1, **q'** can acquire read rights to **q**. ■

With these two definitions, we can develop an analogue to "connection" using the om rules. Informally, a join$_{om}$ is a path along which information, but not rights, may flow. More formally,

DEFINITION D4.9: Let **p** and **q** be subject vertices. If **p** *rw*-initially$_{om}$ spans to **q**, or **q** *rw*-terminally$_{om}$ spans to **p**, or **p** *rw*-initially$_{om}$ spans to a vertex to which **q** *rw*-terminally$_{om}$ spans, then **q** is said to be *joined$_{om}$* to **p**, and the path between them is called a *join$_{om}$*.

As usual, we check that this meets the intuitive requirements:

LEMMA 4.11: If **q** is joined$_{om}$ to **p**, then one of the following cases holds:

C4.16. **p** can obtain an explicit write edge to **q**;

C4.17. **q** can obtain an explicit read edge to **p**; or,

C4.18. **q** can obtain an implicit read edge to **p**.

PROOF: If **p** *rw*-initially$_{om}$ spans to **q**, by lemma 4.9, case C4.16 holds; if **q** *rw*-terminally$_{om}$ spans to **p**, by lemma 4.10, case C4.17 holds. So, suppose **p** *rw*-initially$_{om}$ spans to a vertex **x**, and **q** *rw*-terminally$_{om}$ spans to **x**. By lemma 4.4, **p** can acquire write rights over **x**, and by lemma 4.2 **q** can acquire read rights over **x**; then case C4.18 holds with an application of the post rule.

As a result, we may prove a stronger lemma, namely:

LEMMA 4.12: Let the following condition hold:

C4.19. There is a sequence of classes $D_1, \ldots, D_m$ such that there are subjects $p' \in D_1$, $q' \in D_m$, and there is a subject $d_i \in D_i$, $i \leq m$, such that either d_i is linked$_{om}$ to D_{i+1} or d_i is joined$_{om}$ to a subject vertex $d_{i+1} \subset D_{i+1}$.

Then one of three cases holds:

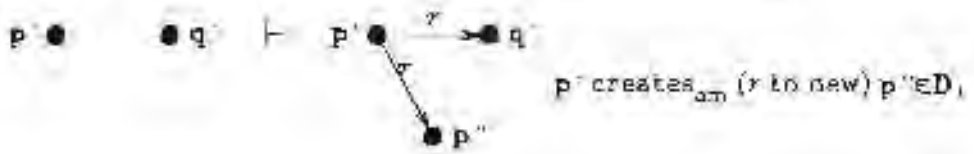
C4.20. q' can obtain an explicit write edge to p';

C4.21. p' can obtain an explicit read edge to q';

C4.22. p' can obtain an implicit read edge to q';

PROOF: The proof is by induction on the number m of classes.

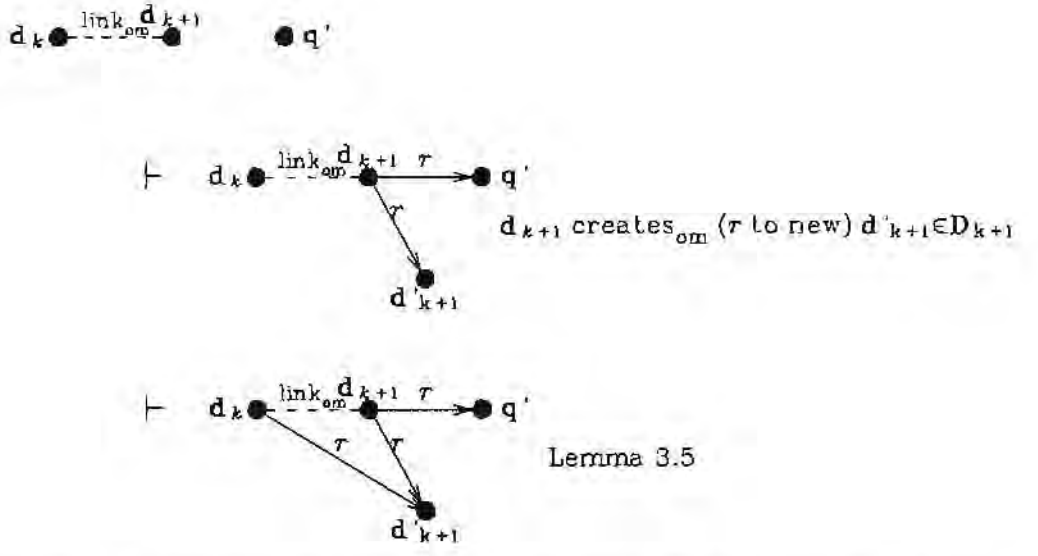
BASIS: $m = 1$. Then p' and q' are in the same class D_1; so,



and case C4.21 holds.

INDUCTION HYPOTHESIS: The claim holds for $m = 1, \ldots, k$.

INDUCTION STEP: Let $m = k + 1$. If d_k is linked$_{om}$ to D_{k+1}, let d_{k+1} be the link vertex. The following sequence of rule applications produces a witness to the claim:



and by the induction hypothesis, we are done. If $\mathbf{d}_k$ is joined$_{om}$ to $\mathbf{d}_{k+1}$, then from the first step of the above construction, it is clear that $\mathbf{d}_{k+1}$ can acquire read rights over $\mathbf{q}'$. By lemma 4.11 and the appropriate *de facto* rule (either the spy rule, if cases C4.17 or C4.18 of lemma 4.11 hold, or the pass rule, if case C4.16 of lemma 4.11 holds), $\mathbf{d}_k$ can obtain an explicit or implicit read edge to $\mathbf{q}'$. Apply the induction hypothesis to get the required result. ■

Notice that cases C4.20 and C4.21 will be possible only when *all* of the classes $\mathbf{D}_1, \ldots, \mathbf{D}_m$ are connected by links$_{om}$ since across joins$_{om}$ only information and not rights can be transferred. Only implicit edges can occur among classes which are joined$_{om}$.

With these terms we can now define the sharing of information:

DEFINITION D4.10: The predicate can•know$_{om}$(p, q, G_0) is true if and only if there is a finite sequence of om *de jure* and *de facto* rule applications resulting in an explicit write edge from **q** to **p**, or an (explicit or implicit) read edge from **p** to **q**.

In other words, if **p** can obtain information from **q**, can•know$_{om}$(**p**, **q**, G_0) is true. We can also state necessary and sufficient conditions for can•know$_{om}$ to hold:

THEOREM 4.13: Let **p** and **q** be vertices in a protection graph G_0. Then can•know$_{om}$(**p**, **q**, G_0) is true if and only if at least one of the following holds:

C4.23. can•share$_{om}$(α, **q**, **p**, G_0) is true; or,

C4.24. can•share$_{om}$(τ, **p**, **q**, G_0) is true; or,

C4.25. *all* of C4.14, C4.15, and C4.19 hold simultaneously.

INFORMAL ARGUMENT: Assume exactly one of the three conditions holds. By definition, if either conditions C4.23 or C4.24 hold, can•know$_{om}$(**p**, **q**, G_0) is true. So, assume only condition C4.25 holds. By lemmas 4.9, 4.10, and 4.12, **p** can obtain information from **q**. Going the other way, again if condition C4.23 or C4.24 is true, the result is immediate; if only condition C4.25 holds, we will show that if one of C4.14, C4.15, and C4.19 is false, can•know$_{om}$ is also false.

PROOF: ($\Rightarrow$) If condition C4.23 or condition C4.24 holds, can•know$_{om}$(**p**, **q**, G_0) is true by definition. So, assume both are false and condition C4.25 is true. By lemma 4.9, it suffices to have $\mathbf{p}' \in \mathbf{D}_1$ acquire information from **q**; by lemma 4.10, it suffices to have $\mathbf{q}' \in \mathbf{D}_m$ transmit information to $\mathbf{p}'$; and by lemma 4.12, this can be done. Hence, can•know$_{om}$(**p**, **q**, G_0) is true.

($\Leftarrow$) Assume can•know$_{om}$(**p**, **q**, G_0) is true. Then, if there is an explicit read edge from **p** to **q**, condition C4.24 holds; if there is an explicit write edge from **q** to **p**, condition C4.23 holds; so, assume there is an implicit read edge from **p** to **q**. If C4.14 fails, the implicit edge cannot originate at **p**; if C4.15 fails, the implicit edge cannot terminate at **q**; if C4.19 fails, no such implicit edge can exist. In any case, we have a contradiction, so all three conditions C4.14, C4.15, and C4.19 must hold simultaneously.

Now that we have established conditions for the sharing of information, let us consider the theft of information. To do this, we define a predicate analogous to can•snoop, namely

DEFINITION D4.11: Let **p** and **q** be vertices in a protection graph G_0. Then can•snoop$_{om}$(**p**, **q**, G_0) is true if, and only if,

C4.26. can•steal$_{om}$(r, **p**, **q**, G_0) is true; or,

C4.27. there exists a sequence of protection graphs and rule applications $G_0 \vdash_{\rho_1} \ldots, \vdash_{\rho_n} G_n$ for which all of the following hold:

a. there is no edge from **p** to **q** labelled r in G_0;

b. there is an edge from **p** to **q** labelled r in G_n; and

c. neither **q** nor any vertex directly connected to **q**" in the same class as **q** is an actor in a grant$_{om}$ rule or a *de facto* rule resulting in an (explicit or implicit) edge labelled r going to **q**, or an explicit read edge going to **q**".

Just as we derived necessary and sufficient conditions for can•steal$_{om}$, we can do so for can•snoop$_{om}$:

THEOREM 4.14: For distinct vertices **p** and **q** in a protection graph G_0, can•snoop$_{om}$(**p**, **q**, G_0) is true if and only if one of the following is true:

C4.28. can•steal$_{om}$(r, **p**, **q**, G_0) is true, or

C4.29. all of the following hold:

a. there is no edge from **p** to **q** labelled r in G_0;

b. there is a subject vertex **p**' such that **p** = **p**' or **p**' rw-initially$_{om}$ spans to **p**;

c. there is a vertex q' such that $q' \neq q$, q' is not directly connected to q, and if q is a subject, q' rw-terminally$_{am}$ spans to q; and

d. can•know$_{am}(p', q', G_0)$ is true.

INFORMAL ARGUMENT: If can•snoop$_{am}(p, q, G_0)$ is true, and can•steal$_{biz}(r, p, q, G_0)$ is false, we have to show the parts of condition C4.29 are true. Condition C4.29a follows from the definition; by part C4.27b of the definition, can•know$_{am}(p, q, G_0)$ is true, from which condition C4.29b springs. Also, by theorem 4.13, condition C4.15, we have q'. Combining this with the definition, it becomes clear that although q' rw-terminally$_{am}$ spans to q, $q' \neq q$ and there is no edge labelled r from q' to q in G_0. The proof that can•know$_{am}(p', q', G_0)$ is true involves proving that the first rule to add a read edge with target q is a take rule.

Going from the conditions to can•snoop$_{am}$ is straightforward.

PROOF: ($\Rightarrow$) Let can•snoop$_{am}(p, q, G_0)$ be true. If can•steal$_{am}(r, p, q, G_0)$ holds, we are done, since part C4.27a of the definition is part C4.26 of this theorem. So, assume can•steal$_{am}(r, p, q, G_0)$ is false.

Part C4.27d of definition D4.11 gives condition C4.29a of this theorem.

By condition C4.15 of theorem 4.13, there is a subject q' such that $q' = q$, or q' rw-terminally$_{am}$ spans to q.

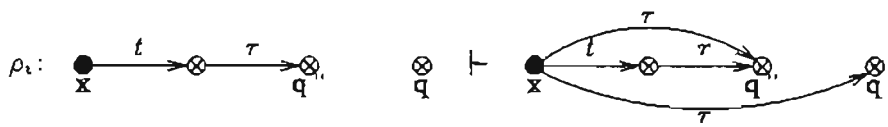
If q is an object, take $q' = q$ in condition C4.29c of this theorem.

If q is a subject, by part C4.9c of the definition of can•snoop$_{am}$, it is not used in the sequence of rule applications witnessing can•snoop$_{am}$. In this case, $q' \neq q$; choose q' in condition C4.29c to be this q'. Now, assume q' is directly connected to a vertex q'' in the same class as q with an edge labelled r. Either can•share$_{am}(t, p', q', G_0)$ is true (in which case can•steal$_{am}(t, p, q, G_0)$ is true,

contradicting assumption] or $\mathbf{q}'$ must actively participate in a grant_{om} spy, or pass rule application [which contradicts part C4.27c of the definition of can•snoop_{om}, contradicting assumption again]. In either case, there is no edge labelled τ from $\mathbf{q}$ to $\mathbf{q}''$ in G_0.

It remains to be shown that $\text{can•know}_{om}(\mathbf{p}', \mathbf{q}', G_0)$ is true. Let $G_0 \vdash_{\rho_1} \ldots \vdash_{\rho_n} G_n$ be a minimum length derivation sequence, and let i be the first index such that the first (explicit or implicit) read edge with target vertex in $\mathbf{q}$'s class is added in G_i. That is, i is the least index such that $G_{i-1} \vdash_{\rho_i} G_i$, there is no (explicit or implicit) read edge from any vertex $\mathbf{x}$ to $\mathbf{q}''$ in the same class as $\mathbf{q}$ in G_{i-1}, and there is an (explicit or implicit) read edge from $\mathbf{x}$ to $\mathbf{q}''$ in G_i. Consider what rule ρ_i was used to add this edge. It cannot be a grant_{om} rule because that would violate part C4.9c of definition D4.11. Nor can it be a spy, pass, or find rule, or a post rule, for this would violate the same part of the definition. As the create_{om} and remove_{om} rules do not add edges to existing classes, ρ_i cannot be either of these. Hence, ρ_i must be a take_{om} rule.

We therefore have:



Recalling that $\text{can•know}_{om}(\mathbf{p}, \mathbf{q}, G_0)$ is true, by theorem 4.13 we see $\text{can•know}_{om}(\mathbf{p}', \mathbf{q}, G_0)$ is true. Apply theorem 4.13 again; by this theorem, there is a subject vertex $\mathbf{q}'$ such that $\mathbf{q}' = \mathbf{q}$ or $\mathbf{q}'$ τw-terminally$_{om}$ spans to $\mathbf{q}$. Noting that there is no edge from $\mathbf{q}'$ to $\mathbf{q}$ labelled τ in G_0, we take $\mathbf{q}' = \mathbf{x}$ in theorem 4.13, whence $\text{can•know}_{om}(\mathbf{p}', \mathbf{q}', G_0)$ immediately follows.

($\Leftarrow$) If condition C4.28 of the theorem holds, by part C4.27a of the definition, $can \cdot snoop_{om}(\mathbf{p}, \mathbf{q}, G_0)$ is true.

So, assume condition C4.29 holds. Part C4.9a of the definition is the same as condition C4.29a of the theorem. By theorem 4.13, conditions C4.29b, C4.29c, and C4.29d establish part C4.9b of the definition. And as $\mathbf{q}' \neq \mathbf{q}$ when $\mathbf{q}$ is a subject, part C4.9c of the definition also holds.

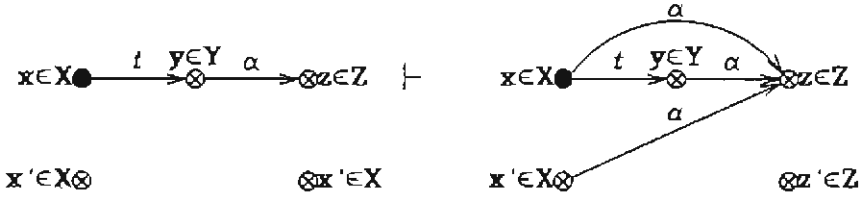
This completes the proof. ■

IV.3. The Many–One Extension

In this extension, we change the graph rewriting rules so that a class of vertices act upon one vertex. The new rules are:

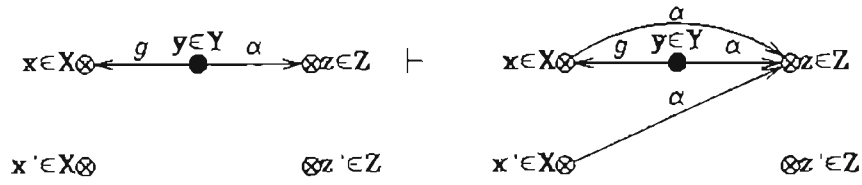
RULE R4.5: $take_{mo}$

Let $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ be vertices belonging to classes $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, respectively, and let $\mathbf{x}$ be a subject. Suppose $\mathbf{x}$ has $take_{mo}$ rights over $\mathbf{y}$ and $\mathbf{y}$ has α rights over $\mathbf{z}$. Then all vertices in $\mathbf{X}$ obtain α rights over $\mathbf{z}$. In pictures,



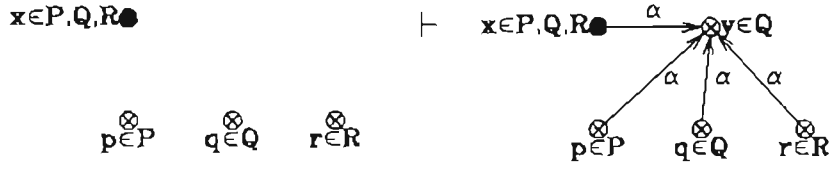
RULE R4.6: $grant_{mo}$

Let $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ be vertices belonging to classes $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, respectively, and let $\mathbf{y}$ be a subject. Suppose $\mathbf{y}$ has $grant_{mo}$ rights over $\mathbf{x}$ and $\mathbf{y}$ has α rights over $\mathbf{z}$. Then all vertices in $\mathbf{X}$ obtain α rights over $\mathbf{z}$. In pictures,



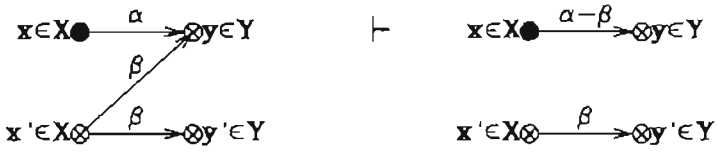
RULE R4.7: $\mathbf{create}_{mo}$

Let **x** belong to a set of classes **S** and let **x** be a subject. Then **x** *creates*$_{mo}$ a new (subject or object) vertex **y** belonging to classes in a set $\mathbf{S}' \subset \mathbf{S}$, with all vertices in class **S** having α rights over **y**. In pictures,



RULE R4.8: $\mathbf{remove}_{mo}$

Let **x** and **y** be vertices belonging to classes **X** and **Y**, respectively, and let **x** be a subject. Suppose **x** has α rights over **y** and let $\beta \subseteq \alpha$. Then all vertices in **X** *removes*$_{mo}$ β rights over **y**. In pictures,



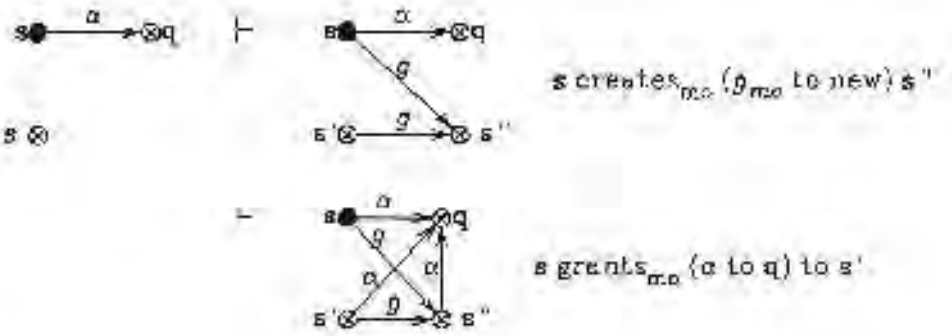
The remove_{mo} rule is present mainly for completeness; since we assume that, once given, a right has been exercised, and hence should remain exhibited in the corresponding protection graph as something to be considered when testing predicates, this rule is rarely used.

Before we begin defining terms, there is one effect of these rules that will be seen over and over; basically, it says that if a subject has rights over another vertex, all vertices in the subject's class can obtain these rights over that

vertex. This is so useful that we state and prove it as a lemma:

LEMMA 4.15: Let **s** be a subject vertex, let **s'** be a vertex in the same class as **s**, and let **q** be a vertex in a protection graph G_0. If **s** has α rights over **q**, then **s'** can acquire α rights over **q**.

PROOF:



This completes the proof. ■

Among the many interesting consequences of this lemma is its effect on the transfer of rights from one vertex to another. It is sufficient for one subject vertex in a class to obtain a right over a vertex, in order that all vertices in the subject's class get that right. To take a more concrete example, think of files as objects, and processes as subjects; each process is a member of several classes, each class corresponding to a group of which the process is a member, as well as the class of processes with the same owner. If a process can access a file, all processes owned by the same user can access that file. Basically, that is all the lemma says.

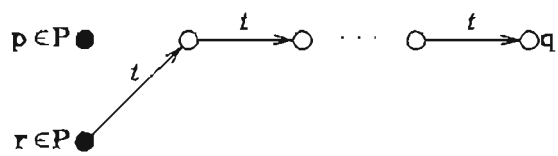
Our immediate goal is to determine necessary and sufficient conditions for the transfer of authority in this model. Let us now look at the analogue to initial and terminal spans and bridges.

Recall that a *terminal span* is a path along which rights can be obtained. That is, if **x** terminally spans to **y**, **x** can obtain any right **y** has. Under the four rules R4.1 – R4.4, it is clear that the terminal span need not go from **x** to **y** directly, but may go from any vertex in the same class as **x** to **y**.

More formally,

DEFINITION D4.12: A subject vertex **p** *terminally*$_{mo}$ *spans* to another (subject or object) vertex **q** if there is a vertex **r** in the same class as **p** that terminally spans to **q**.

Here, **r** spans to **q** with a terminal span, in the sense of the original model. Pictorially,

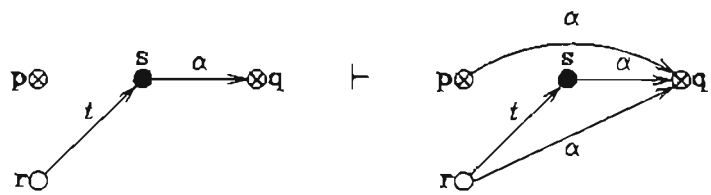


To see that this formal definition meets the intuitive one, we show:

LEMMA 4.16: Let **p**, **r**, and **s** be vertices in a protection graph, let **p** and **r** be in the same class, and let **r** terminally$_{mo}$ span to **s**. If **s** has α rights over **q**, then **p** can obtain α rights over **q**.

PROOF: We induct on the length m of the terminal$_{mo}$ span from **r** to **s**.

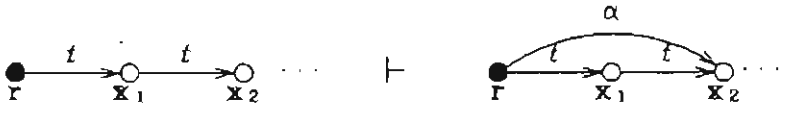
BASIS: $m = 1$. We have the following:



r takes$_{mo}$ (α to **q**) from **s**

INDUCTION HYPOTHESIS: The claim is true for a terminal$_{mo}$ span of length m, $m = 1, \ldots, k$.

INDUCTION STEP: Let $m = k + 1$. Let the object vertices on the terminal$_{mo}$ path from $\mathbf{r}$ to $\mathbf{s}$ be $\mathbf{x}_1, \ldots, \mathbf{x}_{m-1}$. It suffices to show that $\mathbf{r}$ can acquire take rights, at which point we can apply the induction hypothesis. But, since



$\mathbf{r}$ takes$_{mo}$ (t to $\mathbf{x}_2$) from $\mathbf{x}_1$

we are done. ■

We can make this lemma more general:

COROLLARY 4.17: Let q be a vertex in a protection graph. Let the following two conditions hold:

C4.30. There is a vertex $\mathbf{s}$ with an α edge to $\mathbf{q}$;

C4.31. There is a subject vertex $\mathbf{s}'$ such that $\mathbf{s}' = \mathbf{s}$ or $\mathbf{s}'$ terminally$_{mo}$ spans to $\mathbf{s}$.

Then $\mathbf{s}'$ can acquire α rights to $\mathbf{q}$.

PROOF: If $\mathbf{s}' = \mathbf{s}$, we are done. Suppose $\mathbf{s}' \neq \mathbf{s}$. By definition D4.12 and lemma 4.16, the corollary is immediate. ■

Going back to definitions, recall that an *initial span* is a path along which rights may be bestowed; that is, if $\mathbf{p}$ initially spans to $\mathbf{q}$, $\mathbf{p}$ may pass any right it has to $\mathbf{q}$. More formally,

DEFINITION D4.13: A subject vertex $\mathbf{p}$ *initially$_{mo}$ spans* to a (subject or object) vertex $\mathbf{q}$ if there is a subject $\mathbf{r}$ in the same class as $\mathbf{p}$ and $\mathbf{r}$ initially spans to $\mathbf{q}$.

Again, **r** spans to **q** in the sense of an initial span in the original model.

Again, let us show that this definition meets the intuitive requirements.

LEMMA 4.18: Let the following condition hold:

C4.32. There is a subject vertex **p'** such that **p'** = **p** or **p'** initially$_{mo}$ spans to **p**.

Then **p'** may pass to **p** any right it has.

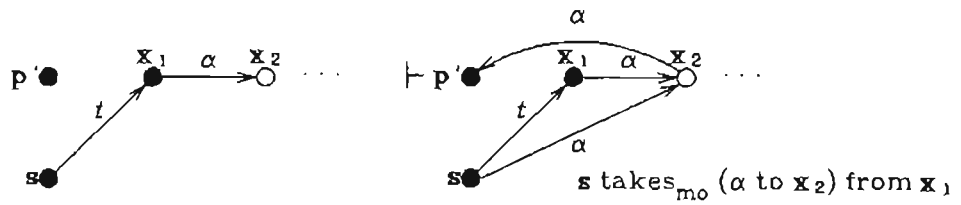
PROOF: Let **s** be a vertex in the same class as **p'**, and let **s** initially$_{mo}$ span to **p**. We induct on the length m of this initial span.

BASIS: $m = 1$. We have

Lemma 4.15

INDUCTION HYPOTHESIS: The claim holds when **s** initially spans to **p** with a path of length m, for $m = 1, \ldots, k$.

INDUCTION STEP: Let $m = k + 1$. Let the object vertices on the initial path be $\mathbf{x}_1, \ldots, \mathbf{x}_{m-1}$. It suffices to show that **s** can acquire $\mathbf{x}_1$'s rights over $\mathbf{x}_2$; then we merely apply the induction hypothesis. But, for $\alpha \in \{t, g\}$,



we are done. ■

Finally, we come to characterizing a *bridge*; recall that a bridge is a path along which rights may be transferred from subject to subject.

DEFINITION D4.14: Two subject vertices **p** and **q** are joined by a *bridge*$_{mo}$ if one of the following conditions holds:

C4.33. Either of the two subjects terminally$_{mo}$ spans to the other;

C4.34. Either of the two subjects initially$_{mo}$ spans to the other;

C4.35. There is an object vertex **s** such that one of the subjects initially$_{mo}$ spans to **s**, and the other terminally$_{mo}$ spans to **s**.

Again, we show this formal definition meets the intuitive requirements.

LEMMA 4.19: Let **p** and **q** be subjects with a bridge between them. Then **p** can obtain any right **q** has.

PROOF: If case C4.33 applies, either **p** terminally$_{mo}$ spans to **q** or vice versa; in either case, by definition D1.5 or by lemma 4.16, the claim is proven. If case C4.34 applies, either **p** initially$_{mo}$ spans to **q** or vice versa; in either case, by definition D1.4 or by lemma 4.18, the claim is proven. If case C4.35 applies with **p** terminally$_{mo}$ spanning to **s**, and **q** initially$_{mo}$ spanning to **s**, by lemma 4.18 **q** can pass any right to **s**, which **p** can then acquire by lemma 4.16; if **p** initially$_{mo}$ spans to **s**, and **q** terminally$_{mo}$ spans to **s**, by the construction used to prove case 2 of lemma 3.1, we are done. ■

Note that, by lemma 4.15, if there is a bridge from a subject vertex in one class to a subject in another class, any vertex in either class can acquire rights over any vertex in the other class; therefore, we will often speak of a class and a vertex being *linked*$_{mo}$. All this means is that there is a bridge from a vertex in the class to the vertex. The vertex linked$_{mo}$ to the class, incidentally, will often be called a *link*$_{mo}$ *vertex*.

One more lemma will be useful in our next proof:

LEMMA 4.20: Let the class **X** be linked$_{mo}$ to a vertex **c**, and let **c** have α rights over another vertex **q** distinct from **c** and the link$_{mo}$ vertex $\mathbf{x} \in \mathbf{X}$. Then any vertex $\mathbf{x}' \in \mathbf{X}$ can acquire α rights to **q**.

PROOF: Consider how **x** would obtain these rights. Either **c** would grant$_{mo}$ then to **x**, or **x** would take$_{mo}$ them (using the bridge$_{mo}$ from **x** to **c**.) When either of these is done, all vertices in **X** would acquire exactly the same rights as **x**, that is, α rights to **q**. Hence, **x**' can acquire α rights to **q**. ■

We can now consider several classes with vertices linked by bridges.

LEMMA 4.21: Let the following condition be true:

C4.7. There exists a sequence of classes $C_1, \ldots, C_m$ such that subjects $\mathbf{p}' \in C_1$, $\mathbf{s}' \in C_m$, and in each C_i there is a subject vertex $\mathbf{c}_i$ to which C_{i-1} is linked$_{mo}$, for $1 \leq i \leq m$, and $\mathbf{c}_m = \mathbf{s}'$.

Then **p**' can acquire any right **s**' has.

PROOF: We induct on the number m of classes.

BASIS: When $m = 1$, **p**' and **s**' are in the same class. If **p**' = **s**', we are done. If not, as **s**' is a subject, we need only apply lemma 4.15.

INDUCTION HYPOTHESIS: For $m = 1, \ldots, k$, the claim holds for m classes.

INDUCTION STEP: Let $m = k + 1$. As $\mathbf{s}' \in C_{k+1}$, by assumption there is a subject vertex $\mathbf{c}_k \in C_k$ with a bridge$_{mo}$ to **s**'. By lemma 4.20, $\mathbf{c}_k$ can acquire any rights **s**' has. At this point, we may apply the induction hypothesis to show that **p**' can acquire any right $\mathbf{c}_k$ has, whence **p**' can acquire any right **s**' has. ■

Given lemmas 4.16, 4.18, and 4.21, we can characterize sharing completely. If conditions C4.30, C4.31, C4.32, and C4.36 hold, by lemma 4.16, **s**' can get α rights to **q**; by lemma 4.21, **p**' can get α rights to **q** from **s**'; and by lemma 4.18, **p**

can be given α rights to **q** by **p'**. Hence, we define the predicate:

DEFINITION D4.15: The predicate can•share$_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true if and only if there is a finite sequence of the four mo graph rewriting rules which, when applied to the protection graph G_0, result in an edge labelled α going from **p** to **q**.

The necessary and sufficient conditions for this predicate to be true are:

THEOREM 4.22: The predicate can•share$_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true if and only if conditions C4.30, C4.31, C4.32, and C4.36 hold simultaneously.

PROOF: ($\Rightarrow$) Consider what happens if any one of the conditions fails. If C4.30 fails, no vertex can obtain α rights to **q**, since none of the graph rewriting rules add new incoming rights to a vertex; if C4.31 fails, no vertex in any class $\mathbf{C}_m$ can obtain α rights to **q**; if C4.32 fails, no vertex can grant$_{mo}$ **p** α rights to **q**; and if C4.36 fails, no vertex which can grant$_{mo}$ **p** rights will be able to obtain α rights to **q**. Hence, all four conditions must hold simultaneously for can•share$_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ to be true.

($\Leftarrow$) Immediate from lemmas 4.16, 4.18, and 4.21. ■

Now that we have characterized sharing, let us think about stealing rights. Informally, we want the predicate can•steal$_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ to be true whenever **p** can obtain α rights to **q** without the owner of that right granting it. However, suppose an owner grants to a vertex in the same class as **p** the right to α **q**. It is reasonable to bar this from taking place in a theft, because a vertex granting rights grants a *class*, and not a vertex, rights to a vertex , So, if such a grant takes place, we shall not consider the action a theft.

More formally, we define the predicate

DEFINITION D4.16: Let **p** be a vertex in class **P** and let **q** be a vertex. The predicate $can\text{-}steal_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true if and only if all the following hold:

C4.37. there is no edge labelled α from **p** to **q** in G_0;

C4.38. there is a sequence $G_1, \ldots, G_n$ of protection graphs and $\rho_1, \ldots, \rho_n$ of rule applications such that

a. $G_0 \vdash_{\rho_1}, \ldots, \vdash_{\rho_n} G_n$,

b. there is an edge labelled α from **p** to **q** in G_n, and

c. if **s** has α rights over **q** in G_0, no ρ_j has the form

$$\mathbf{s}\ grants_{mo}\ (\alpha\ to\ \mathbf{q}')\ to\ \mathbf{x}_j$$

for any $\mathbf{x}_j \in \mathbf{P}$ in G_{i-1}, $1 \leq i, j \leq n$.

Clearly, part C4.38c of the definition is the key part. We can modify the conditions necessary and sufficient for $can\text{•}share_{mo}$ to be true to give conditions both necessary and sufficient for $can\text{•}steal_{mo}$ to be true:

THEOREM 4.23: Let **p** and **q** be vertices in a protection graph G_0. Then $can\text{•}steal_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true if and only if the following conditions all hold:

C4.39 there is no edge labelled α from **p** to **q** in G_0,

C4.40 there is a vertex **s** with an α edge to **q** in G_0,

C4.41 there is a subject vertex **p'** for which **p'** = **p** or **p'** $initially_{mo}$ spans to **p**, and

C4.42 $can\text{•}share_{mo}(t, \mathbf{p}', \mathbf{s}, G_0)$ is true.

INFORMAL ARGUMENT: To show the "only if" part, note C4.39 comes from the definition; as $can\text{•}steal_{mo}$ being true means $can\text{•}share_{mo}$ is true, conditions C4.40 and C4.41 both hold. To show that C4.42 holds, we show that the first

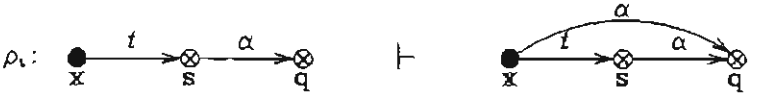
rule to add an α edge to $\mathbf{q}$ must be a take_{mo} rule, and then we show the condition cannot fail to hold. Proving the "if" part merely requires that we show part C4.38c of the definition is met, which we do by looking at various cases.

PROOF: ($\Rightarrow$) Suppose $\text{can•steal}_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true. By part C4.37 of definition D4.16, condition C4.39 holds. By part C4.38b of the definition, $\text{can•share}_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true, which by conditions C4.32 and C4.30 of theorem 4.22 give conditions C4.41 and C4.40 of this theorem. All that is left is to show $\text{can•share}_{mo}(t, \mathbf{p}', \mathbf{s}, G_0)$ is true.

To prove this, we first require (without loss of generality) that the derivation sequence $G_0 \vdash_{\rho_1} , \ldots , \vdash_{\rho_n} G_n$ witnessing $\text{can•share}_{mo}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ be of minimal length. Let i be the index for which G_i is the first graph in the derivation sequence where an edge labelled α to $\mathbf{q}'$ is added; that is, i is the first index for which $G_{i-1} \vdash_{\rho_i} G_i$, there is an edge labelled α from some vertex $\mathbf{x}$ in class $\mathbf{X}$ to $\mathbf{q}'$ in G_i, but there is no edge labelled α from $\mathbf{x}$ to $\mathbf{q}'$ in G_{i-1}. What rule ρ_i added this edge? The rule is neither a create_{mo} nor a remove_{mo} rule, neither of which add incoming edges to existing vertices. By part C4.38c of the definition, it cannot be a grant_{mo} rule, since by our choice of i, all vertices with α rights to $\mathbf{q}$ in G_{i-1} are also in G_0. Hence, ρ_i must be a take_{mo} rule of the form

$$\mathbf{x} \text{ takes}_{mo} (\alpha \text{ to } \mathbf{q}) \text{ from } \mathbf{s},$$

In pictures,



By C4.32, there is a subject vertex $\mathbf{s}'$ such that $\mathbf{s}' = \mathbf{s}$ or $\mathbf{s}'$ terminally_{mo} spans to $\mathbf{s}$. By C4.7, there are classes $\mathbf{C}_1, \ldots, \mathbf{C}_m$ such that $\mathbf{p}' \in \mathbf{C}_1$ and $\mathbf{s}' \in \mathbf{C}_m$, and there are vertices $c_i \in \mathbf{C}_i$ such that $\mathbf{C}_{i-1}$

is joined$_{mo}$ to $\mathbf{c}_i$, $1 \leq i \leq m$, with $\mathbf{s}' = \mathbf{c}_m$.

Suppose $\mathbf{s}$ is an object. Then $\mathbf{s} \neq \mathbf{s}$ and $\mathbf{s}'$ is not in the same class as $\mathbf{x}$, then our derivation sequence would not be minimal because we could eliminate the class containing $\mathbf{s}'$ from the sequence $\mathbf{C}_1, \ldots, \mathbf{C}_m$; hence, if $\mathbf{s}$ is an object, take $\mathbf{s}' = \mathbf{x}$, as $\mathbf{s}'$ and $\mathbf{x}$ are in the same class and both terminally$_{mo}$ span to $\mathbf{x}$. Hence, if $\mathbf{s}$ is an object, can•share$_{mo}(t, \mathbf{p}', \mathbf{s}, G_0)$ holds.

Suppose, therefore, $\mathbf{s}$ is a subject; that is, $\mathbf{s}' = \mathbf{s}$. The vertex $\mathbf{x}$ is in $\mathbf{C}_m$ or $\mathbf{C}_{m-1}$; hence, if $\mathbf{x} \in G_0$, all of C4.30, C4.31, C4.32, and C4.36 are satisfied, and we are done. If not, since $\mathbf{s} \in G_0$ and new labels on incoming edges cannot be added to existing vertices, there must be some subject $\mathbf{y}$ in some class such that can•share$_{mo}(t, \mathbf{y}, \mathbf{s}, G_0)$ holds. Note that $\mathbf{s}$ need never grant$_{mo}$ α to accomplish the sharing, as

$$\mathbf{x} \text{ takes}_{mo} (\alpha \text{ to } \mathbf{q}) \text{ from } \mathbf{s}$$

if $\mathbf{x} = \mathbf{y}$, and

$$\mathbf{s} \text{ grants}_{mo} (\alpha \text{ to } \mathbf{q}) \text{ to } \mathbf{y}$$

can be replaced by

$$\mathbf{x} \text{ takes}_{mo} (\alpha \text{ to } \mathbf{q}) \text{ from } \mathbf{s}$$

$$\mathbf{x} \text{ takes}_{mo} (g \text{ to } \mathbf{y}) \text{ from } \mathbf{s}$$

$$\mathbf{x} \text{ grants}_{mo} (\alpha \text{ to } \mathbf{q}) \text{ to } \mathbf{y}$$

if $\mathbf{x} \neq \mathbf{y}$. Thus, since $\mathbf{s}$ need never grant$_{mo}$ α to do the sharing, can•share$_{mo}(t, \mathbf{y}, \mathbf{s}, G_0)$ is true. This satisfies C4.42 of this theorem.

($\Rightarrow$) Let the four conditions in the theorem hold. If $\mathbf{p}$ is a subject, it can take$_{mo}$ (α to $\mathbf{q}'$) from $\mathbf{s}$ since it gets the take$_{mo}$ right to $\mathbf{s}$, and hence get an α right to $\mathbf{q}$.

So, suppose p is an object. Then by condition C4.32, there must be a subject p' that initially$_{mo}$ spans to p. By condition C4.38, can•share$_{mo}$(t, p', s, G_0). If in G_0 p' does not have an α edge to q, then p' can take$_{mo,\alpha}$ (α to q) from s and grant$_{mo}$ it to p. If p' does have an α edge to q in G_0, note simply that by conditions C4.32 and C4.36 there is a G_i such that p' has a take$_{mo}$ edge to s and a grant$_{mo}$ edge to p; hence, the following sequence enables p to pass the right (α to q) to p without ever granting$_{mo}$ it:

p' creates$_{mo}$ (g to new subject) z

p' grants$_{mo}$ (t to s) to z

p' grants$_{mo}$ (g to p) to z

z takes$_{mo}$ (α to q) from s

z grants$_{mo}$ (α to q) to p

This is a witness for can•steal$_{mo}$(α, p, q, G_0), thus proving the theorem. ■

Thus far, we have dealt with the transfer of rights in a protection graph. Let us now consider the transfer of information. As in the original model, we shall define predicates analogous to can•know and can•snoop to test for information flow.

First, let us define conditions under which a vertex can grant$_{mo}$ or take$_{mo}$ read and write rights. Recall that an rw-initial path is a path along which information can be transmitted. We define an rw-initial$_{mo}$ path to be the same under the mo rules:

DEFINITION D4.17: A vertex **p** *rw-initially$_{mo}$ spans* to **q** if there is a vertex **s** in the same class as **p** which *rw*-initially spans to **q**.

To see this meets the intuitive requirements, we now show:

LEMMA 4.24: Let the following condition be true:

C4.43. there is a subject **p'** which *rw*-initially$_{mo}$ spans to **p**, or **p'** = **p**.

Then **p'** can acquire *w* rights to **p**.

PROOF: If **p** = **s**, by definition of *rw*-initial span, we are done. If **p'** ≠ **s**, then one of two cases arise: if the *rw*-initial span from **s** to **p** is of length 1, by lemma 4.15 the claim holds; if not, when **s** applies the take$_{mo}$ rule to acquire write rights over **p**, **p'** will also acquire write rights over **p**. ■

Also, recall that an *rw*-terminal span is a path along which information can be obtained. Again, we define an *rw*-terminal$_{mo}$ span to be the same under the mo rules:

DEFINITION D4.18: A subject vertex **p** *rw-terminally$_{mo}$ spans* to **q** if there is a vertex **s** such that **p** is in the same class as **s** and **s** *rw*-terminally spans to **q**.

Once again, we must prove this definition meets the intuitive requirement:

LEMMA 4.25: Let the following condition hold:

C4.44. there is a subject **q'** which *rw*-terminally$_{mo}$ spans to **q**, or **q'** = **q**.

Then **q'** can acquire read rights over **q**.

PROOF: If **q'** = **s**, by the definition of *rw*-terminal span, we are done. So, suppose **q'** ≠ **s**. As in the proof of the previous lemma, two cases arise: if the *rw*-terminal span from **s** to **q** is of length 1, by lemma 4.15, the claim holds; if not, when **s** applies the take$_{mo}$ rule to acquire read rights over **q**, **q'** will also acquire read rights over **q**. ■

With these two definitions, we can develop an analogue to "connection" using the mo rules. Informally, a $join_{mo}$ is a path along which information, but not rights, may flow. More formally,

DEFINITION D4.19: Let **p** and **q** be subject vertices. If **p** *rw*-initially$_{mo}$ spans to **q**, or **q** *rw*-terminally$_{mo}$ spans to **p**, or **p** *rw*-initially$_{mo}$ spans to a vertex to which **q** *rw*-terminally$_{mo}$ spans, then **q** is said to be *joined*$_{mo}$ to **p**, and the path between them is called a *join*$_{mo}$.

As usual, we check that this meets the intuitive requirements:

LEMMA 4.26: If **q** is joined$_{mo}$ to **p**, then one of the following cases holds:

C4.45. **p** can obtain an explicit write edge to **q**;

C4.46. **q** can obtain an explicit read edge to **p**; or,

C4.47. **q** can obtain an implicit read edge to **p**.

PROOF: If **p** *rw*-initially$_{mo}$ spans to **q**, by lemma 4.24, case C4.45 holds; if **q** *rw*-terminally$_{mo}$ spans to **p**, by lemma 4.25, case C4.46 holds. So, suppose **p** *rw*-initially$_{mo}$ spans to a vertex **x**, and **q** *rw*-terminally$_{mo}$ spans to **x**. By lemma 4.18, **p** can acquire write rights over **x**, and by lemma 4.16 **q** can acquire read rights over **x**; then case C4.47 holds with an application of the post rule. ■

As a result, we may prove a stronger lemma, namely:

LEMMA 4.27: Let the following condition hold:

C4.48. There is a sequence of classes $\mathbf{D}_1, \ldots, \mathbf{D}_m$ such that there are subjects $\mathbf{p}' \in \mathbf{D}_1$, $\mathbf{q}' \in \mathbf{D}_m$, and there is a subject $\mathbf{d}_i \in \mathbf{D}_i$, $i \leq m$, such that either $\mathbf{D}_{i-1}$ is linked$_{mo}$ to $\mathbf{d}_i$, or $\mathbf{d}_{i-1} \in \mathbf{D}_{i-1}$ is joined$_{mo}$ to $\mathbf{d}_i$, and $\mathbf{q}' = \mathbf{d}_m$.

Then one of three cases holds:

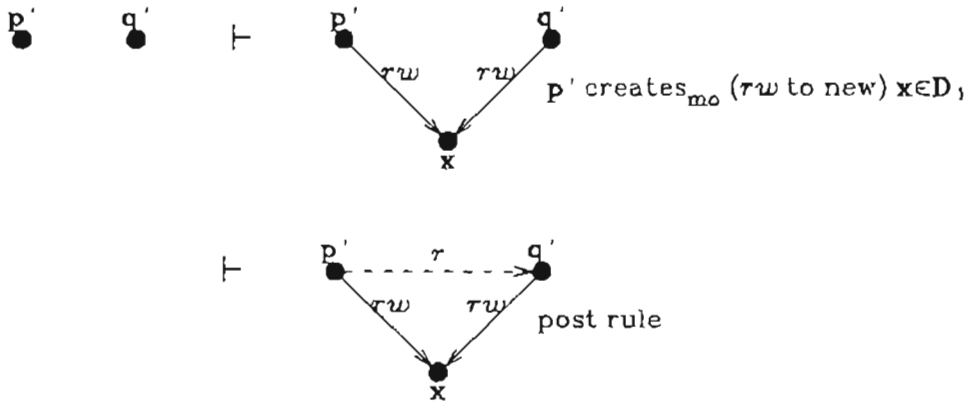
C4.49. $\mathbf{q}'$ can obtain an explicit write edge to $\mathbf{p}'$;

C4.50. $\mathbf{p}'$ can obtain an explicit read edge to $\mathbf{q}'$;

C4.51. $\mathbf{p}'$ can obtain an implicit read edge to $\mathbf{q}'$;

PROOF: The proof is by induction on the number m of classes.

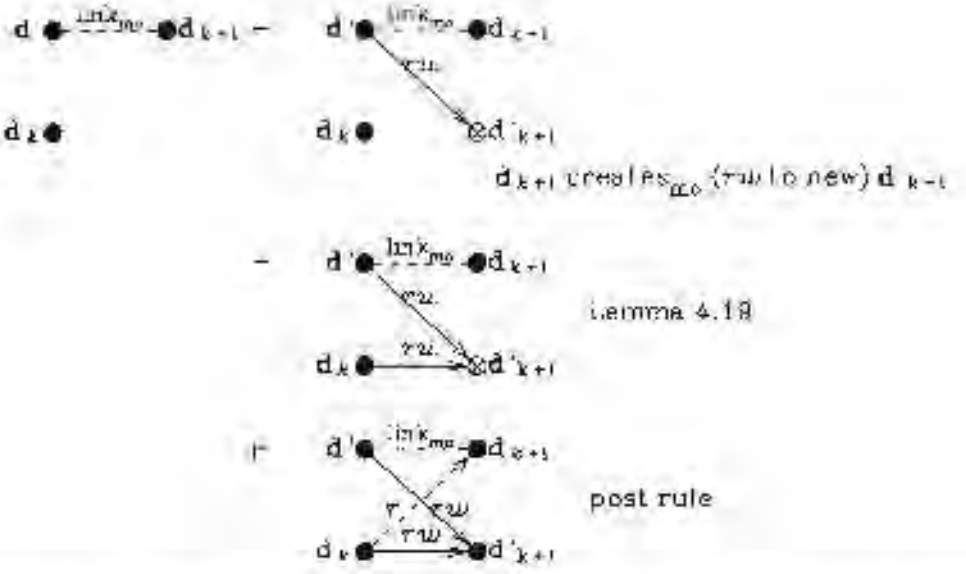
BASIS: $m = 1$. Then $\mathbf{p}'$ and $\mathbf{q}'$ are in the same class $\mathbf{D}_1$; so,



and case C4.51 holds.

INDUCTION HYPOTHESIS: The claim holds for $m = 1, \ldots, k$.

INDUCTION STEP: Let $m = k + 1$. If $\mathbf{D}_k$ is linked_{mo} to $\mathbf{q}' = \mathbf{d}_{k+1}$, let $\mathbf{d}'$ be the link_{mo} vertex. We must exhibit a witness to prove the claim. The following sequence of rule applications suffices:



and by the induction hypothesis, we are done. If d_k is joined$_{mo}$ to d_{k+1}, then from lemma 4.28, one of the three cases in that lemma apply. So apply the induction hypothesis to get the required result. ■

Notice that case C4.45 or C4.46 will be possible only when there are 2 classes and they are joined$_{mo}$ with either an rw-initial$_{mo}$ or rw-terminal$_{mo}$ span; otherwise, the read edge from p' to q' will always be implicit.

With these terms we can now define the sharing of information:

DEFINITION D4.20: The predicate can•know$_{mo}$(p, q, G_0) is true if and only if there is a finite sequence of *mo de jure* and *de facto* rule applications resulting in an explicit write edge from q to p, or an (explicit or implicit) read edge from p to q.

In other words, if p can obtain information from q, can•know$_{mo}$(p, q, G_0) is true. We can also state necessary and sufficient conditions for can•know$_{mo}$ to hold:

THEOREM 4.28: Let p and q be vertices in a protection graph G_0. Then can•know$_{mo}$(p, q, G_0) is true if and only if at least one of the following holds:

C4.52. can•share$_{mo}(w, \mathbf{q}, \mathbf{p}, G_0)$ is true; or,

C4.53. can•share$_{mo}(r, \mathbf{p}, \mathbf{q}, G_0)$ is true; or,

C4.54. *all* of C4.43, C4.44, and C4.48 hold simultaneously.

INFORMAL ARGUMENT: Assume exactly one of the three conditions holds. By definition, if either condition C4.52 or C4.53 holds, can•know$_{mo}(\mathbf{p}, \mathbf{q}, G_0)$ is true. So, assume only condition C4.54 holds. By lemmas 4.24, 4.25, and 4.27, **p** can obtain information from **q**. Going the other way, again if condition C4.52 or C4.53 is true, the result is obvious; if only condition C4.54 holds, we will show that if any of C4.43, C4.44, and C4.48 is false, can•know$_{mo}$ is also false.

PROOF: ($\Rightarrow$) If condition C4.52 or condition C4.53 holds, can•know$_{mo}(\mathbf{p}, \mathbf{q}, G_0)$ is true by definition. So, assume both are false and condition C4.54 is true. By lemma 4.24, it suffices to have $\mathbf{p}' \in \mathbf{D}_1$ acquire information from **q**; by lemma 4.25, it suffices to have **q**' acquire information from **q**; and by lemma 4.27, this can be done. Hence, can•know$_{mo}(\mathbf{p}, \mathbf{q}, G_0)$ is true.

($\Leftarrow$) Assume can•know$_{mo}(\mathbf{p}, \mathbf{q}, G_0)$ is true. Then, if there is an explicit read edge from **p** to **q**, condition C4.53 holds; if there is an explicit write edge from **q** to **p**, condition C4.52 holds; so, assume there is an implicit read edge from **p** to **q**. If C4.43 fails, the implicit edge cannot originate at **p**; if C4.44 fails, the implicit edge cannot terminate at **q**; if C4.48 fails, no such implicit edge can exist. In any case, we have a contradiction, so all three conditions C4.43, C4.44, and C4.48 must hold simultaneously. ■

Now that we have established conditions for the sharing of information, let us consider the theft of information. To do this, we define a predicate analogous to can•snoop, namely

DEFINITION D4.21: Let **p** and **q** be vertices in a protection graph G_0. Then can•snoop$_{mo}$(**p**, **q**, G_0) is true if, and only if,

C4.55. can•steal$_{mo}$(r, **p**, **q**, G_0) is true; or,

C4.56. there exists a sequence of protection graphs and rule applications $G_0 \vdash_{\rho_1} \ldots \vdash_{\rho_n} G_n$ for which all of the following hold:

a. there is no edge from **p** to **q** labelled r in G_0;

b. there is an edge from **p** to **q** labelled r in G_n; and

c. neither **q** nor any vertex directly connected to **q** is an actor in a grant$_{mo}$ rule or a *de facto* rule resulting in an (explicit or implicit) edge labelled r with **q** as its target.

Just as we derived necessary and sufficient conditions for can•steal$_{mo}$, we can do so for can•snoop$_{mo}$:

THEOREM 4.29: For distinct vertices **p** and **q** in a protection graph G_0, can•snoop$_{mo}$(**p**, **q**, G_0) is true if and only if one of the following is true:

C4.57. can•steal$_{mo}$(r, **p**, **q**, G_0) is true, or

C4.58. all of the following hold:

a. there is no edge from **p** to **q** labelled r in G_0;

b. there is a subject vertex **p'** such that **p** = **p'** or **p'** rw-initially$_{mo}$ spans to **p**;

c. if **q** is a subject, there is a vertex **q'** such that **q'** ≠ **q**, there is no read edge from **q'** to **q** in G_0, and **q'** rw-terminally$_{mo}$ spans to **q**; and

d. can•know$_{mo}$(**p'**, **q'**, G_0) is true.

INFORMAL ARGUMENT: If $can\text{-}snoop_{rw0}(p, q, G_0)$ is true, and $can\text{-}steal_{rw0}(r, p, q, G_0)$ is false, we have to show the parts of condition C4.58 are true. Condition C4.58a follows from the definition: by part C4.58b of the definition $can\text{-}know_{rw0}(p, q, G_0)$ is true, from which condition C4.58b springs. Also, by theorem 4.29, condition C4.58b, we have q'. Combining this with the definition, it becomes clear that although q' $rw\text{-}terminally_{rw0}$ spans to q, $q' \neq q$ and there is no edge labelled r from q' to q in G_0. The proof that $can\text{-}know_{rw0}(p', q', G_0)$ is true involves proving that the first rule to add a read edge with target q is a take rule.

Going from the conditions to $can\text{-}snoop_{rw0}$ is straightforward.

PROOF: ($\Rightarrow$) Let $can\text{-}snoop_{rw0}(p, q, G_0)$ be true. If $can\text{-}steal_{rw0}(r, p, q, G_0)$ holds we are done, since part C4.26 of the definition is part C4.57 of this theorem. So assume $can\text{-}steal_{rw0}(r, p, q, G_0)$ is false.

Part C4.58a of definition D4.31 gives condition C4.58a of this theorem.

By part C4.58a of the definition, there is an implicit read edge from p to q in G_0, whence by definition $can\text{-}snoop_{rw0}(p, q, G_0)$ is true; so condition C4.58b of this theorem comes from condition C4.43 of theorem 4.28.

By condition C4.44, there is a subject q' such that $q' = q$ or q' $rw\text{-}terminally_{rw0}$ spans to q.

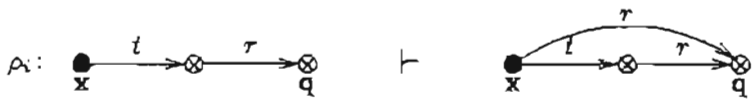
If q is an object, take q' to be the q' in condition C4.58c of this theorem.

If q is a subject, by part C4.58c of the definition of $can\text{-}snoop_{rw0}$, it is not used in the sequence of rule applications witnessing $can\text{-}snoop_{rw0}$. In this case, $q' \neq q$; choose q' in condition C4.58c to be this q'. Now, assume q' and q are directly connected with an edge labelled r in G_0. Either $can\text{-}share_{rw0}(r, p', q, G_0)$ is true (in which case $can\text{-}steal_{rw0}(r, p, q, G_0)$ is true, contradicting assumption) or q' must actively participate in a $grant_{rw0}$ or *de facto* rule application

[which contradicts part C4.56c of the definition of can•snoop$_{mo}$, contradicting assumption again]. In either case, there is no edge labelled r from $\mathbf{q}''$ to $\mathbf{q}$ in G_0.

It remains to be shown that can•know$_{mo}$($\mathbf{p}'$, $\mathbf{q}'$, G_0) is true. Let $G_0 \vdash_{\rho_1} \cdots \vdash_{\rho_n} G_n$ be a minimum length derivation sequence, and let i be the first index such that the first (explicit or implicit) read edge with target vertex in $\mathbf{q}$'s class is added in G_i. That is, i is the least index such that $G_{i-1} \vdash_{\rho_i} G_i$, there is no (explicit or implicit) read edge from any vertex $\mathbf{x}$ to $\mathbf{q}$ in G_{i-1}, and there is an (explicit or implicit) read edge from $\mathbf{x}$ to $\mathbf{q}$ in G_i. Consider what rule ρ_i was used to add this edge. It cannot be a grant$_{mo}$ rule because that would violate part C4.56c of definition D4.21. Nor can it be a *de facto* rule, for this would violate the same part of the definition. As the create$_{mo}$ and remove$_{mo}$ rules do not add edges to existing classes, ρ_i cannot be either of these. Hence, ρ_i must be a take$_{mo}$ rule.

We therefore have:



Recalling that can•know$_{mo}$($\mathbf{p}$, $\mathbf{q}$, G_0) is true, by theorem 4.28 we see can•know$_{mo}$($\mathbf{p}'$, $\mathbf{q}$, G_0) is true. Apply theorem 4.28 again; by this theorem, there is a subject vertex $\mathbf{q}'$ such that $\mathbf{q}' = \mathbf{q}$ or $\mathbf{q}'$ rw-terminally$_{mo}$ spans to $\mathbf{q}$. Noting that there is no edge from $\mathbf{q}'$ to $\mathbf{q}$ labelled r in G_0, we take $\mathbf{q}' = \mathbf{x}$ in theorem 4.28, whence can•know$_{mo}$($\mathbf{p}'$, $\mathbf{q}'$, G_0) immediately follows.

($\Leftarrow$) If condition C4.57 of the theorem holds, by part C4.55 of the definition, can•snoop$_{mo}$($\mathbf{p}$, $\mathbf{q}$, G_0) is true.

So, assume condition C4.58 holds. Part C4.56a of the definition is the same as condition C4.58a of the theorem. By theorem 4.28, conditions C4.58b, C4.58c, and C4.58d establish part C4.56b of the definition. And as $\mathbf{q}' \neq \mathbf{q}$ when $\mathbf{q}$ is a subject, part C4.56c of the definition is also true.

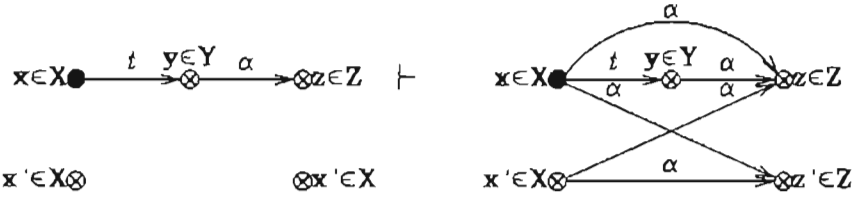
This completes the proof. ■

IV.4. The Many–Many Extension

In this extension, we change the graph rewriting rules so that one class of vertices acts upon another class of vertices. The new rules are:

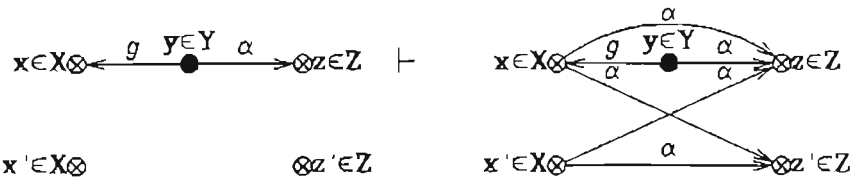
RULE R4.9: take$_{mm}$

Let $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ be vertices belonging to classes $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, respectively, and let $\mathbf{x}$ be a subject. Suppose $\mathbf{x}$ has $take_{mm}$ rights over $\mathbf{y}$ and $\mathbf{y}$ has α rights over $\mathbf{z}$. Then all vertices in $\mathbf{X}$ obtain α rights over all vertices in $\mathbf{Z}$. In pictures,



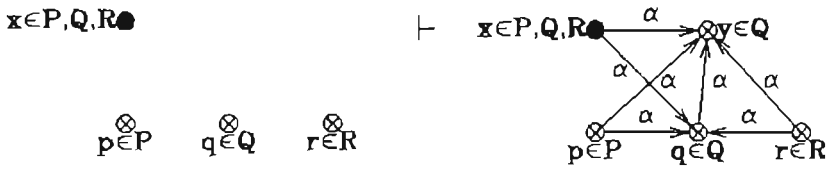
RULE R4.10: grant$_{mm}$

Let $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ be vertices belonging to classes $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, respectively, and let $\mathbf{y}$ be a subject. Suppose $\mathbf{y}$ has $grant_{mm}$ rights over $\mathbf{x}$ and $\mathbf{y}$ has α rights over $\mathbf{z}$. Then all vertices in $\mathbf{x}$ obtain α rights over all vertices in $\mathbf{Z}$. In pictures,



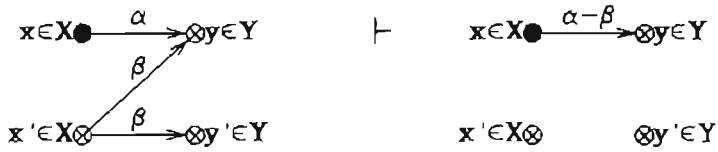
RULE R4.11: $\mathbf{create}_{mm}$

Let **x** belong.to a set of classes **S** and let **x** be a subject. Then **x** *creates*$_{mm}$ a new (subject or object) vertex **y** belonging to classes in a set $S' \subset S$, with all vertices in the same classes as **x** having α rights over all vertices in classes to which **y** belongs. In pictures,



RULE R4.12: $\mathbf{remove}_{mm}$

Let **x** and **y** be vertices belonging to classes **X** and **Y**, respectively, and let **x** be a subject. Suppose **x** has α rights over **y** and let $\beta \subset \alpha$. Then all vertices in **X** *removes*$_{mm}$ β rights over all vertices in **Y**. In pictures,

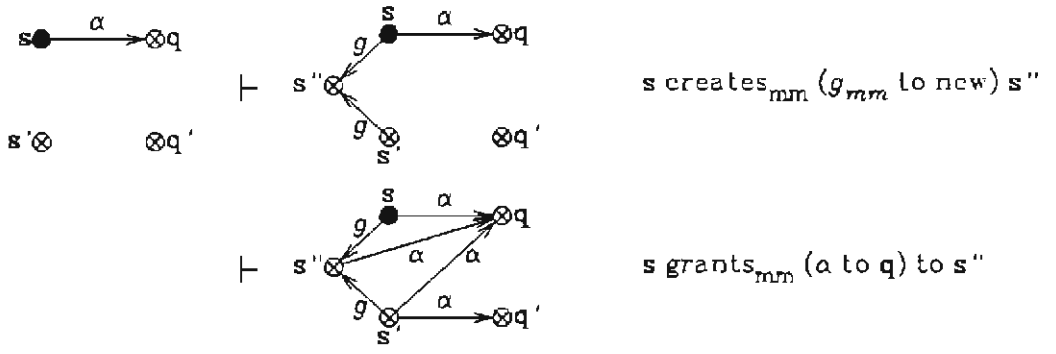


As in the original model and the previous two extensions, we will rarely use the remove_{mm} rule, because once added, it is very difficult do guarantee that deleting a right undoes all effects of its having existed. Thus, it is better to assume that once an edge is added to the protection graph, it remains there.

Using these rules, we can easily show that if a subject vertex in one class has the right to α a vertex in another class, then any member of the first class may obtain α rights to any member of the second class:

LEMMA 4.30: Let **s** be a subject vertex in class **S**, and let **s'** be a vertex distinct from **s** in **S**. Let **q** and **q'** be vertices in class **Q** in a protection graph G_0. If **s** has α rights over **q**, then **s'** can acquire α rights over **q'**.

PROOF:



This completes the proof. ■

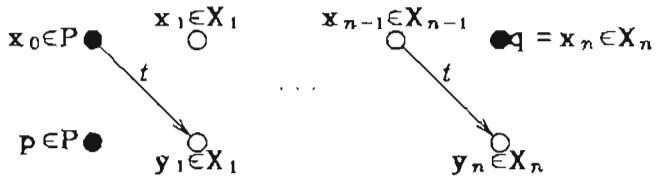
This lemma will have many consequences. The most interesting one is the effect on the transfer of rights from one vertex to another; it is sufficient for a member of one class to obtain a right over any member of a class in order that all members of the first class gain that right over all members of that class.

Our immediate goal is to determine necessary and sufficient conditions for the transfer of authority in this model. Let us now look at the analogue to initial and terminal spans and bridges.

Recall that a *terminal span* is a path along which rights can be obtained. That is, if **x** terminally spans to **y**, **x** can obtain any right **y** has. Using the four rules R4.9 – R4.12, we can define:

DEFINITION D4.22: A subject vertex **p** *terminally$_{mn}$ spans* to another (subject or object) vertex **q** if there exists a sequence of vertices $\mathbf{x}_0, \ldots, \mathbf{x}_n$, $n > 0$, with $\mathbf{x}_0$ a subject, such that **p** is in the same class as $\mathbf{x}_0$, $\mathbf{q} = \mathbf{x}_n$, and there are vertices $\mathbf{y}_1, \ldots, \mathbf{y}_n$ for which there is an edge labelled t from $\mathbf{x}_i$ to $\mathbf{y}_{i+1}$, and

y_{i+1} is in the same class as x_{i+1}. Pictorially,

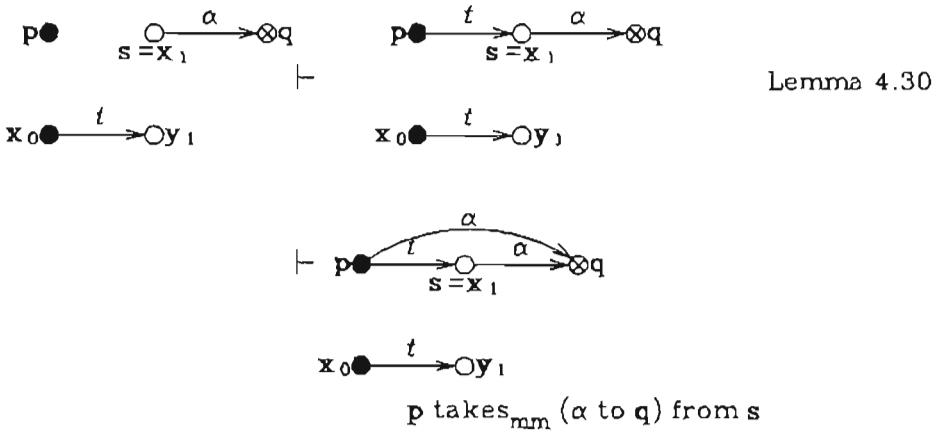


Now we show this definition agrees with our intuition:

LEMMA 4.31: Let **p**, **s**, and **q** be vertices in a protection graph, let **s** have α rights over **q**, and let **p** terminally$_{mm}$ span to **s**. Then **p** can obtain α rights over **q**.

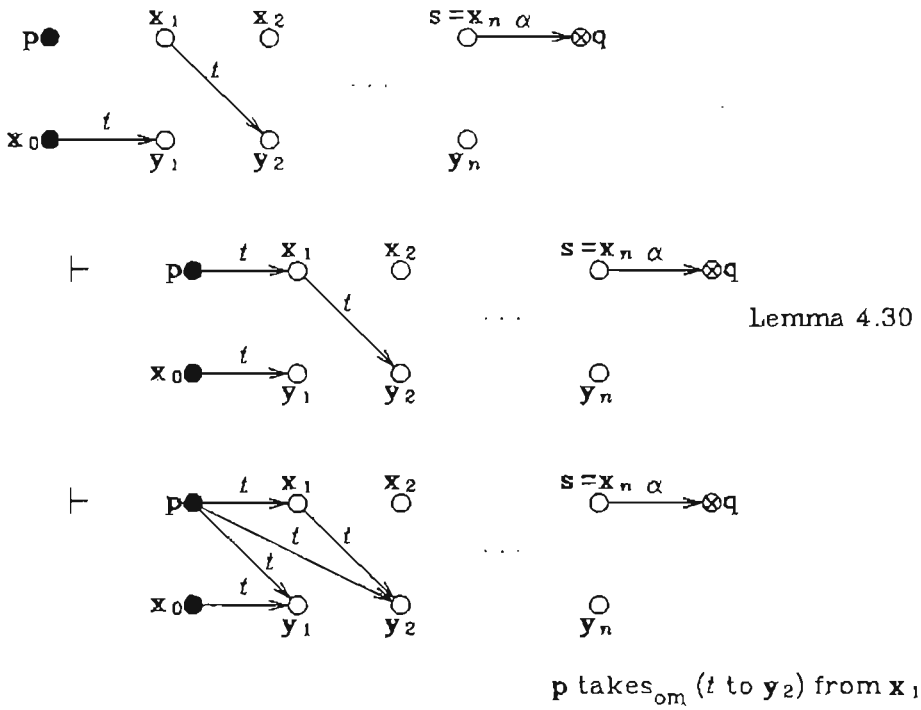
PROOF: We induct on the length m of the terminal$_{mm}$ span.

BASIS: $m = 1$. We have the following:



INDUCTION HYPOTHESIS: The claim is true for a terminal$_{mm}$ span of length m, $m = 1, \ldots, k$.

INDUCTION STEP: Let $m = k + 1$. Then apply the following sequence of rule applications, which witnesses a reduction of the length of the path involved:



p takes$_{om}$ (t to $\mathbf{y}_2$) from $\mathbf{x}_1$

Now, the terminal span is of length k, so the claim follows from the induction hypothesis. Hence, **p** can obtain α rights to **q**. ■

We can make this lemma more general:

COROLLARY 4.32: Let **q** and **q**' be vertices (not necessarily distinct) in the same class in a protection graph. Let the following two conditions hold:

C4.59. There is a vertex **s** with an α edge to **q**' in the same class as **q**;

C4.60. There is a subject vertex **s**' such that **s**' = **s** or **s**' terminally$_{mm}$ spans to a vertex **s**" in the same class as **s**.

Then **s**' can acquire α rights to **q**.

PROOF: If **s**' = **s**, then by lemma 4.30 we are done. If **s**" = **s**, we use lemma 4.31. If neither of these is true, we can still use lemma 4.31, taking **s** = **s**" in that lemma and noting that by the take$_{mm}$ rule, the claim holds. ■

We also do this for initial spans. recall that an initial span is a path along which rights may be bestowed; that is, if **p** initially spans to **q**, **p** may pass any right it has to **q**. More formally,

DEFINITION D4.23: A subject vertex **p** *initially*$_{mm}$ *spans* to a (subject or object) vertex **q** if there exists a sequence of vertices $\mathbf{x}_0, \ldots, \mathbf{x}_n$, $n > 0$, with $\mathbf{x}_0$ a subject, such that **p** and $\mathbf{x}_0$ are in the same class, $\mathbf{q} = \mathbf{x}_n$, and there are vertices $\mathbf{y}_1, \ldots, \mathbf{y}_n$, $n > 0$, such that $\mathbf{y}_i$ and $\mathbf{x}_i$ are in the same class, there is an edge labelled g from $\mathbf{x}_{n-1}$ to $\mathbf{y}_n$, and for all $i \neq n$, there is an edge labelled t from $\mathbf{x}_{i-1}$ to $\mathbf{y}_i$.

As with terminal$_{mm}$ spans, let us show that this definition meets the intuitive requirements.

LEMMA 4.33: Let the following condition hold:

C4.61. There is a subject vertex **p'** such that **p'** = **p** or **p'** initially$_{mm}$ spans to a vertex **p"** in the same class as **p**.

Then **p'** may pass to **p** any right it has.

PROOF: If **p'** = **p**, we are done. So, assume **p'** ≠ **p**; by corollary 4.31, taking **q'** = **p**, **s** = $\mathbf{x}_n$, **s'** = **p'**, **s"** = **p"**, and $\alpha = g$, the result follows. ∎

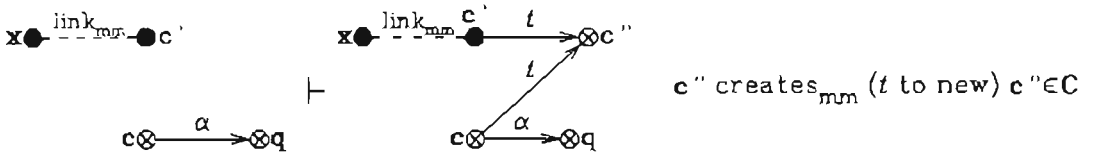
Finally, we define the analogue of a *bridge*, called a link$_{mm}$, as:

DEFINITION D4.24: Let **C** and **X** be distinct classes, and let **c** and **x** be subject vertices in **C** and **X**, respectively. If there is a bridge from **c** to **x**, then class **X** is said to be *linked*$_{mm}$ to class **C**, and **x** and **c** are called *linked*$_{mm}$ *vertices*.

Recall that a bridge is a path along which rights may be transferred from subject to subject; specifically, in the above definition. **x** can acquire rights from **c**. Given rules R4.9 – R4.12, we have:

PROPOSITION 4.34: Let **X** and **C** be linked$_{mm}$ classes, with $\mathbf{c} \in \mathbf{C}$, and let **c** have α rights over a vertex **q** distinct from the link vertex $\mathbf{x} \in \mathbf{X}$ and **c**. Then **x** can acquire α rights over **q**.

PROOF: Choose **c'** to be the link$_{mm}$ vertex in **C** associated with the bridge from **x** to **c**. Then



Then, by definition of bridge, **x** can obtain take$_{mm}$ rights to **c''** and due to the graph rewriting rules, also to **c**; at which point, **x** takes$_{mm}$ (α to **q**) from **c**. ■

One comment about this proof — it is very tempting to create$_{mm}$ the edge from **c'** to **c''** as being labelled t, g, and simply have **c** grant α rights to **q** to **c''**, whence **x** can obtain the right directly. The only problem is that this sequence requires **c** to be a subject, something not assumed in proposition 4.34.

Bearing in mind that when **x** either takes$_{mm}$ or is granted$_{mm}$ a right, all vertices in the class **X** (to which **x** belongs) obtain the same right, we have as a corollary:

COROLLARY 4.35: Let **X** be linked$_{mm}$ to **C**, and let $\mathbf{c} \in \mathbf{C}$ have α rights over a vertex **q** distinct from any vertex $\mathbf{x} \in \mathbf{X}$ and **c**. Then **x** can acquire α rights over **q**.

PROOF: Follows from the link$_{mm}$ vertex taking$_{mm}$ α rights to **q**. ■

The main reason this corollary is mentioned is that the proposition 4.34 requires **x** to be an endpoint of a bridge, and hence a subject. The corollary shows that the result in the proposition is true whether or not **x** is a subject, so long as all other conditions are met.

Recall that rights can be transmitted in either direction along a bridge. That is, if one endpoint has α rights to a vertex, the other endpoint can also obtain that right. This leads to

LEMMA 4.36: The relation linked$_{mm}$ is symmetric.

PROOF: Let **X** be linked$_{mm}$ to **C**, and let **x** and **c** be their respective link$_{mm}$ vertices. Recall that a bridge is a tg-path with associated word in $\{\vec{t}^{\,*}, \vec{t}^{\,*}, \vec{t}^{\,*}\vec{g}\vec{t}^{\,*}, \vec{t}^{\,*}\vec{g}\vec{t}^{\,*}\}$. Clearly, if there is a bridge from **x** to **c**, there is also a bridge from **c** to **x**. Thus, **C** is also linked$_{mm}$ to **X**. ■

To reflect this symmetry, we will speak of classes as being linked$_{mm}$ unless we wish to emphasize a particular direction of the link$_{mm}$. As a result,

COROLLARY 4.37: Let **D** and **C** be linked$_{mm}$ classes with at least one subject each. Then any vertex in either class may acquire any rights that another vertex in either class has.

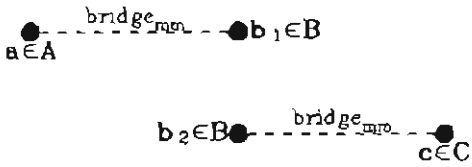
PROOF: If both vertices are in the same class, then by corollary 4.34 a vertex in the other class may acquire the right, whereupon the vertex seeking to acquire the right may use that corollary to get it. If the vertices are in different classes, this corollary may be applied directly. ■

Also true is:

COROLLARY 4.38: Let **A** and **B** be linked$_{mm}$, and **B** and **C** be linked$_{mm}$. Then any vertex in **A** may acquire any rights a vertex in **C** has.

PROOF: Apply corollary 4.37 twice. ■

One word of warning – corollary 4.38 does not mean that the relation linked$_{mm}$ is transitive. In fact, linked$_{mm}$ is not transitive; here is a counterexample:

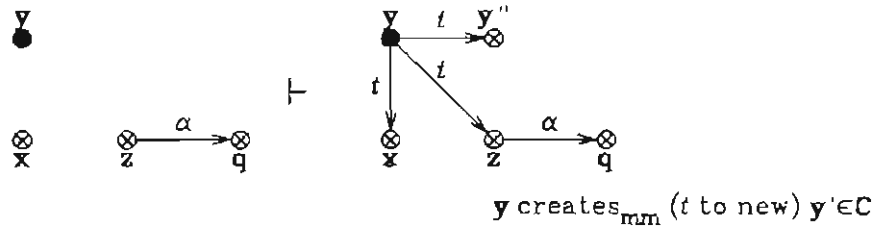


where A, B, and C are the three relevant classes. Obviously, A and B are linked$_{mm}$, and B and C are linked$_{mm}$, but A and C are not linked$_{mm}$.

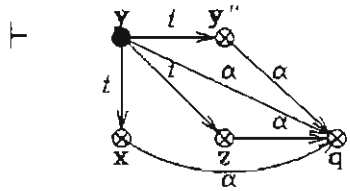
Given corollary 4.38, an immediate question is whether or not one vertex in the same class as a second can obtain any right the second has. In general, the answer is no; the class may be an object-only class. But if the class has one subject, the result is different:

LEMMA 4.39: Let C be a class, and let **x**, **y**, and **z** ∈ C with **y** a subject. Then **x** can obtain any right **z** has.

PROOF: Let **z** have α rights over **q**. Then



y creates$_{mm}$ (t to new) **y'**∈C



y takes$_{mm}$ (α to **q**) from **z**

This completes the proof. ■

Again, note **x** need not be the subject.

Combining all these lemmas, propositions, and corollaries, we can state and prove the following lemma, which provides a basis for the sharing of rights

by vertices in different, but linked, classes.

LEMMA 4.40: Let the following condition be true:

C4.62: There exists a sequence of classes $C_1, \ldots, C_m$ such that subjects $p' \in C_1$, $s' \in C_m$, and each C_i is $\text{linked}_{\alpha\beta}$ to C_{i+1} for $1 \leq i \leq m$.

Then p' can acquire any right s' has.

PROOF: Let s' have α rights over q. We induct on the number m of classes.

BASIS: When $m = 1$, p' and s' are in C_1 and, as both are subjects, lemma 4.30 applies to demonstrate the claim.

INDUCTION HYPOTHESIS: For $m = 1, \ldots, k$, the claim holds for m classes.

INDUCTION STEP: Let $m = k + 1$, and consider c_k, the link_{mm} vertex in C_k with respect to C_{k-1}. Both C_k and C_{k+1} contain subjects (namely, the link vertices). Hence, by corollary 4.35, c_k can acquire α rights over q. Applying the induction hypothesis gives us the desired result. ■

Given lemmas 4.31, 4.33, and 4.40, we can characterize sharing completely. If conditions C4.59, C4.60, C4.61, and C4.62 hold, by lemma 4.31, Vs' can get α rights to q; by lemma 4.40, p' can get α rights to q from s'; and by lemma 4.33, p can be given α rights to q by p'. Hence, we define the predicate

DEFINITION D4.25: The predicate $\text{can•share}_{mm}(\alpha, p, q, G_0)$ is true if and only if there is a finite sequence of the four mm graph rewriting rules which, when applied to the protection graph G_0, result in an edge labelled α going from p to q.

The necessary and sufficient conditions for this predicate to be true are:

THEOREM 4.41: The predicate $\text{can•share}_{mm}(\alpha, p, q, G_0)$ is true if and only if all four conditions C4.59, C4.60, C4.61, and C4.62 hold simultaneously.

PROOF: ($\Rightarrow$) Consider what happens if any one of the conditions fails. If C4.59 fails, no vertex can obtain α rights to $\mathbf{q}$, since none of the graph rewriting rules add new incoming rights to a vertex; if C4.60 fails, no vertex in any class C_m can obtain α rights to $\mathbf{q}$; if C4.61 fails, no vertex can grant_{mm} $\mathbf{p}$ α rights to $\mathbf{q}$; and if C4.62 fails, no vertex which can grant_{mm} $\mathbf{p}$ rights will be able to obtain α rights to $\mathbf{q}$. Hence, all four conditions must hold simultaneously for $\text{can•share}_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ to be true.

($\Leftarrow$) Immediate from lemmas 4.31, 4.33, and 4.40. ■

Now that we have characterized sharing, let us think about stealing rights. Informally, we want the predicate $\text{can•steal}_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ to be true whenever $\mathbf{p}$ can obtain α rights to $\mathbf{q}$ without the owner of that right granting it. However, suppose an owner grants to a vertex $\mathbf{p}$ the right to α a vertex $\mathbf{q}''$ in the same class as $\mathbf{q}$. It is reasonable to bar this from taking place in a theft, because a vertex granting rights grants rights to a class, not to a vertex. So, if such a grant takes place, we shall not consider the action a theft.

More formally, we define the predicate

DEFINITION D4.26: Let $\mathbf{p}$ be a vertex and let $\mathbf{q}$ be a vertex in class $\mathbf{Q}$. The predicate $\text{can•steal}_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true if and only if all the following hold:

C4.63. there is no edge labelled α from $\mathbf{p}$ to $\mathbf{q}$ in G_0;

C4.64. there is a sequence $G_1, \ldots, G_n$ of protection graphs and $\rho_1, \ldots, \rho_n$ of rule applications such that

a. $G_0 \vdash_{\rho_1}, \ldots, \vdash_{\rho_n} G_n$,

b. there is an edge labelled α from $\mathbf{p}$ to $\mathbf{q}$ in G_n, and

c. if $\mathbf{s}$ has α rights over any $\mathbf{q}' \in \mathbf{Q}$ in G_0, no ρ_j has the form

$$\mathbf{s}\ \text{grants}_{mm}\ (\alpha\ \text{to}\ \mathbf{q}')\ \text{to}\ \mathbf{x}_j$$

for any vertex $x_j \in G_{i-1}, 1 \leq i,j \leq n$.

Clearly, part C4.64.c of the definition is the key part. We can modify the conditions necessary and sufficient for can•share$_{mm}$ to be true to give conditions both necessary and sufficient for can•steal$_{mm}$ to be true:

THEOREM 4.42: Let **p** and **q** be vertices in a protection graph G_0. Then can•steal$_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true if and only if the following conditions all hold:

C4.65. there is no edge labelled α from **p** to **q** in G_0,

C4.66. there is a vertex **s** with an α edge to a vertex **q'** in the same class as **q**,

C4.67. there is a subject vertex **p'** for which **p'** = **p** or **p'** initially$_{mm}$ spans to a vertex in the same class as **p**, and

C4.68. can•share$_{mm}(t, \mathbf{p}', \mathbf{s}, G_0)$ is true.

INFORMAL ARGUMENT: Going from the conditions to the predicate involves finding a witness that satisfies part C4.64.c of the definition. Going the other way involves applying the definition to obtain the first three conditions, and proving the fourth by checking various cases.

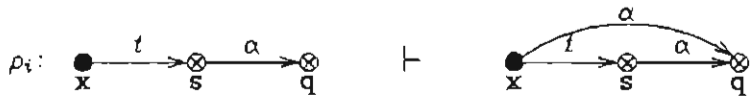
PROOF: ($\Rightarrow$) Assume can•steal$_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true. By part C4.63 of definition D4.26, condition C4.65 holds. By part C4.64.b of the definition, can•share$_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$ is true, so condition C4.61 of theorem 4.41 gives condition C4.67 of this theorem, and condition C4.59 of theorem 4.41 gives condition C4.66 of this theorem.

To prove condition C4.68 holds, consider the minimal length derivation sequence $G_0 \vdash_{\rho_1} \ldots \vdash_{\rho_n} G_n$ witnessing can•share$_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$. Let i be the index for which G_i is the first graph in the derivation sequence where an edge labelled α to q' is added; that is, i is the first index for which $G_{i-1} \vdash_{\rho_i} G_i$, there

is an edge labelled α from some vertex $\mathbf{x}$, in a different class than $\mathbf{s}$, to $\mathbf{q}'$ in G_i, but there is no edge labelled α from $\mathbf{x}$ to $\mathbf{q}'$ in G_{i-1}. Clearly, ρ_i is not a remove$_{mm}$ rule, and since $\mathbf{x}$ and $\mathbf{s}$ are in different classes, ρ_i is not a create$_{mm}$ rule. By part C4.64.c of the definition, it cannot be a grant$_{mm}$ rule, since by our choice of i, all vertices with α rights to $\mathbf{q}'$ in G_{i-1} are also in G_0. Hence, ρ_i must be a take$_{mm}$ rule of the form

$$\mathbf{x} \text{ takes}_{mm} (\alpha \text{ to } \mathbf{q}') \text{ from } \mathbf{s}.$$

Pictorially, this rule looks like:



By C4.60, there is a subject vertex $\mathbf{s}'$ such that $\mathbf{s}' = \mathbf{s}P$ or $\mathbf{s}'$ terminally$_{mm}$ spans to $\mathbf{s}''$ in the same class as $\mathbf{s}$. By C4.61, there are classes $\mathbf{C}_1, \ldots, \mathbf{C}_m$ such that $\mathbf{p}' \in \mathbf{C}_1$ and $\mathbf{s}' \in \mathbf{C}_m$.

Suppose $\mathbf{s}$ is an object. As $\mathbf{s}'$ and $\mathbf{x}$ are in different classes, then our derivation sequence would not be minimal because ρ_i could be omitted entirely; hence, $\mathbf{s}'$ is in the same class as $\mathbf{x}$, and terminally$_{mm}$ spans to $\mathbf{x}$; thus, take $\mathbf{s}' = \mathbf{x}$. In this case, can•share$_{mm}(t, \mathbf{p}', \mathbf{s}, G_0)$ holds.

Now suppose $\mathbf{s}$ is a subject; then $\mathbf{s}'' = \mathbf{s}' = \mathbf{s}$. The vertex $\mathbf{x}$ is in $\mathbf{C}_{n-1}$, for if not, ρ_i could be omitted, whence the derivation was not of minimal length.) If $\mathbf{x} \in G_0$, all of C4.59, C4.60, C4.61, and C4.62 are satisfied, and we are done. So, let $\mathbf{x} \notin G_0$. As $\mathbf{s} \in G_0$ and new labels on incoming edges cannot be added to extant classes, there must be a subject $\mathbf{y}$ in $\mathbf{C}_{n-1}$ for which can•share$_{mm}(t, \mathbf{y}, \mathbf{s}^*, G_0)$ is true, where $\mathbf{s}^* \in G_0$ is in the same class as $\mathbf{s}$. But then take $\mathbf{x} = \mathbf{y}$ and $\mathbf{s}^* = \mathbf{s}$; this again establishes the conditions for can•know$_{mm}(t, \mathbf{p}', \mathbf{s}, G_0)$, so we are done.

($\Rightarrow$) Let the four conditions in the theorem hold. By condition C4.65, part C4.63 of definition D4.26 holds. Let $G_0 \vdash_{\rho_1} \ldots, \vdash_{\rho_{n_1}} G_{n_1}$ be a witness to can•share$_{mm}(t, \mathbf{p}', \mathbf{s}, G_0)$; by C4.68, such a witness exists. Let ρ_{n_1+1} be the rule application

$$\mathbf{p}' \text{ takes}_{mm} (\alpha \text{ to } \mathbf{q}') \text{ from } \mathbf{s}$$

and let $G_{n_1+1} \vdash_{\rho_{n_1+2}} \ldots, \vdash_{\rho_n} G_n$ be a witness to can•share$_{mm}(t, \mathbf{p}, \mathbf{q}, G_0)$; such a rule may be used by conditions C4.66 and C4.68, and by condition C4.67 the witness exists. Then $G_0 \vdash_{\rho_1} \ldots, \vdash_{\rho_n} G_n$ is a witness satisfying part C4.64.c of the definition. Thus, can•steal$_{mm}(\alpha, \mathbf{p}, \mathbf{q}, G_0)$, and the theorem is proven. ■

Thus far, we have dealt with the transfer of rights in a protection graph. Let us now consider the transfer of information. As in the original model, we shall define predicates analogous to can•know and can•snoop to test for information flow.

First, let us define conditions under which a vertex can grant$_{mm}$, or take$_{mm}$, read and write rights. Recall that an *rw*-initial path is a path along which information can be transmitted. We define an *rw*-initial$_{mm}$ path to be the same under the mm rules:

DEFINITION D4.27: A vertex **p** *rw-initially$_{mm}$ spans* to **q** if there is a vertex **s** with a write edge to another vertex q' in the same class as **q**, and **p** terminally$_{mm}$ spans to **s**, or **p** = **s**.

To see that this definition is what we want, we now show:

LEMMA 4.43: Let the following condition be true:

C4.69. there is a subject **p'** which *rw*-initially$_{mm}$ spans to **p**, or **p'** = **p**.

Then **p'** can acquire *w* rights to **p**.

PROOF: If **p'** ≠ **s**, as **p'** terminally$_{mm}$ spans to **s**, **p'** can acquire *w* rights to **q"** by lemma 4.31. If **p'** = **s**, it already has those rights. As **q'** and **p** are in the same class, and **p'** is a subject, by lemma 4.30, **p'** can acquire *w* rights over **p**. ■

Also, recall that an *rw*-terminal span is a path along which information can be obtained. Again, we define an *rw*-terminal$_{mm}$ span to be the same under the mm rules:

DEFINITION D4.28: A vertex **p** *rw-terminally*$_{mm}$ *spans* to **q** if there is a vertex **s** with a read edge to another vertex *q'* in the same class as **q**, and **p** terminally$_{mm}$ spans to **s**, or **p** = **s**.

Once again, we must prove this definition meets the intuitive requirement:

LEMMA 4.44: Let the following condition hold:

C4.70. there is a subject **q'** which *rw*-terminally$_{mm}$ spans to **q**, or **q'** = **q**.

Then **q'** can acquire read rights over **q**.

PROOF: If **q'** ≠ **s**, as **q'** terminally$_{mm}$ spans to **s**, **q'** can acquire read rights to a vertex **q"** in the same class as **q**, by lemma 4.31. If **q'** = **s**, it has such rights to **q"**. As **q"** and **q** are in the same class, and **q'** is a subject, by lemma 4.30, **q'** can acquire read rights to **q**. ■

With these two definitions, we can develop an analogue to "connection" using the mm rules. Informally, a join$_{mm}$ is a path along which information, but not rights, may flow. More formally,

DEFINITION D4.29: Let **p** and **q** be subject vertices. If **p** *rw*-initially$_{mm}$ spans to **q**, or **q** *rw*-terminally$_{mm}$ spans to **p**, or **p** *rw*-initially$_{mm}$ spans to a vertex to which **q** *rw*-terminally$_{mm}$ spans, then **q** is said to be *joined*$_{mm}$ to **p**, and the path between them is called a *join*$_{mm}$.

As usual, we check that this meets the intuitive requirements:

LEMMA 4.45: If **q** is joined$_{mm}$ to **p**, then one of the following cases holds:

C4.71. **p** can obtain an explicit write edge to **q**;

C4.72. **q** can obtain an explicit read edge to **p**; or,

C4.73. **q** can obtain an implicit read edge to **p**.

PROOF: If **p** *rw*-initially$_{mm}$ spans to **q**, by lemma 4.43, case C4.71 holds; if **q** *rw*-terminally$_{mm}$ spans to **p**, by lemma 4.44, case C4.72 holds. So, suppose **p** *rw*-initially$_{mm}$ spans to a vertex **x**, and **q** spans to **x**. By lemma 4.33, **p** can acquire write rights over **x**, and by lemma 4.31 **q** can acquire read rights over **x**; then case C4.73 holds with an application of the post rule. ■

As a result, we may prove a stronger lemma, namely:

LEMMA 4.46: Let the following condition hold:

C4.74. There is a sequence of classes $\mathbf{D}_1, \ldots, \mathbf{D}_m$ such that there are subjects $\mathbf{p}' \in \mathbf{D}_1$, $\mathbf{q}' \in \mathbf{D}_m$, and either $\mathbf{D}_i$ is linked$_{mm}$ to $\mathbf{D}_{i+1}$, or there are vertices $\mathbf{d}_i \in \mathbf{D}_i$, $\mathbf{d}_{i+1} \in \mathbf{D}_{i+1}$, which are joined$_{mm}$, for $1 \leq i < m$.

Then one of three cases holds:

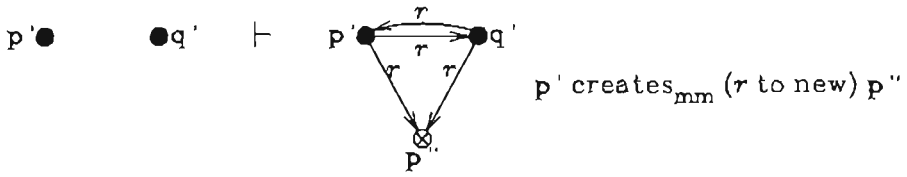
C4.75. **q**' can obtain an explicit write edge to **p**';

C4.76. **p**' can obtain an explicit read edge to **q**';

C4.77. $\mathbf{p}'$ can obtain an implicit read edge to $\mathbf{q}'$;

PROOF: The proof is by induction on the number m of classes.

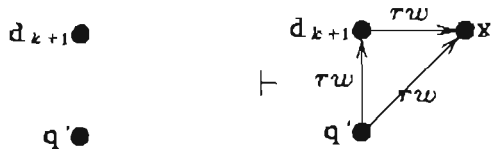
BASIS: $m = 1$. Then $\mathbf{p}'$ and $\mathbf{q}'$ are in the same class $\mathbf{D}_1$; so,



and case C4.76 holds.

INDUCTION HYPOTHESIS: The claim holds for $m = 1, \ldots, k$.

INDUCTION STEP: Let $m = k + 1$. If $\mathbf{d}_k$ and $\mathbf{d}_{k+1}$ are joined$_{mm}$, then we have by lemma 4.45 that one of the three cases holds for $\mathbf{d}_k$ and $\mathbf{d}_{k+1}$. Noting that $\mathbf{d}_{k+1}$ is a subject, we have



and hence one of the three cases applies to $\mathbf{d}_k$ and $\mathbf{q}'$. By the induction hypothesis, one of the three cases also applies to $\mathbf{d}_k$ and $\mathbf{p}'$, so using the appropriate *de facto* rule

Now, if $\mathbf{D}_k$ and $\mathbf{D}_{k+1}$ are linked$_{mm}$, consider the link$_{mm}$ vertices $\mathbf{d}'_k$ and $\mathbf{d}'_{k+1}$. By the construction above, $\mathbf{d}'_{k+1}$ can acquire read rights over $\mathbf{q}'$, whence by corollary 4.35, $\mathbf{d}'_k$ can also obtain those rights. By hypothesis, one of the three cases applies between $\mathbf{d}'_k$ and $\mathbf{p}'$, so by applying either the spy or pass *de facto* rule, the claim follows. ■

As with the om rules, notice that case C4.75 or C4.76 will be possible only when *all* of the classes $\mathbf{D}_1, \ldots, \mathbf{D}_m$ are connected by links$_{mm}$, as joins$_{mm}$ only

transfer information, not rights, only implicit read edges can occur among classes which are joined$_{mm}$.

With these terms we can now define the sharing of information:

DEFINITION D4.30: The predicate can•know$_{mm}$(**p**, **q**, G_0) is true if and only if there is a finite sequence of mm *de jure* and *de facto* rule applications resulting in an explicit write edge from **q** to **p**, or an (explicit or implicit) read edge from **p** to **q**.

In other words, if **p** can obtain information from **q**, can•know$_{mm}$(**p**, **q**, G_0) is true. We can also state necessary and sufficient conditions for can•know$_{mm}$ to hold:

THEOREM 4.47: Let **p** and **q** be vertices in a protection graph G_0. Then can•know$_{mm}$(**p**, **q**, G_0) is true if and only if at least one of the following holds:

C4.78. can•share$_{mm}$(w, **q**, **p**, G_0) is true; or,

C4.79. can•share$_{mm}$(r, **p**, **q**, G_0) is true; or,

C4.80. *all* of C4.69, C4.70, and C4.74 hold simultaneously.

INFORMAL ARGUMENT: Assume exactly one of the three conditions holds. By definition, if either condition C4.78 or C4.79 holds, can•know$_{mm}$(**p**, **q**, G_0) is true. So, assume only condition C4.80 holds. By lemmas 4.43, 4.44, and 4.46, **p** can obtain information from **q**. Going the other way, again if condition C4.78 or C4.79 is true, the result is immediate; if only condition C4.80 holds, we will show that if one of C4.69, C4.70, and C4.74 is false, can•know$_{mm}$ is also false.

PROOF: (⇒) If condition C4.78 or condition C4.79 holds, can•know$_{mm}$(**p**, **q**, G_0) is true by definition. So, assume both are false and condition C4.80 is true. By lemma 4.43, it suffices to have $\mathbf{p}' \in \mathbf{D}_1$ acquire information from **q**; by lemma 4.44, it suffices to have $\mathbf{q}' \in \mathbf{D}_m$ transmit information to $\mathbf{p}'$; and by lemma 4.46,

this can be done. Hence, can•know$_{mm}$(p, q, G_0) is true.

($\Leftarrow$) Assume can•know$_{mm}$(p, q, G_0) is true. Then, if there is an explicit read edge from p to q, condition C4.79 holds; if there is an explicit write edge from q to p, condition C4.78 holds; so, assume there is an implicit read edge from p to q. If C4.69 fails, the implicit edge cannot originate at p; if C4.70 fails, the implicit edge cannot terminate at q; if C4.74 fails, no such implicit edge can exist. In any case, we have a contradiction, so all three conditions C4.69, C4.70, and C4.74 must hold simultaneously. ■

Now that we have established conditions for the sharing of information, let us consider the theft of information. To do this, we define a predicate analogous to can•snoop, namely

DEFINITION D4.31: Let p and q be vertices in a protection graph G_0. Then can•snoop$_{mm}$(p, q, G_0) is true if, and only if,

C4.81. can•steal$_{mm}$(r, p, q , G_0) is true; or,

C4.82. there exists a sequence of protection graphs and rule applications $G_0 \vdash_{\rho_1} , \ldots , \vdash_{\rho_n} G_n$ for which all of the following hold:

a. there is no edge from p to q labelled r in G_0;

b. there is an edge from p to q labelled r in G_n; and

c. neither q, any vertex directly connected to q, or any vertex in the same class as q is an actor in a grant$_{mm}$ or create$_{mm}$ rule or a *de facto* rule application resulting in an (explicit or implicit) edge labelled r going to q.

Just as we derived necessary and sufficient conditions for can•steal$_{mm}$, we can do so for can•snoop$_{mm}$:

THEOREM 4.48: For distinct vertices **p** and **q** in a protection graph G_0, can•snoop$_{mm}$(**p**, **q**, G_0) is true if and only if one of the following is true:

C4.83. can•steal$_{mm}$(r, **p**, **q**, G_0) is true, or

C4.84. all of the following hold:

a. there is no edge from **p** to **q** labelled r in G_0;

b. there is a subject vertex **p**' such that **p** = **p**' or **p**' rw-initially$_{mm}$ spans to **p**;

c. there is a vertex **q**' not in the same class as **q** or that of any vertex directly connected to **q** such that **q**' rw-terminally$_{mm}$ spans to **q**; and

d. can•know$_{mm}$(**p**', **q**', G_0) is true.

PROOF: ($\Rightarrow$) Let can•snoop$_{mm}$(**p**, **q**, G_0) be true. If can•steal$_{mm}$(r, **p**, **q**, G_0) holds, we are done, since part C4.81 of the definition is part C4.83 of this theorem. So, assume can•steal$_{mm}$(r, **p**, **q**, G_0) is false.

Part C4.82a of definition D4.31 gives condition C4.84a of this theorem.

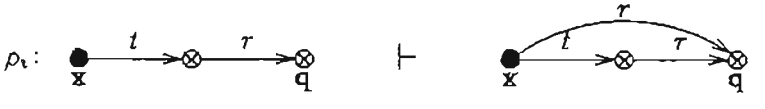
By part C4.82b of the definition, there is an implicit read edge from **p** to **q** in G_n, whence by definition can•snoop$_{mm}$(**p**, **q**, G_0) is true; so, condition C4.84a of this theorem results from condition C4.69 of theorem 4.47.

By condition C4.70 of theorem 4.47, there is a subject **q**' such that **q**' = **q** or **q**' rw-terminally spans to **q**. If **q** is an object, take **q**' to be the q' in condition C4.84c of this theorem.

If **q** is a subject, by part C4.82c of the definition of can•snoop$_{mm}$, it is not used in the sequence of rule applications witnessing can•snoop$_{mm}$. In this case, choose **q**' in condition C4.84c to be this **q**'. If **q**' and **q** are directly connected (or any vertex in **q**''s class is directly connected to any vertex in **q**'s class, for that matter) by a read edge in G_0, then either can•share$_{mm}(t, \mathbf{p}', \mathbf{q}', G_0)$ is true [in which case can•steal$_{mm}(t, \mathbf{p}, \mathbf{q}, G_0)$ is true, contradicting assumption] or **q**' must actively participate in a grant$_{mm}$, create$_{mm}$, or a *de facto* rule application [which contradicts part C4.82c of the definition of can•snoop$_{mm}$, contradicting assumption again]. In either case, there is no edge labelled r from any vertex in **q**'s class to a vertex in **q**''s class in G_0.

It remains to be shown that can•know$_{mm}(\mathbf{p}', \mathbf{q}', G_0)$ is true. Let $G_0 \vdash_{\rho_1} \ldots \vdash_{\rho_n} G_n$ be a minimum length derivation sequence, and let i be the first index such that the first (explicit or implicit) read edge with target vertex in **q**'s class is added in G_i. That is, i is the least index such that $G_{i-1} \vdash_{\rho_i} G_i$, there is no (explicit or implicit) read edge from any vertex **x** to **q**'' in the same class as **q** in G_{i-1}, and there is an (explicit or implicit) read edge from **x** to **q**'' in G_i. Consider what rule ρ_i was used to add this edge. It cannot be a grant$_{mm}$ or a create$_{mm}$ rule because that would violate part C4.82c of definition D4.31. Nor can it be a spy, pass, or find rule, or a post rule, for this would violate the same part of the definition. As a remove$_{mm}$ rule application deletes rights to existing classes, ρ_i could not be that, either. Hence, ρ_i must be a take$_{mm}$ rule.

We therefore have:



Recalling that can•know$_{mm}$(p, q, G_0) is true, by theorem 4.47 we see can•know$_{mm}$(p', q, G_0) is true. Apply theorem 4.47 again; by this theorem, there is a subject vertex q' such that q' = q or q' rw-terminally$_{mm}$ spans to q. Noting that there is no edge from q' to q labelled r in G_0, we take q' = x in theorem 4.47, whence can•know$_{mm}$(p', q', G_0) immediately follows.

(⇐) If condition C4.83 of the theorem holds, by part C4.81 of the definition, can•snoop$_{mm}$(p, q, G_0) is true.

So, assume condition C4.84 holds. Part C4.82a of the definition is the same as condition C4.84a of the theorem. By theorem 4.47, conditions C4.84b, C4.84c, and C4.84d establish part C4.82b of the definition. And conditions C4.84c and C4.84d of this theorem lead immediately to part C4.82c of the definition.

This completes the proof. ■

IV.5. Discussion

At this point, let us take stock of what we have done. We have proposed three extensions, each of which incorporates a notion of "class", to the Take–Grant Protection Model. Although all three of these extensions are similar, there are enough differences to make the derivations for each extension interesting.

The technique used emphasized the similarities among the three extensions; in fact, the theorems proving what conditions are necessary and sufficient for can•know$_{om}$, can•know$_{mo}$, and can•know$_{mm}$ to be true are virtually identical. Of course, since the terms in the statement of the theorem change meaning from extension to extension, one cannot say that the proof for all is therefore obvious. An alternate development (considered but rejected as

too confusing) would have been to derive the three extensions simultaneously. And not all the predicates have the same necessary and sufficient conditions – the theorems for the various can-snoop predicates do vary substantially.

Having derived these three extensions, let us now proceed to consider their practical applications.

Chapter V

Applications of the Model and its Extensions

V.1. Introduction

In this chapter, we shall examine practical aspects of computer security using the theory developed in the preceding chapter. We shall first define a reference monitor in terms of our predicates, and then we shall describe in depth some security flaws in the UNIX operating system (including one class of problems, as well as a few specific ones) which involve violations of protection. In order to do these things, however, we must define a *security breach* in terms of the extensions previously described, and determine which extension or extensions are to be used. Next, we shall abstract the relevant features of UNIX into a suitable representation, and from this exhibit the security breaches, as well as suitable remedies.

As the Take Grant model deals only with authority and information transfer in a protection graph, many known security holes cannot be found using the form of the model being discussed here; an example is the holes resulting from the failure of a command interpreter to check its input properly. It is not the purpose of this chapter to demonstrate how this extension to the Take Grant model may be used to cover all such situations; the goal of this

section is simply to show that a Take Grant model can be constructed to model an existing system and locate problems in the protection schema which allow breaches of security involving illicit transfers of authority or flows of information.

As the above paragraph implies, modelling is not done at the program statement level; for every statement executed, a new protection graph is not created. Rather, a new protection graph is created whenever the authorities recorded in the protection controls of the system change. This way, instructions not affecting the protection schema are not dealt with.

V.2. What is a "Security Breach"?

Before we discuss security breaches, we must describe security in terms of the Take Grant model and its extensions. All four versions discussed in this paper use the same basic definitions, so one definition for all four versions is possible. Of course, the definition is written in terms of predicates; the specific requirements are then derived from the appropriate theorems. We will use the notation "pred$_*$" to mean the appropriate predicate; for example, "can•know$_*$" may mean any of "can•know", "can•know$_{am}$", "can•know$_{mo}$", or "can•know$_{mm}$", depending on which extension is being used to model the situation under consideration.

The first aspect of breaches of security is the illicit transfer of authority. Initially, the protection graph has a specific state. Sharing of rights should be permitted whenever authorized by the initial state, however, theft of rights should be barred. This way, if a process does not want to share its rights with another, the second cannot take those rights. In other words, a right may only be obtained with its owner's consent. This leads to the condition:

$$\neg can\cdot steal\cdot(\alpha, \mathbf{x}, \mathbf{y}, G_0) \quad \forall\ \alpha \subset R,\ \mathbf{x}, \mathbf{y} \in V,\ \mathbf{x} \neq \mathbf{y} \qquad \text{(C5.1)}$$

Dealing with information flow is a bit more complex, as a result of the *de facto* rules we use. These rules require the active cooperation of all subjects involved, and with one exception (the post rule) each requires two subjects at least one of which has access to the information being transferred.

In this case, the question of what constitutes a "breach of security" depends on the intent of the subject with the rights to the information. To take an example, suppose it is possible for one process to read another process' memory. There are two processes, A and B. B has access to a file to which A has no rights. If B reads the confidential file, and A reads the memory of B at the same time (thereby seeing the contents of the confidential file), has there been a breach of security?

If B's intent in reading the file was to make it available to A, then there has been no breach of security. All B has done is (indirectly) copied the confidential file into another file and changed the protections so A could read it – certainly allowed as B can read the confidential file. But if B's intent in reading the file was simply to see what it contained without passing any of its contents on to A, a breach of security has certainly occurred, since A has read a file which it had no authority to read. But note that A has not violated any restrictions imposed upon it by the system!

It is reasonable to hold either opinion. Due to the personal preference of the author, we shall use the latter, because we are not examining the adequacy of the protection system, but just looking at ways to evade it. Hence, we use the restriction

$$can\cdot know\cdot(\mathbf{x}, \mathbf{y}, G_0) \Rightarrow can\cdot share\cdot(r, \mathbf{x}, \mathbf{y}, G_0) \quad \forall\ \mathbf{x}, \mathbf{y} \in V,\ \mathbf{x} \neq \mathbf{y} \qquad \text{(C5.2)}$$

(If we had decided that no breach of security were involved, this condition would have become:

$$\neg can\text{-}snoop(x, y, G_0) \quad x, y \in V,\ x \neq y$$

rather than involving can•snoop.)

To summarize, we shall determine that something is a security hole by seeing if it violates the above *security principle*, namely:

$$can\text{-}steal(\alpha, x, y, G_0) \quad \forall\ \alpha \in R,\ x, y \in V,\ x \neq y \qquad (C5.1)$$

and

$$can\text{-}know(x, y, G_0) \Rightarrow can\text{-}share(r, x, y, G_0) \quad \forall x, y \in V,\ x \neq y \qquad (C5.2)$$

V.3. Reference Monitors

The concept of a *reference monitor* was first described in [ANDE72] as a subject or process meeting three requirements:

C5.3. the process must be *isolated* (that is, tamper-proof);

C5.4. the process must be *complete* (that is, always invoked when the resources it controls is accessed);

C5.5. the process must be *verifiable* (that is, small enough to be subject to analyses and tests the completeness of which can be ensured).

In this section, we shall examine reference monitors from the point of view of the Take-Grant model and its extensions, in order to demonstrate that the theory done in the preceding two chapters may be applied to very general practical aspects of computer security.

Restating conditions C5.3 – C5.5 in terms of our model is quite straightforward. Let the reference monitor be called **m** and let the resource it protects be called **r**. The issue of isolation simply means that no-one can write over the monitor: this may be stated as $\neg can\text{-}share(\{w\}, x, m, G_0)$ for all $x \in V$. The issue of completeness may be restated as requiring that **m** always be invoked to give a subject some rights α over **r**, which becomes the requirement that, for all vertices $x \in V$, only **m** have an edge labelled α to **r**, and for all vertices $x \in V$ such that $x \neq m$, $\neg can\text{-}steal(\alpha, x, r, G_0)$. The issue of verifiability is a bit more tricky, since it consists of two parts. The first, an implementation-level task of verifying that the monitor is indeed implemented correctly, is beyond the scope of this thesis, as stated in section V.1; however, the second, which is a verification that no information will leak when the monitor is implemented correctly, is simply the security property (that is, conditions C5.1 and C5.2.)

V.4. Modelling UNIX

UNIX is a very popular operating system, one which grew from a research system; for this reason, security has not been an important consideration during its development. Nothing that follows should be taken as a denigration of UNIX's merits as an operating system, or the beauty of its philosophy or design; it is the best system the author has ever used. It was designed and developed in a friendly environment, where security is not a major consideration; hence, the security holes which we will discuss should be seen as completely accidental by-products of the design.

We shall look at a specific hole, and at a class of programs which are intended to evade the protection mechanisms. It is important to emphasize that the security holes discussed here are widely known, and on all systems

where penetration is a threat have been (or should have been) dealt with. No dangerous methods of violating system security will be described, and fixes to the methods touched upon here will be presented.

Before modelling those parts of UNIX we need to, we must decide how to abstract the protection schems of UNIX into the model. First, which extension or extensions do we use?

A few words about the protection scheme used by UNIX are in order. UNIX controls access to a file by nine bits, the first three representing *owner* rights, the next three *group* rights, and the last three the *world* rights (that is, the rights of everyone else.) In each set of three bits, the first one controls *read* access, the second *write* access (which includes appending to the file, too), and the third *execute* access for files or *search* access for directories. When a process starts, it has associated with it a unique identifier called the *PID*, a user identification code called a *UID*, and a set of group identification codes called *GIDs*: as indicated above, associated with each file is a unique identifier called an *inode*, an *owner* and a *group*. The algorithm for determining in what manner a process can access a file is:

```
(*
 * returns TRUE if process P has α rights
 * over file F, FALSE otherwise
 *)
function cando(α : set of rights; F : file; P : process) : boolean;

begin
  if ( F . owner = P . uid ) then
    cando := ( ( α ∩ F . owner_rights ) = α )
  else if ( F . group in P . gid ) then
    cando := ( ( α ∩ F . group_rights ) = α )
  else
    cando := ( ( α ∩ F . world_rights ) = α );
end (* cando *);
```

This is the part of UNIX we shall be modelling.

The above algorithm shows something very important; whenever a process is created, it gains access to all files with the owner corresponding to the process' UID as indicated by the owner part of each file's protection mask, access to *all* files with the group corresponding to the process' GID as indicated by the group part of each file's protection mask, and access to *all* other files as indicated by the world part of each file's protection mask. If a process spawns a new process, the new one has the same UID and GID as its parent, so it will have the same accesses as the parent process. Thus, the one-many version of the model is the proper one to use for interactions involving processes and files. The relevant classes will be classes corresponding to the possible UIDs and GIDs of processes. These classes will be distinguished from each other to avoid confusion.

Explicitly, we represent processes as a triple

$$(\text{PID} , \{ \text{UID} \} , \{ \text{GID}_1 , \ldots , \text{GID}_n \})$$

and a file as a sextuple

(inode , { owner } , { group } , owner_rights , group_rights , world_rights)

The algorithm for determining if an explicit edge goes from a process P to a file F is:

```
(*
 * return the rights in the bit mask M as a set of rights
 *)
function labels(M : bitmask) : set of rights;

begin
   labels := φ;
   if ( M bitand 0100 ) then labels := labels + [ r ];
   if ( M bitand 0010 ) then labels := labels + [ w ];
   if ( M bitand 0001 ) then labels := labels + [ x ];
end (* labels *);
```

```
(*
 * determine the labels of the explicit edge from a process P
 * to a file F; φ means no such edge is present
 *)
function explicit(P : process; F : file) : set of rights;

begin
  if ( F . owner = P . uid ) then
    explicit := labels ( F . owner_rights )
  else if ( F . group in P . gid ) then
    explicit := labels ( F . group_rights )
  else
    explicit := labels ( F . world_rights );
end (* explicit *);
```

If *explicit* returns the null set, no explicit edge exists; otherwise, the explicit edge exists and has the named labels.

It will also be necessary to allow edges labelled r and w to go from file to file to indicate how information can flow from one file to another. This does not mean that a file is an actor with these rights; as mentioned before, a file is an inert object and so has no rights which it can exercise. Such flow is allowed if the two files have the same UID or the same GID. Note the use of the verb "allowed"; the flow may never take place if processes do not move the information. These edges merely indicate information may move from one file to another. The algorithm for determining if there is an edge labelled rw from one file F to another file G is:

```
(*
 * determine the labels of the explicit edge from a file F
 * to a file G; φ means no such edge is present
 *)
function fexplicit(F, G : file) : set of rights;

begin
  if ( F . owner ∩ G . owner ≠ φ ) then
    explicit := [ r, w ]
  else if ( F . group ∩ G . owner ≠ φ ) then
    explicit := [ r, w ]
  else
    explicit := φ;
end (* fexplicit *);
```

Note that *fexplicit* is symmetrical.

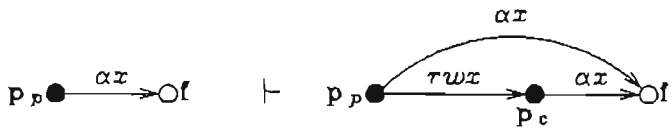
The term *protection state* refers to the explicit edges present and the tuples representing processes and files. If any of these change, or new ones added or deleted, the protection state of the system being modelled has changed.

As for the graph rewriting rules: both the *de jure* and *de facto* rules are the same. there is one new right, that of *execute*. Strictly speaking, this right encompasses two rules, one for directories and one for files; but, in the discussion of security holes below, we will not be dealing with searching directories, so we will just treat "execute" as the right to load and run an executable file. We still have to define the rule, however.

In UNIX, a parent process can communicate with its child via *pipes* (in which case the communication is usually in both directions) or via signals (in which case, while the child communicating with the parent is still possible, it is considerably more complicated.) To capture this, we define a rule for *execute* as follows:

RULE R5.1: execute

Let $\mathbf{p}_p$ be a process and let $\mathbf{f}$ be a file. Suppose $\mathbf{p}_p$ has *execute* rights over $\mathbf{f}$. Then $\mathbf{p}_p$ creates a second process $\mathbf{p}_c$ with $\mathbf{p}_p$ having read and write rights over $\mathbf{p}_c$. In pictures,



No doubt the astute reader will have noticed that this is not really a new rule, but giving a new name to the $create_{mo}$ rule. This is quite true, but the execute rule *explicitly* specifies that the newly created child process $\mathbf{p}_p$ is an executing

image of the contents of the file f. So, it is really a constrained version of the $create_{om}$ rule; in face, so far as processes are concerned, it is the *only* create rule.

This last observation brings up a key point in the application of the model. It may be necessary to mix the different extensions of the theoretical model to capture certain aspects of the system. In UNIX, for example, changes in the protection state (caused by, for example, adding a new process, as in the execute rule) affect all processes in that class. For example, in the situation discussed for the execute rule above, all processes with the same UID as p_p have read and write rights over p_c (albeit crudely; one would need to use signals rather than pipes for all communication except between p_p and p_c.) Hence UNIX interactions between processes should be modelled by the many-one extension to the Take Grant model (which should not be a surprise, considering how close the execute and $create_{mo}$ rules are.)

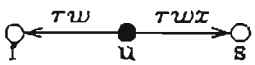
In short, when modelling UNIX, we use the many-one extension to model process–process interactions, and the one-many extension to model process–file interactions. Obviously, which of these extensions modelling any given computer system will require (if, indeed, any of them apply) will depend entirely on the nature of the system. One must choose the extension or version of the model that best captures the real situation. In this there is some science, but far more art.

V.5. Files Writeable by Anybody

This is a security hole which, most of the time, poses no threat to the system; on those rare occasions when it does, however, the effects can be devastating. The loophole here is simply to allow anyone to write on a file, whether or

not that person is the owner, or in the group corresponding to the file's GID. If the file involved is a directory, such as "/tmp", there will most likely be no danger (although one could easily dream up ways in which it would be dangerous); but if the file contains a system program, the likelihood is just the opposite.

The security hole, in this model, may be described as follows. Let a process **u** have UID u_1 and the set of GIDs $G = \{g_1, \ldots, g_n\}$. Let there be two files, the first **s**, with **s**.*owner* $\neq u_1$, s.group $\notin G$, and **u** having w rights over **s**; the second **f**, with **u** having r rights over **f**, and **s**.*owner* $\neq$ **f**.*owner* and **s**.*group* $\neq$ **f**.*group*. Pictorially, we have this situation:



Note here that the UID of process **u** is *not* the same as the UID of file **s**, and **s**'s GID is not in the set of GIDs of **u**. Hence, there is *no* read edge from **f** to **s**. However, by use of the *post* rule, information can flow from **f** to **s**. This clearly violates condition C5.2 of the security principle.

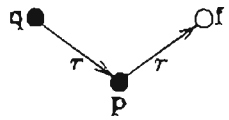
In graphic terms, there are three ways to fix this hole. Deleting the read edge to **f** is not practicable, because **u** can always create a file over which it has read rights. Deleting the write edge to **s** is certainly practicable, and is an adequate solution; thus, no process should be able to write to a file unless its UID corresponds to the file's owner, or the group of the file is in the set of the GIDs of the process. A third, less obvious, fix is to change the owner of the file to that of the process writing to the file. This would result in can•share$_{om}(r, \mathbf{f}, \mathbf{s}, G_0)$ being true, and hence condition C5.2 being satisfied. This also, from the graphical standpoint, is perfectly acceptable.

In practical terms, either the second or the third solution is adequate. Actually, owing to the existance of setUID programs (see section V.7 for a description of these programs), the second is probably overly restrictive; however, this is something which must be decided on a system by system basis.

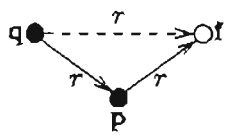
V.6. Processes and Memory

This is an example of a security hole which, most of the time, is not dangerous; but it could be disastrous if it occurred when confidential data was being accessed. It also provides a good example of the use of the security principle discussed earlier.

In UNIX, memory is considered a file (called "/dev/kmem"). Often this file is not protected from nonprivileged processes reading it. Herein lies one security hole. Let process **p** have the right to read a confidential file **f**; let process **q** not have that right. As /dev/kmem is unprotected, **q** has the right to read it, and hence the memory of **p**. When **p** reads **f**, we have in Take Grant terms the following protection state:



Now, when **q** exercises its right to read /dev/kmem and hence **p**, by the spy rule, we have:



Consider the security principle. Since no transfers of authority were made, condition C5.1 still holds. How about information flow? First, can•know$_{om}$(**q**, **f**, G_0) is true by theorem 4.13 (taking **p**' = **q**, **q**' = **p**, and m = 1 in that theorem.) But by 4.7, can•share$_{om}$(r, **q**, **f**, G_0) is false since **p** and **q** are in classes not connected by a bridge$_{om}$. Hence the security principle is violated and we have a security hole.

To fix this hole on the protection graph, we need to make can•know$_{om}$ false. There are two ways to do this: delete the read edge from **p** to **f** (which would prevent both **p** and **q** from reading **f**) or delete the read edge between **p** and **q**. These would make can•know$_{om}$(**q**, **f**, G_0) false, satisfying condition C5.2 of the security principle.

Now, how does this translate into UNIX terms? The first proposal, deleting the r edge from **p** to **q**, would require that **p** be barred from reading **f**. This is not tenable, because it contradicts the assumption that **p** has the right to read **f**. The second, deleting the r edge from **q** to **p**, would require that **q** be barred from reading the memory in which **p** is located. This is very reasonable, and would require only that UNIX memory be read-protected. The best solution, therefore, is to read-protect UNIX memory.

V.7. SetUID Processes

A typical problem in systems programming [BISH83] is often posed as a scorekeeping problem [ALEP71]. Suppose someone has a game program and wants to keep a record of the highest scores. This file, which will be called the *high score file*, must be writable by the game program (so it can be kept up to date), but not by anyone else (so that the entries in it are accurate.) UNIX solves this problem by providing two sets of identifications for processes. The first

set, called the *real* user identification and group identification (or UID and GID, respectively) indicate the real user of the process. The second set, called the *effective* UID and GID, indicates what rights the process has, which may be, and often are, different from the real UID and GID. The protection mask contains a bit which is called the *setUID* bit. (There is another such bit for the effective GID.) If this bit is not set, the effective UID will be that of the person executing the file; but if this bit is set (so that the program runs in *setUID* mode), the effective UID will be that of the file and not of the person (process) executing the file. In either case, the real UID and GID are those of the person (process) executing the file. So, if only the owner of the high score file (who is the user with the same UID as the file) can write on it, the setUID bit of the file containing the game program is turned on, and the UIDs of this file and the high score file are the same, then when someone runs the game program, that process can write into the high score file.

A setUID process, then, gives a user privileges to access information and access files which he would not otherwise have. This clearly violates condition C5.2 of the security principle; information may flow along a non-existent path. As setUID are provided specifically for evading the protection controls, this result is not surprising.

However, this is very unsatisfying. SetUID programs are among the most dangerous programs on UNIX, and it is very reasonable to attempt to provide for them in our model. Unfortunately, this introduces a new factor — time — in the model, since the increased access such programs allow exists for a limited time only, we cannot use any of our earlier rules without modification.

When a setUID process begins, the new rights it provides are added to the protection graph abstracted from the system, and when the setUID process

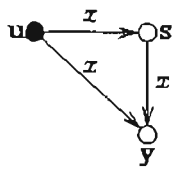
exits, they are removed. This results in a different protection state of the system while the setUID process is executing, a state in which all the authorities allowed by the setUID process are exhibited.

Given that a setUID process is executing, and that the protection graph exhibits those authorities added by the setUID process, let us take another look at the security question. SetUID programs fell outside the protection scheme with the model discussed in the previous sections because information could flow along a path with no corresponding right, or without the owner of the information making it available. But now, the information flows from its source to the setUID process, and thence to the process invoking the setUID program; this flow is authorized on the new protection graph because the setUID program has authority over the information. When it exits, the authority disappears, and the invoking process can no longer access the information. Hence, we add a new condition to the security principle, one which is used only when setUID processes are involved:

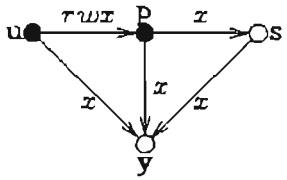
C5.6. No setUID process makes any changes in the protection state of the system except for adding temporary rights which cease to exist when the setUID process exits and that information is allowed to pass through the setUID process as indicated by either added or previously existing rights.

Hence, this manner of treating setUID programs allows us to treat them as ordinary processes once the temporary rights have been added to the protection graph. We must also remember to delete these rights once the process has exited. To distinguish these temporary rights from those rights not associated with a setUID process, we shall label them with a superscripted t, as in "r^t".

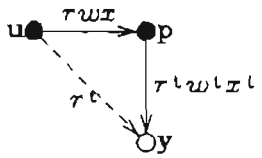
It is time for an example! The example here is that of a setUID process with escapes. Such escapes allow one to execute an arbitrary command from within the setUID program. Let the setUID program be **s**, the corresponding process **p**, the process invoking the setUID program **u**, and the confidential file to be read from the setUID program **y**. Note that in what follows, we are using the many-one extension to model process–process interaction and the one-many extension to model process–file interaction. The sequence of steps is: initially,



Then the setUID process begins:



And, by applying the post rule (and deleting irrelevant edges):



Now, **u** can read **y** as though it were the owner of **s**.

In this case, note that both C5.1 and C5.2 are satisfied; the failure of security is due to the failure of condition C5.6, because the edge from **u** to the confidential file does not pass through the setUID program. The solution is to reset the UID and GIDs of **u** before the process **p** is started. This way, a setUID process is *not* involved in the viewing of the confidential file.

Chapter VI

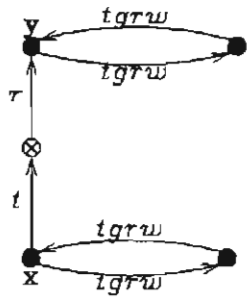
Conclusion

VI.1. Hierarchies

In our discussion of hierarchies, we established conditions under which a hierarchical protection system is secure regardless of how many of its subjects are corrupt. We also found restrictions on *de jure* rules that ensure security without restricting the transfer of rights other than reading or writing.

It is somewhat surprising that the *de facto* rules need not be restricted to ensure security. But *de facto* rules merely indicate graphically how information flows through the graph, whereas the *de jure* rules control the paths along which information can flow. So restricting the *de jure* rules is logical.

Would it be possible to restrict the *de facto* rules to acheive the same results? No, for two reasons. The first is that there are graphs in which one can breach security by using *de jure* rules only, such as



The second is more subtle, and hinges on the distinction between an implicit and an explicit edge.

The *de jure* restriction is not a modification of the *de jure* rules, but only of the instance of application. Unfortunately, such a restriction is meaningless with respect to the *de facto* rules, because the information can still flow; the only restriction is on acknowledging that flow. Implicit edges, added by *de facto* rules, merely indicate the paths along which information can flow, and do not create new paths (or delete old ones). On the other hand, explicit edges, added by *de jure* rules, do create (or remove) paths along which information can flow; hence restricting them does restrict the flow of information.

In the model described in chapter II, the security classification of information cannot be changed without compromising security. No classification can be raised, because anyone with access to the information could have made a private copy; then, after the classification level is raised, he still has access to the information which is now at a higher level. Lowering a classification of some information can also lead to a violation, because a high-level person could declassify confidential information so that someone at a lower level could obtain it. Under the definition of *secure* used here, this is a security violation.

Permitting declassification would raise a host of problems. Suppose a file were declassified. Unless the protection system were to ensure that no user at a level higher than the new level of the file were to have write rights on the file, the system is no longer secure; all one of those users would have to do is to write classified information into that file. Such a protocol would involve assuming a user or classification manager could be trusted — and this would violate an underlying assumption.

This model was developed under the assumption that no user should be able to break security, an assumption not present in earlier Take–Grant models of hierarchical protection systems. The results give constraints on the system which prevent any user, regardless of classification, from passing information to another user with a lower classification. Were it not for this assumption, the protocol for declassification would be obvious; whether or not this assumption is valid, and in what ways it must be weakened to permit declassifications, is still an open question.

Note that when these results are applied to the Take–Grant model of a document system, the total view of security given in [BELL74] is obtained. As the *write* authority in the Take–Grant model is not a viewing right, the *write* authority of the Take–Grant model is the same as the *append* authority of Bell and LaPadula. Then, restriction R2.4a is equivalent to the *refined simple security property*, and restriction R2.4b is the *no write down property*. These restrictions apply to any system that can be modelled as a hierarchy in the Take–Grant model.

The basic difference between this model of hierarchical protection systems and earlier Take–Grant models (cf., [WU80]), is that *de facto* as well as *de jure* rules are used. The representation of security levels makes sharing among vertices in a security level simple. Also, rights from a lower to a higher security level are permitted (subject, of course, to R2.4); this was not the case in earlier models. Both these features make the model very straightforward; even if modified to permit reclassification, this model is far simpler than earlier ones which only used *de jure* rules.

This model is an example of the application of the *de facto* rules developed in [BISH79]. These rules are merely one possible set of *de facto* rules. We do

not claim that these rules are the best ones to use, or model the real world most closely; as their effects have been explored previously, they were most convenient for our purposes. Under different *de facto* rules, the development would be similar; and due to the considerations outlined above, the restrictions obtained would have to act on the *de jure* rules, or the application of the *de jure* rules, rather than the *de facto* rules.

VI.2. Computer Systems

In the rest of this thesis, we extended the Take–Grant Protection Model in such a way that it could be applied to a computer system; we then defined what a secure system was in terms of predicates, and using the theory derived using the relevant rewriting rules, we applied the definition to an existing computer system. A few points must be made about this work.

First, the application here involved examining the protection state rather than analyzing programs line-by-line. While the latter is certainly possible, it is not necessary unless the way in which the programs interact with the operating system is not known. Even in this case, only those parts of programs which affect the protection state would affect the protection graph representing the system, and hence would need to be considered.

The analysis of the computer system in chapter V was by no means complete, nor was it intended to be. Rather, that chapter was meant to demonstrate that the model could be used in practise to model security flaws, and that – as a side benefit – from the modelling at least one method of fixing the flaws becomes apparent. In this, we succeeded.

There are several areas where further work would be profitable.

- The similarity in both theorems and proofs between the three extensions to the Take–Grant Protection Model suggests that it may be possible to prove metatheorems giving necessary and sufficient conditions for the predicates can•share$_e$, can•steal$_e$, can•know$_e$, and can•snoop$_e$ to be true for classes of graph rewriting rules; this would make proving security (or lack of it) much easier, since it would no longer be necessary to rederive the results whenever one changed the graph rewriting rules.
- The complexity of any rigorous checking will be quite large, certainly too much so for a human to cope with. There are two possible approaches to this problem: the first is to subdivide the task among many people, the second to have a computer do the checking. The former will require that very careful checking be done at the places where the subdivisions overlap; the latter requires a rather sophisticated program. Similar programs already exist, such as the theorem prover used to verify the security of PSOS [FEIE79]; would the complexity of such a program for the Take–Grant Protection System be prohibitive?
- This thesis talked about applying the model to existing operating systems. But, very little was said about using the model to prove security of an abstract design for a system. How could such a use be integrated with verifying the correctness of the implementation of the system?
- Finally, and now most importantly, how can we abstract an existing system to a protection graph? This is quite crucial, because without a proper abstraction, all the theory in the universe will not locate security holes in the existing system. With the operating system used here, the abstraction was very simple; methods to handle much

more complex protection schemes would be interesting to learn.

To sum this up – the Take–Grant Protection Model, without extensions, is primarily a theoretical tool. But, when proper extensions are made, it can be a very powerful practical tool as well.

BIBLIOGRAPHY

BIBLIOGRAPHY

ALEP71. Aleph-Null, "Computer Recreations," *Software – Practise and Experience* 1(2) pp. 201-204 (April–June 1971).

ANDE72. Anderson, J. P., "Computer Security Technology Planning Study," Technical Report ESD-TR-73-51, Vols. I and II, USAF Electronic Systems Division, Bedford, MS (October, 1972).

BELL74. Bell, D. and LaPadula, L., "Secure Computer Systems: Mathematical Foundations and Model," Technical Report M74-244, The Mitre Corporation, Bedford, MS (October 1974).

BERS79. Berson, T. A. and Barksdale, G. L., "KSOS – Development Methodology for a Secure Operating System," pp. 365-371 in *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, NJ (1979).

BISH79. Bishop, M. and Snyder, L., "The Transfer of Information and Authority in a Protection System," *Proceedings of the Seventh Symposium on Operating Systems Principles*, pp. 45-54 (December 1979).

BISH81. Bishop, M., "Hierarchical Take–Grant Protection Systems," *Proceedings of the Eighth Symposium on Operating Systems Principles*, pp. 109-122 (December 1981).

BISH83. Bishop, M., "Security Problems with the UNIX Operating System," (unpublished), Purdue University, West Lafayette, IN (January, 1983).

BROA76. Broadbridge, R. and Mekota, J., "Secure Communications Processor Specification," Technical Report ESD-TR-76-351, AD-A055164, Honeywell Information Systems, McLean, VA (June 1976).

DENN76. Denning, D., "A Lattice Model of Secure Information Flow," *CACM* 19(5) pp. 236-243 (May 1976).

DENN82. Denning, D., *Cryptography and Data Security*, Addison-Wesley Publishing Company, Reading, MS (1982).

DIJK68. Dijkstra, E. W., "The Structure of the 'THE'-Multiprogramming System," *CACM* 11(5) pp. 341-346 (May 1968).

ENDE77. Enderton, H., *Elements of Set Theory*, Academic Press, New York, NY (1977).

FEIE79. Feiertag, R. J. and Neumann, P. G., "The Foundations of a Provably Secure Operating System (PSOS)," pp. 329-334 in *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, NJ (1979).

HARR76. Harrison, M. A., Ruzzo, W. L., and Ullman, J. D., "Protection in Operating Systems," *CACM* 19(8) pp. 461-471 (August 1976).

JONE76. Jones, A., Lipton, R., and Snyder, L., "A Linear Time Algorithm for Deciding Security," *Proceedings of the Seventeenth Annual Symposium on the Foundations of Computer Science*, (1976).

JONE78. Jones, A., "Protection Mechanism Models: Their Usefulness," pp. 237-252 in *Foundations of Secure Computing*, ed. Richard Lipton,Academic Press, New York, NY (1978)

LIND75. Linde, R. R., "Operating System Penetration," pp. 361-368 in *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, NJ (1975).

LIPT77. Lipton, R. and Snyder, L., "A Linear Time Algorithm for Deciding Subject Security," *JACM* 24(3) pp. 455-464 (July 1977).

MILL76. Millen, J. K., "Security Kernel Validation in Practice," *CACM* 19(5) pp. 243-250 (May 1976).

POPE79. Popek, G. J., Kampe, M., Kline, C. S., Stoughton, A., Urban, M., and Walton, E., "UCLA Secure Unix," pp. 355-364 in *Proceedings of the National Computer Conference*, AFIPS Press, Montvale, NJ (1979).

SCHI75. Schiller, W. L., "The Design and Specification of a Security Kernel for the PDP-11/45," Technical Report ESD-TR-75-69, The Mitre Corporation, Bedford, MS (March 1975).

SNYD77. Snyder, L., "On the Synthesis and Analysis of Protection Systems," *Proceedings of the Sixth Symposium on Operating Systems Principles*, pp. 141-150 (November 1977).

SNYD81. Snyder, L., "Formal Models of Capability–Based Protection Systems," *IEEE Transactions on Computers* **C-30**(3) pp. 172-181 (March 1981).

WEBS73. *Webster's New Collegiate Dictionary*, G.&C. Merriam Co., Springfield, MS (1973).

WU80. Wu, M., "Hierarchical Protection Systems," Ph.D. Thesis, University of Iowa, Iowa City, IA (December, 1980).

VITA

Matt Bishop was born in New York City on September 29, 1956. He graduated from JR Schools in San Rafael. He accepted a Regents' Scholarship from the University of California at Berkeley, and went there to study Astronomy and Applied Mathematics, graduating with an A. B. with High Honors in Applied Mathematics and Distinction in General Scholarship in 1976. He remained at Berkeley to do graduate work in Mathematics, emerging with an M.A. in 1978. Then, he went to Yale University, where he studied computer science; after one year there, he transferred to Purdue University to continue his studies. At Purdue, he was supported as a teaching assistant, a research assistant, and a student assistant on the computer staff of the Department of Computer Sciences. He recieved an M.S. in computer science in 1981.