# The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes

RAVINDERPAL SINGH SANDHU

*Department of Computer and Information Science, The Ohio State University, Columbus, Ohio*

Abstract. The protection state of a system is defined by the privileges possessed by subjects at a given moment. Operations that change this state are themselves authorized by the current state. This poses a design problem in constructing the initial state so that all derivable states conform to a particular policy. It also raises an analysis problem of characterizing the protection states derivable from a given initial state. A protection model provides a framework for both design and analysis. Design generality and tractable analysis are inherently conflicting goals. Analysis is particularly difficult if creation of subjects is permitted. The schematic protection model resolves this conflict by classifying subjects and objects into protection types. The privileges possessed by a subject consist of a type-determined part specified by a static protection scheme and a dynamic part consisting of tickets (capabilities). It is shown that analysis is tractable for this model provided certain restrictions are imposed on subject creation. A scheme authorizes creation of subjects via a binary relation on subject types. Our principal constraint is that this relation be acyclic, excepting loops that authorize a subject to create subjects of its own type. Our assumptions admit a variety of useful systems.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*systems specification methodology*: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*capability architectures*; C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; D.2.0 [**Software Engineering**]: General—*protection mechanisms*; D.4.6 [**Operating Systems**]: Security and Protection—*access controls*; *security kernels*; H.1.0 [**Models and Principles**]: General; H.2.0 [**Database Management**]: General—*security, integrity, and protection*; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous—*security*

General Terms: Design, Management, Security, Theory, Verification

Additional Key Words and Phrases: Dynamic authorization, flow analysis problem, maximal states, monotonic protection models, protection types, the safety problem, schematic protection model, tickets

## 1. *Introduction*

The *access control* or *protection* problem arises in any computer system that provides for controlled sharing of information among multiple users. Such systems can be viewed as consisting of *subjects* and *objects*. Active entities such as users and processes are subjects, whereas passive entities such as text files are objects. Protection is enforced by ensuring that only those operations for which the invoking subject possesses *privileges* in its *domain* actually get executed. Operations may be

A preliminary version of this paper, for the special case of the schematic protection model called SSR, "Analysis of acyclic attenuating systems for the SSR protection model," appears in the *Proceedings of the 1985 IEEE Symposium on Security and Privacy* (Oakland, Calif., Apr.). IEEE, New York, 1985, pp. 197–206. © 1985 IEEE.

Author's present address: Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210-1277.

performed on objects (e.g., reading a text file) and on subjects (e.g., blocking a process). We regard subjects and objects as mutually exclusive and use *entity* to denote either a subject or object. By definition, objects do not possess privileges. Passive entities that possess privileges (e.g., directories) are modeled as subjects.

The distribution of privileges in domains of subjects defines the *protection state* of a system. Henceforth we understand state to mean protection state. *Inert privileges* authorize operations that do not modify the state (e.g., reading a file). *Control privileges* authorize operations that modify the state; for example, user X authorizes user Y to read file Z. The paradigm is that an initial state is established and thereafter evolves as constrained by control privileges. The challenge is to construct an initial state such that all derivable states conform with the policy that the designer wishes to implement.

Now what do we mean by policy in this context? At the simplest level an *authorization policy* defines a set of *safe* states where the distribution of privileges is consistent with the underlying objectives; for example, the policy that user X cannot read file Y. At all times the system must be in a safe state. Safety considerations are typically *attribute based* in that the concern is with classes of entities identified by some common attribute rather than with specific individuals; for example, the policy that only users in department D can access files internal to department D. This policy is said to be *selective* since users and files in different departments are treated differently.

At a more sophisticated level it is not enough that the system be in a safe state. We must additionally ensure that the system arrives at safe states in a proper manner. For instance, consider the policy that users outside department D may access internal files of D provided the head of department D approves. Then any distribution of privileges to access internal files of D is safe by definition. However, the policy requires cooperation of the department head to arrive at safe states in which outsiders can access internal files of D. Besides being attribute based and selective, we say this policy is *discretionary* and *cooperative* because of the role of the department head. Discretionary power itself can be dynamically acquired; for example, the policy that the department head may designate any senior member of the department to provide outsiders access to internal files but may not designate a junior member for this purpose. In this manner the department head's discretionary power is limited by *mandatory* controls.

We are primarily concerned with the dynamic aspect of authorization policies. A *protection model* provides a framework and formalism for specification of such policies and should be general enough to accommodate conveniently the kinds of issues outlined above. But generality by itself is not sufficient. To understand the formal statement of a policy and to ensure that it captures our intent, we need to characterize the states that a system may arrive at from a given initial state. Since subjects are usually authorized to create new subjects and objects, we are confronted with unbounded systems, and it is not certain that such analysis can be decidable, let alone tractable, without sacrificing generality.

Analysis issues were first formalized [5] in context of the well-known access-matrix model [1, 3, 7, 8, 10]. Not surprisingly, analysis is undecidable in this general setting. Furthermore, the access matrix lacks any structure for conveniently addressing the policy concerns outlined above. On the other hand, the take-grant model [1, 6, 8, 11, 19] and its variations [12], although efficiently analyzable, accommodate only a very specific class of simple policies. Thus this conflict between generality and analysis is quite real.

The central point of this paper is to demonstrate that the conflicting goals of convenient generality and tractable analysis can be simultaneously achieved. For

this purpose we have developed the *Schematic Protection Model* (*SPM*), which we present in Section 2. The SPM formulation is an outgrowth of ideas developed in the author's Ph.D. dissertation [15–17], which, in turn, were based on Minsky's send–receive transport model [13]. Section 3 discusses some applications of SPM. More extensive examples are discussed in [16] and [18]. Section 4 develops concepts and terminology required for the analysis of Section 5. The principal result of this paper is that for a large class of SPM specifications, the analysis problem of deciding whether states derivable from a given initial state are safe is decidable and even tractable. SPM authorizes creation of subjects via a binary relation on subject types. Our major constraint is that this relation be acyclic, excepting loops that authorize a subject to create subjects of its own type. Our assumptions admit a variety of useful systems including the examples of Section 3 and of [16] and [18]. We conjecture that most, if not all, SPM specifications of practical interest will satisfy this constraint. Section 6 concludes the paper.

## 2. *The Schematic Protection Model*

The key to balancing the conflicting goals of generality and analysis in SPM is the notion of *protection type*. The intuitive concept is that instances of the same protection type are treated uniformly by control privileges. Henceforth, we use type as synonymous with protection type. A critical assumption in SPM is that entities are *strongly typed*, that is, every entity is created to be of a specific type, and its type cannot change thereafter.

SPM views the domain of a subject as consisting of two parts: a static type-dependent part defined by the *protection scheme* and a dynamic part consisting of *tickets* (capabilities). The scheme is defined in terms of types by the *security administrator* when a system is first set up and thereafter cannot be changed. The idea is that major policy decisions are built into the scheme, while details are reflected in the initial distribution of tickets. We find it useful to view an authorization scheme as analogous to a database schema and the distribution of tickets as analogous to an extensional database.

Tickets are privileges of the form Y/x, where Y identifies some unique entity and the *right symbol* x authorizes the possessor of this ticket to perform some operation on Y. More generally x may authorize some particular set of operations on Y. Tickets are unforgeable and cannot be generated at will by a subject. They can be acquired only in accordance with specific rules to be discussed shortly. We do not intend that tickets must be represented at run time as capabilities built into the addressing mechanism of a computer [2, 9]. The correspondence between tickets and run-time representation of dynamic privileges in a given mechanism is a separate issue. The assumption that a ticket carries only one right symbol simplifies the formal framework without loss of generality. Capabilities with multiple right symbols then correspond to sets of tickets. We often abbreviate sets of tickets in this manner; for example, Y/uvw denotes the tickets Y/u, Y/v, and Y/w.

2.1 Types and Right Symbols. The first step in defining a scheme is to specify the disjoint sets of *object types TO* and *subject types TS*. Their union $T$ is the entire set of *entity types*. The idea is that protection types are used to identify classes of entities that share some common attribute. For subjects this may be membership in a department or a particular position of authority in a group. For objects this may be a classification such as an internal document or a public document. By convention types are named in lowercase italics and entities in uppercase roman.

Similarly, italics and roman are used to name sets, functions, and relations whose domains involve types and entities, respectively. The type of an entity Y is denoted by $\tau$(Y), where $\tau$ is called the *type function.*

The next, or perhaps concurrent, step is to define the right symbols carried by tickets. The set of right symbols $R$ is partitioned into two disjoint subsets: $RI$ the set of inert rights and $RC$ the set of control rights. Examples of inert rights are the typical read, write, execute, and append privileges for a file. Because of the role of inert rights, the symbols in $RI$ require no interpretation for analysis purposes. The interpretation of symbols in $RC$ is discussed shortly.

Every right symbol x comes in two variations x and xc, where c is the *copy flag.* The only difference between the Y/x and Y/xc tickets is that the former cannot be copied from one domain to another, whereas the latter can, provided certain additional conditions to be defined shortly are true. It follows that presence of Y/xc in a domain subsumes the presence of Y/x, but not vice versa. We use x:c to denote x or xc with the understanding that multiple occurrences of x:c in the same context are either all read as x or all as xc. When used with multiple right symbols on a ticket, the copy flag applies to each symbol, that is, Y/uvc denotes Y/uc and Y/vc.

We denote the *type of a ticket* Y/x:c by $\tau$(Y/x:c) and define it to be the ordered pair $\tau$(Y)/x:c. That is, the type of a ticket is determined by the type of the entity it addresses and the right symbol it carries. Conventions for representing tickets, especially regarding the copy flag, extend in an obvious way to ticket types. In particular, $\tau$(Y/x) and $\tau$(Y/xc) are different ticket types. This is an important distinction because of the role of the copy flag. The entire set of ticket types is $T \times R$.

The remaining components of a scheme are defined in terms of functions and relations involving the sets $TS$, $T$, and $T \times R$. SPM requires that $T$ and $R$ be finite sets, so a scheme is defined by finite sets, relations, and functions. SPM recognizes three operations that change the protection state: *copy*, *demand*, and *create.*

2.2 THE COPY OPERATION. The copy operation moves a copy of a ticket from the domain of one subject to the domain of another leaving the original ticket intact. We often speak of copying a ticket from one subject to another, although technically a ticket is copied from one subject's domain to another's domain. In addition to the copy flag, this operation is authorized by a *link predicate* $\text{link}_i$ defined by control rights and its associated *filter function* $f_i$ which is a component of the scheme.

2.2.1 *Link Predicates.* A link predicate takes two subjects, say X and Y, as arguments and evaluates to true or false. If true, it establishes a connection from X to Y that can be used to copy tickets from the domain of X to the domain of Y. Its definition is in terms of the presence of some combination of control tickets for X and Y in the domains of X and Y. The idea is that the link predicate should be evaluated by examining the domains of the two subjects of concern, and that only with respect to the presence of control tickets for the two subjects. We emphasize this by calling it a *local* link predicate. That its definition should depend only on the presence and not the absence of tickets is a well-known principle for protection [14]. As a special case, we also allow a link predicate that is always true to be defined. Formally, we have the following definition.

*Definition* 1. Let dom(X) be the set of tickets possessed by subject X. A *local link predicate* $\text{link}_i$(X, Y), with X and Y as formal parameters, is defined

as a conjunction or disjunction, but not negation, of the following terms for any $z \in RC$.

(1) $X/z \in \text{dom}(X)$,
(2) $X/z \in \text{dom}(Y)$,
(3) $Y/z \in \text{dom}(X)$,
(4) $Y/z \in \text{dom}(Y)$,
(5) **true.**

For a given state, if $\text{link}_i(A, B)$ is true, we say there is a $\text{link}_i$ from A to B. This is necessary but not sufficient for copying tickets from A to B. If the context permits it, we often omit the subscript i.

Some examples of local link predicates are listed below:

(1) $\text{link}(X, Y) \equiv Y/g \in \text{dom}(X) \vee X/t \in \text{dom}(Y)$,
(2) $\text{link}(X, Y) \equiv X/t \in \text{dom}(Y)$,
(3) $\text{link}(X, Y) \equiv Y/g \in \text{dom}(X)$,
(4) $\text{link}(X, Y) \equiv Y/s \in \text{dom}(X) \wedge X/r \in \text{dom}(Y)$,
(5) $\text{link}(X, Y) \equiv X/b \in \text{dom}(X)$,
(6) $\text{link}(X, Y) \equiv Y/p \in \text{dom}(Y)$,
(7) $\text{link}(X, Y) \equiv X/b \in \text{dom}(X) \wedge Y/p \in \text{dom}(Y)$,
(8) $\text{link}(X, Y) \equiv$ **true.**

The first example is from the take–grant model [11], where the t and g control rights are read, respectively, as take and grant. The next two examples each retain just one of these privileges [12]. The fourth example is from the send–receive mechanism [13, 15], where the s and r control rights are read, respectively, as send and receive. Thus the notion of a local link predicate is able to capture all these cases.

The first four cases are defined in terms of a control ticket for X in Y's domain, or vice versa. The next three cases are quite different and are defined in terms of a control ticket for X in X's domain, or similarly for Y. In the fifth case the definition of link depends only on X and is independent of Y. The control right b is read as broadcast. Similarly, in the sixth case the definition of link depends only on Y and is independent of X. Here the control right p is read as pickup. The seventh case is a combination of the fifth and sixth cases. The eighth case is unique in that it requires no tickets for a link to exist. We call this the *universal link*. There are other interesting possibilities for defining link predicates. We anticipate that simple link predicates of the kind defined above will suffice in practice, although the model does allow for arbitrarily complex predicates.

In SPM a finite collection of local link predicates $\{\text{link}_i \mid i = 1 \cdots N\}$ is defined in a scheme. If only one link predicate is defined, we drop the subscript. We emphasize that existence of a link from A to B is necessary but not sufficient for copying tickets from A to B.

2.2.2 *Filter Functions.* The final condition required for authorizing a copy operation is defined by the corresponding filter function $f_i$: $TS \times TS \rightarrow 2^{T \times R}$. The interpretation is that Y/x:c can be copied from dom(A) to dom(B) if and only if all of the following are true for some i, where the types of A, B, and Y are $a$, $b$, and $y$, respectively.

(1) $Y/xc \in \text{dom}(A)$,
(2) $\text{link}_i(A, B)$,
(3) $y/x:c \in f_i(a, b)$.

In this manner the copy flag, the link predicate, and the filter function together authorize a copy operation. The first two conditions depend on the distribution of tickets, whereas the third condition depends on the scheme. Selectivity in the copy operation is controlled by the filter function and specified entirely in terms of types. We emphasize there is a different filter function for each predicate $\text{link}_i$.

The filter functions are a powerful tool for specifying policies. They impose mandatory controls that are inviolable and confine the discretionary behavior of individual subjects. Some sample values for $f(a, b)$ are $T \times R$, $TO \times RI$, and $\phi$. In the first case all types of tickets can be copied from a subject of type $a$ to a subject of type $b$ provided the corresponding link predicate is true. In the second case only inert tickets for objects can be copied, whereas in the third case no tickets can be copied.

Finally we note that SPM imposes no assumptions regarding the role of A and B in a copy operation from A to B. It is equally acceptable for copying to take place at the initiative of A or B alone or to require both to cooperate.

2.3 THE DEMAND OPERATION. The demand operation allows a subject to obtain tickets simply by demanding them. A scheme authorizes this operation by the *demand function* $d: TS \rightarrow 2^{T \times R}$. The interpretation of $a/\text{x}:\text{c} \in d(b)$ is that every subject of type $b$ can demand the ticket A/x:c for every entity A of type $a$.

Demand is a method for specifying implicit distribution of tickets. If $a/\text{x}:\text{c} \in d(b)$, then every subject of type $b$ has the ability to access every entity of type $a$, including those that will be created some time in the future. Thus, when an entity A of type $a$ is created, it is not necessary to distribute A/x:c tickets to every subject of type $b$. Also, this access is immediately available to a newly created subject of type $b$, so when a subject of type $b$ is created, there is no need to explicitly provide it all tickets of type $a/\text{x}:\text{c}$. In particular, control tickets can be demanded. This allows for links to be set up between subjects on demand. Also a subject can obtain copiable control tickets for other subjects in this manner, thereby obtaining the authority to establish links involving these subjects. Both features are useful in specifying policies. We shall see the utility of a demand operation for both inert and control tickets in the schemes discussed in Section 3.

2.4 THE CREATE OPERATION. The create operation introduces new subjects and objects into the system. There are two issues here: What types of entities can be created and which tickets are introduced as the immediate result of a create operation?

2.4.1 *Authorization.* The first issue is specified in a scheme by the *can-create relation* $cc \subseteq TS \times T$. The interpretation is that subjects of type $a$ are authorized to create entities of type $b$ if and only if $cc(a, b)$. For notational convenience we sometimes regard $cc$ as a function $cc: TS \rightarrow 2^T$.

Visualize the $cc$ relation as a directed graph with vertex set $T$ and an edge from $a$ to $b$ if and only if $cc(a, b)$. We say $cc$ is *acyclic* if this graph contains no cycles excepting *loops*, that is, edges that connect a vertex to itself. An object type can only have incoming edges, so the acyclic restriction has no effect regarding creation of objects. For subject creation the acyclic restriction states that, if subjects of type $a$ can directly or indirectly create subjects of type $b$, then it is not possible for subjects of type $b$ to directly or indirectly create subjects of type $a$, unless $a = b$. In effect the acyclic assumption eliminates a kind of mutual recursion in $cc$. This is consistent with modern principles of system design that call for simple hierarchical structures. It is also consistent with the natural hierarchical structure of typical organizations. The analysis of Section 5 is based on this assumption.

2.4.2 *Create-Rules.* The tickets introduced by a create operation are specified by a *create-rule* for every pair in *cc*. Let subject A of type *a* create entity B of type *b*. If B is an object, the create-rule $cr(a, b)$ tells us which tickets for B are placed in dom(A) as a result of this operation. If B is a subject, the create-rule must also tell us which tickets for A are placed in dom(B). SPM requires every create-rule be *local* in that the only tickets introduced are for the creating and created entities in the domains of the creating and created entities. The motivation is that frequently occurring incremental events, such as creation, should immediately have only a local incremental impact on the state. Each ticket generated by a create-rule may or may not carry the copy flag as specified by the rule. We emphasize the create-rule may be different for each pair in *cc*.

The facility to generate copiable control tickets for a created subject is certainly useful. The policy decision as to whether the creator, or the created subject, or both get these tickets is properly left open by the model. It is quite reasonable to place copiable control tickets for the created subject in the creator's domain, since this gives the creator some discretionary control over the created subject. Similarly, placing copiable control tickets for the created subject in the created subject's own domain in reasonable. Placing copiable control tickets for the creator in the created subject's domain is also a valid policy option. For example, a policy may allow an *ordinary user* U to create a very powerful subject M of type *system manager* with the intention that M be used solely for experimentation by U in isolation from the rest of the system. If M gets copiable control tickets for U, then M may create a complex subsystem with which U can interact. Placing copiable control tickets for the creator in the creator's own domain is a more subtle issue. We return to this issue shortly, after defining our notation for create-rules.

When an object is created, the only tickets that can be introduced by a local create-rule are inert tickets for the created object in the creator's domain. For such cases the create-rule is specified as $cr(a, b) \subseteq \{b/\mathrm{x}:\mathrm{c} \mid \mathrm{x}:\mathrm{c} \in RI\}$. The interpretation is that when a subject A of type *a* creates an object B of type *b*, A gets $\mathrm{B}/\mathrm{x}:\mathrm{c}$ if and only if $b/\mathrm{x}:\mathrm{c} \in cr(a, b)$.

For subject creation the situation is more complex. First, consider subject types *a* and *b* such that $a \neq b$. In such cases we specify the create-rule in two parts separated by a vertical line; that is, $cr(a, b) = LEFT \mid RIGHT$. Let subject A of type *a* create subject B of type *b*. The left part of $cr(a, b)$ specifies the tickets placed in A's domain, and the right part specifies tickets placed in B's domain. Both *LEFT* and *RIGHT* are subsets of $\{a/\mathrm{x}:\mathrm{c},\ b/\mathrm{x}:\mathrm{c} \mid \mathrm{x}:\mathrm{c} \in R\}$. The interpretation is that A gets $\mathrm{A}/\mathrm{x}:\mathrm{c}$ provided $a/\mathrm{x}:\mathrm{c} \in LEFT$, and $\mathrm{B}/\mathrm{x}:\mathrm{c}$ provided $b/\mathrm{x}:\mathrm{c} \in LEFT$. Similarly, B gets $\mathrm{A}/\mathrm{x}:\mathrm{c}$ provided $a/\mathrm{x}:\mathrm{c} \in RIGHT$, and $\mathrm{B}/\mathrm{x}:\mathrm{c}$ provided $b/\mathrm{x}:\mathrm{c} \in RIGHT$. If $a = b$, this notation breaks down, since it is not apparent whether the ticket types in the create-rule refer to the created subject or to the creator. For such cases we introduce a special symbol *self* and specify *LEFT* and *RIGHT* as subsets of $\{a/\mathrm{x}:\mathrm{c},\ self/\mathrm{x}:\mathrm{c} \mid \mathrm{x}:\mathrm{c} \in R\}$, with the understanding that $self/\mathrm{x}:\mathrm{c}$ denotes tickets for the creator, whereas $a/\mathrm{x}:\mathrm{c}$ denotes tickets for the created subject.

2.4.3 *Attenuating Create-Rules.* The analysis of Section 5, in addition to the acyclic assumption regarding *cc*, requires an assumption about create-rules for loops in *cc*, that is, create-rules of the form $cr(a, a)$. Consider subject A of type *a* such that $cc(a, a)$. Subject A can create a child A′ of type *a*, which in turn can create a child A″ of type *a*, and so on. The possibility of indefinitely long chains of create operations complicates analysis. The acyclic restriction eliminates certain kinds of indefinitely long chains but does not eliminate them completely because

of loops. We account for loops in *cc* by insisting that the child A′ be "no more powerful" than its creator A. Since A and A′ are both of type *a*, this is a somewhat curious requirement. The crucial difference between A and A′ lies in the tickets introduced by $cr(a, a)$ when A creates A′.

Our first restriction on $cr(a, a)$ is that immediately after the create operation $\mathrm{dom(A')} \subseteq \mathrm{dom(A)}$. This is consistent with the principle of attenuation [14] in that a newly created subject should not get more tickets than its creator. Our second restriction is more subtle. It requires that, if a ticket for A′ is placed in dom(A), the corresponding ticket for A should also be placed in dom(A). The formal definition of these restrictions is as follows.

*Definition* 2. A create-rule $cr(a, a) = LEFT \mid RIGHT$ is *attenuating* provided

(1) $RIGHT \subseteq LEFT$,
(2) $a/\mathrm{x}:\mathrm{c} \in LEFT \Rightarrow self/\mathrm{x}:\mathrm{c} \in LEFT$.

A scheme is attenuating if, for all *a* such that $cc(a, a)$, the create-rule $cr(a, a)$ is attenuating.

Note that for an attenuating scheme only the create-rules for loops in *cc* are required to be attenuating.

In order to understand the attenuating assumption, we focus on control tickets introduced by $cr(a, a)$ when subject A of type *a* creates subject A′ of type *a*. We say a control right $\mathrm{z} \in RC$ is *external* if z occurs in the definition of a predicate $\mathrm{link}_i(\mathrm{X}, \mathrm{Y})$ in terms of the form $\mathrm{X/z} \in \mathrm{dom(Y)}$ or $\mathrm{Y/z} \in \mathrm{dom(X)}$. Conversely, z is *internal* if it occurs in terms of the form $\mathrm{X/z} \in \mathrm{dom(X)}$ or $\mathrm{Y/z} \in \mathrm{dom(Y)}$. For the collection of link predicates defined in Section 2.2, the rights t, g, s, and r are external, whereas b and p are internal. In principle it is possible that a control right is both external and internal, but we ignore that possibility in our discussion here.

Let subject A of type *a* create subject A′ of type *a*. Consider an external control right such as the grant right that defines link(X, Y) by $\mathrm{Y/g} \in \mathrm{dom(X)}$. Placing A′/g in dom(A′) or A/g in dom(A) is meaningless and can be assumed to occur if required to make $cr(a, a)$ attenuating. So the attenuating assumption only applies to copiable grant tickets, that is, tickets with the gc right. The first part of the attenuating assumption says that, if A′ gets A′/gc or A/gc, then A should also get A′/gc or A/gc, respectively. The second part says that, if A gets A′/gc, then A should also get A/gc. The motivation for the first part is to ensure that all tickets possessed by A′ immediately after the create operation are also possessed by A, so A′ is not "more powerful" than A on this account. For the second part consider what happens if A gets A′/gc but does not possess A/gc. The A′/gc ticket (partially) authorizes A to establish link(B, A′). So it may be possible to establish link(B, A′), whereas link(B, A) cannot be established. But then A′ is "more powerful" than A, in a sense. The second part of the attenuating assumption eliminates this possibility.

Similarly, consider an internal control right such as pickup that defines link(X, Y) by $\mathrm{Y/p} \in \mathrm{dom(Y)}$. Now placing A′/p in dom(A) is meaningless and can be assumed to occur if required to make $cr(a, a)$ attenuating. If A′/p is placed in dom(A′) but A/p is not in dom(A), then clearly A′ is "more powerful," since there is a link from every subject to A′ but not to A. With an attenuating create rule this cannot happen.

The net effect of the attenuating assumption can be formulated slightly differently if we allow the effect of the create rule to depend on the current domain of the creator.

*Definition* 3. A create-rule $cr(a, a) = LEFT \mid RIGHT$ is *attenuating* provided

(1) $RIGHT \subseteq LEFT$.
(2) Only those tickets A′/z:c or A/z:c for which the A/z:c ticket is present in the creator's current domain, before the create operation, will be actually introduced by the create-rule.

Thus A′/gc will be placed in dom(A′) or dom(A) only if A already possesses the ticket A/gc. Similarly, A′/p will be placed in dom(A) only if A already has A/p. Definition 3 is a conservative counterpart of Definition 2. The analysis of this paper applies equally well to either definition, and it is a matter of taste as to which one is preferred.

As a final point on create-rules, we mention that the create-rules can always be treated as upper bounds on the tickets that can be generated, instead of specifying the exact tickets that will be generated. We can allow subjects to specify a subset of tickets permitted by the create-rule to be actually generated for a particular create operation. This will not affect the analysis of this paper.

2.5 SUMMARY AND DISCUSSION. In summary, the Schematic Protection Model requires the security administrator to specify a protection scheme by defining the following components:

(1) A finite set of entity types $T$ partitioned into subject types $TS$ and object types $TO$.
(2) A finite set of right symbols $R$ partitioned into inert rights $RI$ and control rights $RC$.
(3) A finite collection of local link predicates $\{\text{link}_i \mid i = 1 \cdots N\}$.
(4) A filter function $f_i: TS \times TS \rightarrow 2^{T \times R}$ corresponding to each $\text{link}_i$.
(5) The demand function $d: TS \rightarrow 2^{T \times R}$.
(6) The can-create relation $cc \subseteq TS \times T$. Equivalently, $cc: TS \rightarrow 2^T$.
(7) A local create-rule for each pair in $cc$.

A *system* is specified by defining a protection scheme and the initial protection state, that is, the initial set of entities and the initial distribution of tickets. Thereafter, the protection state evolves by copy, demand, and create operations.

A scheme is acyclic if its $cc$ relation is acyclic, excepting loops of the form $cc(a, a)$ that authorize a subject to create subjects of its own type. A scheme is attenuating if all create-rules of the form $cr(a, a)$ are attenuating. The analysis in this paper is based on acyclic attenuating schemes. These restrictions admit a wide variety of schemes of practical interest. Indeed we have not been able to construct any realistic scheme that cannot be formulated as an acyclic attenuating scheme. Our conjecture is that most, if not all, schemes of practical interest will turn out to have an acyclic attenuating formulation.

Our approach to analysis is based on a worst-case scenario in which we assume all subjects will cooperate with one another. Now it is certainly desirable to analyze systems under assumptions about the behavior of specific subjects. For some mechanisms this may be unavoidable. For example, the UNIX[1] superuser has privileges to effect essentially arbitrary changes in the protection state. To do any interesting analysis in this environment, we must make assumptions that superusers will not exercise their privileges arbitrarily. SPM is able to accommodate such assumptions to the extent they can be captured in the scheme. Thus the worst-case scenario is more flexible than may appear at first sight.

[1] UNIX is a trademark of AT&T Bell Laboratories.

2.5.1 *Revocation and the Restoration Principle.* SPM lacks facilities for revocation of tickets and deletion of entities. This reflects a deliberate decision to focus on policies for acquisition of tickets and creation of entities, while setting aside the issue of specifying revocation and deletion policies for the moment. Fortunately, it turns out that under rather general assumptions revocation and deletion can be ignored for analysis purposes. Also, this greatly simplifies analysis because without revocation and deletion the protection state evolves in a monotonic manner.

Revocation can be ignored in a worst-case scenario provided the effect of revocation can be undone. We call this the *restoration principle*; that is, whatever can be revoked can be restored. In SPM, if a ticket obtained by a copy or demand operation is revoked, it is easily restored by repeating the operation. However, if a ticket introduced by a create operation is revoked, it may not be restorable by repeating the operation, since each created entity is unique. Also tickets distributed in the initial state may not be restorable. If we assume tickets distributed in the initial state or introduced by create-rules are irrevocable, the restoration principle does not entail any loss of generality in context of SPM. The need for a restoration principle is also demonstrated by the lost object problem. With unrestricted revocation it is possible that all tickets for an object may disappear. If tickets for this object cannot be generated on demand, the object thereby becomes inaccessible.

The situation regarding deletion of entities is similar. Here the restoration principle requires that an entity that can be deleted should be replaceable by an equivalent entity. In general, this rules out deletion of entities present in the initial state. Regarding deletion of entities created subsequently, it is always possible to re-create an entity of the same type as was deleted. In other words, the individuality of created entities is not significant for analysis of the safety problem, whereas the individuality of entities in the initial state may be significant.

To summarize, revocation and deletion policies that are consistent with the restoration principle can be ignored for analysis of the safety problem in a worst-case scenario.

## 3. *Applications*

In this section we specify a variety of policies in the SPM formalism. Our objective is to demonstrate the power of the SPM framework rather than provide an exhaustive investigation of policy alternatives in the different contexts considered. The examples are necessarily brief. We point out that the schemes discussed in this section, as well as those in the more extensive case studies of [16] and [18], are all either acyclic attenuating or can be easily reformulated to be acyclic attenuating.

3.1 OWNER-BASED POLICIES. The concept of ownership is a well-known approach to dynamic authorization. A user is regarded as the owner of all files he or she has created and has complete discretion regarding access to these files. Current operating system mechanisms are mostly based on this concept. In this context a simple policy is that a user U can authorize another user U′ to access file F if and only if U is the owner of F. The following scheme specifies this policy in SPM:

*Scheme I.* Basic owner-based policy.

(1) $TS = \{user\}$, $TO = \{file\}$
(2) $RI = \{x:c\}$, $RC = \phi$
(3) $\text{link}_u(X, Y) \equiv \textbf{true}$
(4) $f_u(user, user) = \{file/xc\}$
(5) $d(user) = \phi$

(6) $cc(user) = \{file\}$
(7) $cr(user, file) = \{file/\text{xc}\}$

The types *user* and *file* obviously correspond to users and files, respectively. For simplicity, a single inert right x:c provides access to files. This suffices so long as the policy regarding the dynamics of different inert rights, such as the typical read, write, execute, and append, remains the same. There are no control rights, so only the universal link predicate is defined. Tickets for files, with or without the copy flag, can be copied across universal links. Users can create files and get a copiable ticket for each created file.

We now refine the policy to allow users to set up groups for distribution of tickets. The idea is that a user can identify a group of users and provide access to files on a group basis instead of having to distribute tickets to all members of a group. For this purpose we introduce a second subject type *group* and a control right g for grant. The grant right is used to define the link predicate $\text{link}_g(X, Y)$ as $Y/g \in \text{dom}(X)$. We authorize users to create groups by placing *group* in $cc(user)$. The corresponding create-rule establishes $\text{link}_g(U, G)$ on creation of a group G by user U. This is the only way to establish a $\text{link}_g$ from a user to a group, so the only incoming $\text{link}_g$ to a group is from its creator. Membership in a group is effected by establishing links from the group to each user who is a member, and this is entirely at the discretion of the group's creator. For this purpose we authorize users to demand all tickets of type *user*/gc. By placing $user/g \in f(user, group)$, we allow the creator of a group to move grant tickets for all users who are members of the group into the group's domain. Also, by placing *file*/xc in $f(user, group)$, we authorize the creator of a group to make files available to group members by moving such tickets into the group's domain. Finally, we authorize users to obtain tickets for files from a group to which they belong by placing *file*/x in $f(group, user)$. This results in the following scheme:

*Scheme II.* Owner-based policy with owner defined groups.

(1) $TS = \{user, group\}$, $TO = \{file\}$
(2) $RI = \{\text{x:c}\}$, $RC = \{\text{g:c}\}$
(3) $\text{link}_u(X, Y) \equiv \textbf{true}$
$\text{link}_g(X, Y) \equiv Y/g \in \text{dom}(X)$
(4) $f_u(user, user) = \{file/\text{xc}\}$ $f_g(user, user) = \phi$
$f_u(user, group) = \phi$ $f_g(user, group) = \{file/\text{xc}, user/g\}$
$f_u(group, user) = \phi$ $f_g(group, user) = \{file/\text{x}\}$
$f_u(group, group) = \phi$ $f_g(group, group) = \phi$
(5) $d(user) = \{user/\text{gc}\}$
(6) $cc(user) = \{file, group\}$
$cc(group) = \phi$
(7) $cr(user, file) = \{file/\text{xc}\}$
$cr(user, group) = \{group/g\} \mid \phi$

Additional examples of schemes based on the notion of ownership and sharing by means of groups are discussed in [18].

This scheme is acyclic and has no loops in *cc*, so it is also attenuating. It is worth considering the enhancement that $user \in cc(user)$. This allows a user to create another user perhaps simply for organizing his or her own workspace. Since

user-to-user links can be established on demand, *cr*(*user*, *user*) need not introduce any tickets. Thus the scheme remains acyclic attenuating.

3.2 THE DEPARTMENTAL EXAMPLE. Next we consider the policy issues raised in Section 1. For simplicity, assume there is a single department. Users in the department can access internal documents of that department, but outsiders cannot. For this purpose we divide users into two types: *in* for insiders and *out* for outsiders. We define a single object type *idoc* for internal documents and a single inert right x. The policy is easily specified by means of the demand function as follows:

*Scheme III*

(1) $TS = \{in, out\}$, $TO = \{idoc\}$
(2) $RI = \{x:c\}$, $RC = \phi$
(3) $d(in) = \{idoc/x\}$
    $d(out) = \phi$
(4) $cc(in) = \{idoc\}$
(5) $cr(in, idoc) = \phi$

Every insider can create internal documents and obtain access to all such documents on demand. There are no links in this scheme, and therefore the copy operation is ruled out. In practice this scheme would be a fragment of a larger scheme, for instance, one with several departments and document types.

As a refinement to this policy we single out the department head as a distinguished insider who can authorize outsiders to access *idoc*'s. For this purpose we define a new subject type *head* for department head with all privileges of *in* and some additional ones. Tickets for *idoc*'s are provided by a department head to outsiders by the copy operation. Since the department head has complete discretion in this regard, we define a single control right b (for broadcast), which sets up a link from the department head to all subjects, that is, link(A, B) if and only if $A/b \in dom(A)$. We authorize the department head to demand *head*/b and thereby establish links to every subject on demand. We place *idoc*/xc in *d*(*head*) rather than just *idoc*/x. And finally, we allow tickets of type *idoc*/x to be copied to outsiders from the department head's domain by setting *f*(*head*, *out*) to {*idoc*/x}. These changes result in the following scheme:

*Scheme IV*

(1) $TS = \{in, head, out\}$, $TO = \{idoc\}$
(2) $RI = \{x:c\}$, $RC = \{b:c\}$
(3) $link(A, B) \equiv A/b \in dom(A)$
(4) $f(head, out) = \{idoc/x\}$
    All other values of $f$ are empty.
(5) $d(in) = \{idoc/x\}$
    $d(head) = \{idoc/xc, head/b\}$
    $d(out) = \phi$
(6) $cc(in) = \{idoc\}$
    $cc(head) = \{idoc\}$
(7) $cr(in, idoc) = \phi$
    $cr(head, idoc) = \phi$

This scheme has no facility for restricting the number of instances of *head*, that is, several department heads may exist. This number gets fixed once the initial state is defined. Outsiders can access *idoc*'s only if at least one instance of *head* is present. This illustrates, in a somewhat trivial way, why analysis must consider the scheme as well as the initial state.

As a final refinement we allow the department head to delegate his or her discretionary power to senior members of the department. We do this by replacing *in* by two types *jun* and *sen* for junior and senior members, respectively. The type *jun* has exactly the privileges of *in*, whereas *sen* has some privileges in addition to these. Specifically, the discretionary ability of a department head to broadcast tickets of type *idoc*/x to outsiders can be acquired by senior members as follows:

(1) Let the department head demand copiable broadcast tickets for senior members, that is, *sen*/bc $\in d(head)$.
(2) Authorize the department head to copy broadcast tickets to domains of senior members, that is, *sen*/b $\in f(head, sen)$.
(3) Authorize senior members to demand copiable tickets for internal documents, that is, *idoc*/xc $\in d(sen)$ rather than just *idoc*/x $\in d(sen)$.
(4) Authorize senior members to copy tickets for internal documents to outsiders, that is, *idoc*/x $\in f(sen, out)$.

In short, a senior member cannot export tickets for internal documents to outsiders until the department head gives him or her the broadcast privilege. The result is the following scheme.

*Scheme V*

(1) $TS = \{jun, sen, head, out\}$, $TO = \{idoc\}$
(2) $RI = \{x:c\}$, $RC = \{b:c\}$
(3) link(A, B) $\equiv$ A/b $\in$ dom(A)
(4) $f(sen, out) = \{idoc/x\}$
$f(head, out) = \{idoc/x\}$
$f(head, sen) = \{sen/b\}$
All other values of $f$ are emtpy.
(5) $d(jun) = \{idoc/x\}$
$d(sen) = \{idoc/xc\}$
$d(head) = \{idoc/xc, sen/bc, head/b\}$
$d(out) = \phi$
(6) $cc(jun) = \{idoc\}$
$cc(sen) = \{idoc\}$
$cc(head) = \{idoc\}$
(7) $cr(jun, idoc) = \phi$
$cr(sen, idoc) = \phi$
$cr(head, idoc) = \phi$

It is instructive to consider the restoration principle in the context of this scheme. A reasonable revocation policy would be to allow the department head to revoke access to internal documents from outsiders and to revoke the broadcast privilege from the domains of senior members. This policy is consistent with the restoration principle because a ticket that is revoked can be restored by the department head.

The preceding three schemes do not allow creation of subjects and are trivially acyclic attenuating. We can extend scheme V to allow subject creation, for instance, by modifying *cc* as follows:

$$cc(head) = \{idoc, head, sen, jun\},$$
$$cc(sen) = \{idoc, sen, jun\},$$
$$cc(jun) = \{idoc, jun\}.$$

The modified scheme is acyclic but with loops. It should be evident that the create-rules for creation of subjects need not introduce any tickets, so the scheme remains acyclic attenuating.

3.3 THE TAKE–GRANT MODEL. In our final example we demonstrate how the take–grant model [11] is specified in SPM. The name take–grant is due to the control rights t for take and g for grant. A single link predicate link(X, Y) is defined by $Y/g \in dom(X)$ or $X/t \in dom(Y)$. That is, a link requires a grant ticket at the source or a take ticket at the destination.

In the basic take–grant model there is a single subject type that we call *sub*. The model does not explicitly include inert rights and objects in the sense of SPM. Nevertheless, we include these in our specification by defining the object type *file* and the inert right x. There is no selectivity in the copy operation, so we set $f(sub, sub)$ to be $T \times R$. Since there is no demand operation in the model, we set $d(sub)$ to be empty. Subjects are allowed to create files and other subjects, and this is easily specified by $cc(sub)$. On creation of a file, the creator gets a copiable ticket for it. On creation of a subject, the creator gets copiable take and grant tickets for the created subject. This establishes links between the creator and created subject in both directions. All this results in the following scheme, which is equivalent to the take–grant model.

*Scheme VI.* The basic take–grant model.

(1) $TS = \{sub\}$, $TO = \{file\}$

(2) $RI = \{x:c\}$, $RC = \{t:c, g:c\}$

(3) $link(X, Y) \equiv Y/g \in dom(X) \vee X/t \in dom(Y)$

(4) $f(sub, sub) = T \times R$

(5) $d(sub) = \phi$

(6) $cc(sub) = \{file, sub\}$

(7) $cr(sub, file) = \{file/xc\}$
$cr(sub, sub) = \{sub/tgc\} \mid \phi$

We mention that the take–grant model does not include the notion of copy flag. This requires us to make the additional assumption that all tickets in the initial state are copiable.

In this scheme $cr(sub, sub)$ is not attenuating. We can modify this create-rule to be attenuating by reformulating it as $\{sub/tgc, self/tgc\} \mid \phi$. With this change, whenever a subject creates another subject, the creator gets copiable take and grant tickets for itself. This amounts to the assumption that every subject has copiable take and grant tickets for itself. For a subject A created subsequent to the initial state, from a worst-case viewpoint this assumption is without loss of generality. The creator of A does get the A/tc and A/gc tickets and can copy these to dom(A). However, for the initial set of subjects, this assumption is not satisfactory.

This suggests that we need to treat the initial set of subjects differently from subjects created subsequently. For this purpose we break *sub* into two subject types: *isub* for initial subjects and *csub* for created subjects. All subjects in the initial state are of type *isub*, whereas those created subsequently are of type *csub*. These two types differ only with respect to *cc* and the create-rules. Only *csub*'s can be created, so we place *csub* in *cc*(*isub*) and *cc*(*csub*). The create-rules for these two cases differ in that *cr*(*isub*, *csub*) does not generate tickets for the creator, whereas *cr*(*csub*, *csub*) does. This makes the resulting scheme acyclic attenuating, as shown below.

*Scheme VII.* The basic take-grant model as an acyclic attenuating scheme.

(1) $TS = \{isub, csub\}$, $TO = \{file\}$
(2) $RI = \{x:c\}$, $RC = \{t:c, g:c\}$
(3) $\text{link}(X, Y) \equiv Y/g \in \text{dom}(X) \vee X/t \in \text{dom}(Y)$
(4) $f(isub, isub) = T \times R$
$f(isub, csub) = T \times R$
$f(csub, isub) = T \times R$
$f(csub, csub) = T \times R$
(5) $d(isub) = \phi$
$d(csub) = \phi$
(6) $cc(isub) = \{file, csub\}$
$cc(csub) = \{file, csub\}$
(7) $cr(isub, file) = \{file/xc\}$
$cr(csub, file) = \{file/xc\}$
$cr(isub, csub) = \{csub/tgc\} \mid \phi$
$cr(csub, csub) = \{csub/tgc, self/tgc\} \mid \phi$

This scheme is equivalent to Scheme VI, in the worst case but is attenuating. Thus the attenuating requirement is less restrictive than may appear at first sight.

Next we consider the so called subject–object version of take–grant [6]. From our viewpoint, in this version there are two types of subjects: *asub* for active subjects and *psub* for passive subjects. A passive subject does not initiate any operation and is merely a repository for tickets. A link from one active subject to another active subject is much the same as before. However, a link from a passive subject to another passive subject cannot be used to copy tickets. This is easily achieved in SPM by setting the appropriate value of the filter function to empty.

In the take–grant model a link from an active subject A to a passive subject B can be used to copy tickets only if the link is established by $B/g \in \text{dom}(A)$. Similarly, a link to an active subject A from a passive subject B can be used to copy tickets only if the link is established by $B/t \in \text{dom}(A)$. In both of these cases, the link can be used to copy tickets only if it is established by a ticket for the passive subject in the domain of the active subject. To specify this in the SPM framework, we need two link predicates as follows:

(1) $\text{link}_g(X, Y) \equiv Y/g \in \text{dom}(X)$,
(2) $\text{link}_t(X, Y) \equiv X/t \in \text{dom}(Y)$.

Regarding interaction between active subjects or between passive subjects, these two kinds of links are equivalent, and we define the corresponding values of the

filter functions as follows:

(1) $f_g(asub, asub) = T \times R, f_g(psub, psub) = \phi$,
(2) $f_t(asub, asub) = T \times R, f_t(psub, psub) = \phi$.

Regarding interaction between an active subject and a passive subject, these two kinds of links are different, and we have the following values for the filter function:

(1) $f_g(asub, psub) = T \times R, f_g(psub, asub) = \phi$,
(2) $f_t(asub, psub) = \phi, f_t(psub, asub) = T \times R$.

This example demonstrates the utility of multiple link predicates and filter functions in SPM.

It remains to consider the create operation. Passive subjects are not allowed to create entities, whereas active subjects can create all types of entities. The create-rules are essentially the same as in the previous scheme. The resulting scheme is shown below.

*Scheme VIII.* The take–grant model with passive subjects.

(1) $TS = \{asub, psub\}$, $TO = \{file\}$
(2) $RI = \{x:c\}$, $RC = \{t:c, g:c\}$
(3) $\text{link}_g(X, Y) \equiv Y/g \in \text{dom}(X)$ $\quad \text{link}_t(X, Y) \equiv X/t \in \text{dom}(Y)$
(4) $f_g(asub, asub) = T \times R$ $\quad f_t(asub, asub) = T \times R$
$f_g(asub, psub) = T \times R$ $\quad f_t(asub, psub) = \phi$
$f_g(psub, asub) = \phi$ $\quad f_t(psub, asub) = T \times R$
$f_g(psub, psub) = \phi$ $\quad f_t(psub, psub) = \phi$
(5) $d(asub) = \phi$
$d(psub) = \phi$
(6) $cc(asub) = \{file, asub, psub\}$
$cc(psub) = \phi$
(7) $cr(asub, file) = \{file/xc\}$
$cr(asub, asub) = \{asub/tgc\} \mid \phi$
$cr(asub, psub) = \{psub/tgc\} \mid \phi$

Because of $cr(asub, asub)$, this scheme is not attenuating. The technique used to convert Scheme VI into its attenuating version VII is again applicable. Here we would need to refine *asub* into two types, *iasub* and *casub*, to distinguish the initial set of active subjects from those created subsequently.

## 4. *The Flow Function and Maximal States*

Having demonstrated the expressive power of SPM we turn to analysis. In this section we develop the terminology and concepts required for this purpose. A change in state caused by a single copy, demand, or create operation is called a *transition.* A transition is *legal* provided there is proper authorization for the operation causing it. A *history* is a sequence of legal transitions. Histories are denoted by uppercase roman letters and states by lowercase roman letters or special symbols. Unless otherwise mentioned, a history is applied to the initial state. Any state that can be derived by a history is *derivable.*

In analysis we are interested in functions and relations that depend on the state, for example, dom and $\text{link}_i$. When appropriate, we qualify these with a superscript to identify the state, for example, $\text{dom}^h$ and $\text{link}_i^h$ identify the context as state h.

The initial state is identified by the superscript 0. The set of subjects and entities in state h are, respectively, denoted by $SUB^h$ and $ENT^h$. Both dom and $link_i$ exhibit a monotonic property because of the absence of revocation and deletion; that is, if g is derived from h, then $link_i^h \subseteq link_i^g$, and for all $A \in SUB^h$, $dom^h(A) \subseteq dom^g(A)$. Because the functions and relations used in analysis depend on the presence rather than absence of tickets in domains, this monotonic property extends to all the functions and relations we consider.

4.1 THE FLOW FUNCTION. The flow function expresses the authorization for copying tickets from one subject to another in a given state accounting for indirect as well as direct copying. For every pair of subjects, its value is a set of ticket types determined by the state and scheme. Its definition is based on the following notion.

*Definition* 4. There is a *path*$^h$ from A to B provided either one of the following conditions is true:

(1) For some i, $link_i^h(A, B)$.

(2) There exists a sequence of subjects $C_1C_2 \cdots C_n$ such that for some $i_0i_1 \cdots i_n$, $link_{i_0}^h(A, C_1)$, $link_{i_k}^h(C_k, C_{k+1})$, $k = 1 \cdots n - 1$, and $link_{i_n}^h(C_n, B)$.

In the former case we say the path is *single link*, whereas in the latter case the path is *multilink* and *traverses* $C_1C_2 \cdots C_n$ with *word* $i_0i_1 \cdots i_n$.

Consider a multilink path from A to B that traverses $C_1C_2 \cdots C_n$. Let $Y/xc \in dom(A)$. Y/xc can be copied from A to B using this path provided that Y/xc can be copied across each link in the path. Further, Y/x can be copied from A to B using this path provided that Y/xc can be copied across each link in the path from A to $C_n$ and Y/x can be copied from $C_n$ to B; that is, the copy flag must be copied on all except the last link. This leads to the following definition.

*Definition* 5. Define the *capacity* of a path$^h$ from A to B as follows, where the types of A and B are, respectively, *a* and *b*.

1. For a single link path, $link_i^h(A, B)$ the capacity is $f_i(a, b)$.
2. For a multilink path that traverses subjects $C_1C_2 \cdots C_n$ of types $c_1c_2 \cdots c_n$, respectively, with word $i_0i_1 \cdots i_n$, the capacity is the set of $y/x:c$ such that $y/xc \in f_{i_0}(a, c_1)$, $y/xc \in f_{i_k}(c_k, c_{k+1})$, $k = 1 \cdots n - 1$ and $y/x:c \in f_{i_n}(c_n, b)$.

Note that only the types of entities involved in this definition are significant, not their specific identities. Also recall our convention by which the two occurrences of $y/x:c$ above are either both read as $y/x$ or both as $y/xc$.

We are now ready to define the flow function.

*Definition* 6. For every state h define the *flow* function $flow^h$: $SUB^h \times SUB^h \rightarrow 2^{T \times R}$ by $flow^h(A, B)$ equal to the union of the capacity of all paths in state h from A to B. By convention $flow^h(A, A)$ is $T \times R$.

The convention regarding $flow^h(A, A)$ is consistent with the underlying intuition and is convenient.

Computation of $flow^h$ is straightforward in principle and of polynomial complexity in $|SUB^h|$ [15]. It is convenient first to compute the flow of ticket types with the copy flag. This can be done separately for each ticket type $y/xc$ by computing the transitive closure of $\{\langle A, B\rangle \mid$ there exists $link_i^h(A, B)$ with capacity including $y/xc\}$. It is then easy to account for ticket types that do not carry the copy flag since such tickets can be copied at most over a single link. Assigning a

cost of $O(|\,\mathrm{SUB}^h\,|^3)$ steps for each transitive closure computation, the entire computation requires $O(|\,T \times R\,| * |\,\mathrm{SUB}^h\,|^3)$ steps.

4.2 MAXIMAL STATES. The fundamental issue in analysis is to predict behavior of the flow function. This is especially so since create and demand operations are authorized solely by the scheme, whereas copy is authorized by both the scheme and the distribution of tickets. Because flow is monotonic, for a given pair of subjects it can only increase in derived states. From this fact we show in Section 4.3 that there exists a derivable state with the maximum value of flow between all subjects in $\mathrm{SUB}^0$. We call such a state a *maximal state.* In general, the maximal state is not unique. Indeed, the system can continue to evolve indefinitely by creation of new entities. A maximal state is a worst-case concept and will be derived only if all subjects cooperate toward this end.

Let flow* denote the flow function in a maximal state. By definition, flow* specifies the ticket types that can be copied from A to B, either directly or indirectly via some other subjects, in the worst case. We can then trivially determine whether a specific ticket can be copied from A to B. The safety problem [5] poses the question whether or not it is possible to have a derivable state with Y/x:c in dom(B). The flow* function allows us to answer this question by asking the following question: Is there any subject A who possesses Y/xc in the initial state or can demand Y/xc and $\tau(\mathrm{Y})/\mathrm{x}:\mathrm{c} \in \mathrm{flow}^*(\mathrm{A}, \mathrm{B})$?

Before formalizing the notion of maximal states and proving their existence, we present an example to illustrate changes in the flow function. The example is intended solely to demonstrate the concepts and does not have a meaningful practical interpretation. We use the following scheme:

*Scheme IX*

(1) $TS = \{a\}$, $TO = \phi$
(2) $RI = \phi$, $RC = \{\mathrm{s}:\mathrm{c}, \mathrm{r}:\mathrm{c}\}$
(3) $\mathrm{link}(\mathrm{X}, \mathrm{Y}) \equiv \mathrm{Y/s} \in \mathrm{dom}(\mathrm{X}) \wedge \mathrm{X/r} \in \mathrm{dom}(\mathrm{Y})$
(4) $f(a, a) = \{a/\mathrm{sc}\}$
(5) $d(a) = \{a/\mathrm{r}\}$
(6) $cc(a) = \{a\}$
(7) $cr(a, a) = \{a/\mathrm{sc}, a/\mathrm{r}\} \mid \{self/\mathrm{s}\}$

The requirement that $\mathrm{X/r} \in \mathrm{dom}(\mathrm{Y})$ in the definition of link is a technicality since Y can always demand this ticket.

For this scheme consider an initial state with two subjects A and B, both of type $a$, with $\mathrm{dom}^0(\mathrm{A}) = \{\mathrm{B/s}\}$ and $\mathrm{dom}^0(\mathrm{B}) = \phi$. Figure 1a shows this state, where the contents of dom(A) and dom(B) are listed within the circles labeled A and B, respectively. Since there is no link, both $\mathrm{flow}^0(\mathrm{A}, \mathrm{B})$ and $\mathrm{flow}^0(\mathrm{B}, \mathrm{A})$ are empty.

Let us see how this system evolves without create operations. B can demand the ticket A/r, thereby establishing link(A, B), as shown in Figure 1b, with the directed edge from A to B indicating link(A, B). Since B cannot get the A/s ticket, no more links can be established, and the maximum value of flow with respect to A and B has been realized in this state. We call such a state a *no-creates maximal state,* identified by the symbol #. By inspection of Figure 1b it is evident that $\mathrm{flow}^\#(\mathrm{A}, \mathrm{B}) = \{a/\mathrm{sc}\}$, while $\mathrm{flow}^\#(\mathrm{B}, \mathrm{A}) = \phi$.

Note that the # state is not unique. For instance, both A and B can, respectively, demand the tickets A/r and B/r, but this does not affect flow since these tickets do not establish any links. It should be apparent that a # state can be easily computed in general simply by executing copy and demand operations until the flow function

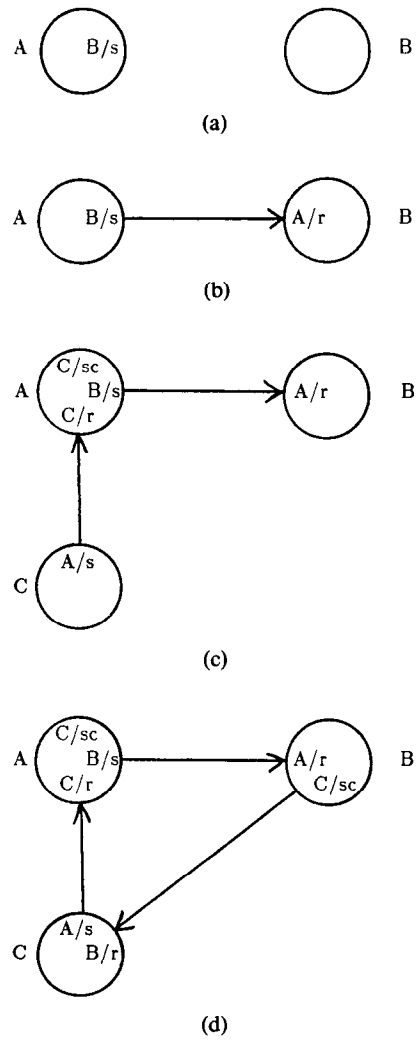


FIG. 1. Evolution of flow: (a) initial state; (b) no-creates maximal state; (c) subject A creates subject C; (d) maximal state.

stabilizes. Straightforward algorithms for this purpose can be easily constructed with complexity polynomial in $|\,SUB^0\,|$ [15]. Since the number of subjects does not change, by our earlier discussion the flow function for any derivable state can be computed in $O(|\,T \times R\,| * |\,SUB^0\,|^3)$ steps. If there are N link predicates, there are no more than $N * |\,SUB^0\,|^2$ links that exist in a # state. So the flow function is evaluated no more than $N * |\,SUB^0\,|^2$ times before it stabilizes. This gives us an upper bound of $O(N * |\,T \times R\,| * |\,SUB^0\,|^5)$ steps for computation of flow#.

Now consider further evolution of the system with create operations permitted. Let A create subject C of type *a*. By the create-rule this operation establishes link(C, A) and also places C/sc in dom(A), as shown in Figure 1c. Now C/sc can be copied from dom(A) to dom(B), and C can demand the ticket B/r. This sets up link(B, C), as shown in Figure 1d. In conjunction with link(C, A) we now have a flow of {*a*/sc} from B to A. By inspection of *f* it is evident that flow can at most be {*a*/sc} so the flow between A and B cannot be increased any further. We call such a state a *maximal state* identified by the symbol *.

The * state is not unique. Indeed there typically are an unbounded number of such states. For example, from the state of Figure 1d the system can continue to

evolve indefinitely by creation of new subjects. The important point is that this can no longer increase the flow between A and B. Our example shows that, in general, $\text{flow}^0(A, B)$ is a subset of $\text{flow}^\#(A, B)$, which, in turn, is a subset of $\text{flow}^*(A, B)$. The fundamental analysis question in SPM is to compute a maximal state. Before we consider this problem, we must first show that maximal states exist.

4.3 EXISTENCE OF MAXIMAL STATES. The concept of maximal state is defined in terms of the initial set of subject. To focus on changes in flow with respect to subjects in $\text{SUB}^0$, we introduce the following notions of reducibility and equivalence.

*Definition* 7. A derivable state h is *0-reducible* to a derivable state g written $h \preceq_0 g$ if and only if

$$(\forall A, B \in \text{SUB}^0)[\text{flow}^h(A, B) \subseteq \text{flow}^g(A, B)].$$

For a given system two derivable states h and g are *equivalent* written $h \equiv_0 g$ if and only if $h \preceq_0 g$ and $g \preceq_0 h$.

Because of its focus on the initial set of subjects, this equivalence relation partitions the derivable states into a finite collection of equivalence classes. For future reference we state this as a lemma.

LEMMA 8. *For every system there is a finite number of equivalence classes of derivable states.*

PROOF. For every pair of subjects in $\text{SUB}^0$, flow can take on at most $|2^{T\times R}|$ distinct values. Hence there are at most $|\text{SUB}^0|^2 * |2^{T\times R}|$ distinct equivalence classes, which is clearly finite. □

We are now ready to formalize the notion of maximal state.

*Definition* 9. For a given system, m is a *maximal state* if and only if m is derivable and for every derivable state h, $h \preceq_0 m$.

Clearly all maximal states are equivalent. The flow function in a maximal state completely defines the potential for copying tickets between subjects present in the initial state.

The existence of maximal states is a consequence of the monotonic nature of state transitions in SPM. Consider a state h in which operation *op* is authorized. If *op* is a demand or copy operation, it continues to be authorized in every state derived from h because the conditions on which the authorization depends cannot be revoked. If *op* is a create operation, the situation is slightly different because each create operation is unique and cannot be repeated. We can account for creates by the formulation: If *op* is authorized in state h, then in every history applied to h either *op* will have occurred or will continue to be authorized. This leads to the following property.

LEMMA 10. *Given an arbitrary finite collection $\mathscr{H}$ of derivable states, there exists a derivable state m such that for every $h \in \mathscr{H}$, $h \preceq_0 m$.*

PROOF. The proof is by induction on size of $\mathscr{H}$. The basis follows by setting $\mathscr{H}$ to $\phi$ and m to the initial state. Assume the lemma holds for all $\mathscr{H}$ of size n and consider $\mathscr{H}$ of size n + 1. Then $\mathscr{H} = \mathscr{G} \cup \{h\}$, where $|\mathscr{G}|$ is n. By induction hypothesis there is a derivable state g that satisfies the lemma for $\mathscr{G}$. For the induction step we show for every pair of derivable states g, h there exists a derivable

state m such that $g \preceq_0 m$ and $h \preceq_0 m$. Let g and h be established by histories G and H, respectively. Let N be any interleaving of G and H that preserves the relative order of the transitions within G and H. Construct M by eliminating the second occurrence of every duplicate create operation in N. That every transition in M is legal follows from the discussion above. Let m be the state established by M. By construction every path in state g, and every path in state h, is duplicated in state m. That completes the induction step and the lemma follows. □

Proving the existence of maximal states is now straightforward.

THEOREM 11. *For every system there exists a maximal state.*

PROOF. By Lemma 8 there is a finite number of equivalence classes of derivable states. Let $\mathscr{K}$ be a collection of derivable states that contains exactly one representative from each equivalence class. The theorem follows by applying Lemma 10 to $\mathscr{K}$. □

This proof is nonconstructive and thereby does not provide a method for computing maximal states. In Section 4.2 we demonstrated that in order to derive a maximal state from the initial state we generally need to create new subjects. The problem is to determine which new subjects need to be created. Our conjecture is that in the general case this is an undecidable problem for SPM and we can only offer approximations [15]. We do have exact solutions in several special cases of interest, one of which we discuss in the remainder of this paper.

## 5. *Maximal States for Acyclic Attenuating Schemes*

In this section we show how to compute a maximal state for systems with acyclic attenuating schemes. The importance of this result is underscored by our conjecture that most, if not all, schemes of practical interest will turn out to have an acyclic attenuating formulation. This conjecture is based on our failure to construct any realistic scheme that cannot be formulated as an acyclic attenuating scheme. In particular, the schemes discussed in Section 3 and in [16] and [18] are all either acyclic attenuating or can be easily reformulated to be acyclic attenuating.

Because the authorization for creates and demands is entirely in terms of types, we can assume without loss of generality that all create operations occur first, followed by demand operations and finally followed by copy operations. We say such histories are *canonical*. Formally we have the following property.

LEMMA 12. *Every derivable state can be derived by a canonical history.*

PROOF. Let history H derive state h. Obtain H′ from H by rearranging the operations in H to conform to the canonical form while preserving the relative order of each kind of operation. Because creates are authorized by types and their relative order is preserved, these operations in H′ are legal. Similarly, the demand operations in H′ are legal. Every copy operation is preceded in H′ by all the operations that preceded it in H and therefore is legal. Thus H′ is a canonical history that derives h. □

The most troublesome aspect in deriving a maximal state is the create operation. At the same time, the no-creates maximal state is trivially computed. Our objective is to reduce the former problem to the latter one. By Lemma 12 we know that all create operations can be assumed to occur at the beginning of a history, so, in particular, a maximal state can be derived by such a history. The real problem then is to determine the initial sequence of creates needed for this purpose.

Our strategy for computing a maximal state is as follows. From the given initial state we first derive a state u entirely by create operations, with the objective that entities in state u will account for all possible entities that can exist. We achieve this by defining a mapping $\sigma$, read surrogate, from all possible entities to entities in state u such that entity A is simulated by $\sigma(A)$. In particular, $\sigma$ maps every entity present in the initial state to itself. In our proof we show that for every history H that derives h from the initial state there exists a history G, *without create operations*, that derives g from u such that $\text{flow}^h(A, B) \subseteq \text{flow}^g(\sigma(A), \sigma(B))$. Since G has no creates, the maximal state for the given system is the no-creates maximal state that results from u as the initial state. If the construction of u introduces only a polynomial number of new subjects, the entire computation can be done in polynomial time. The remainder of this section elaborates and formalizes this argument in context of acyclic attenuating schemes.

5.1 THE SURROGATE FUNCTION. For an acyclic scheme, by definition *cc* is acyclic. For the moment assume *cc* contains no loops. Consider a subject A of type *a*. We say A is *unfolded* if A creates one entity of each type *b* such that *cc*(*a*, *b*). For each *b* the entity created in this manner is called the *b-surrogate* of A. The idea is that the *b*-surrogate of A will simulate all type *b* children of A. To account for descendants of A's children, we apply the unfolding construction recursively to the surrogates of A, and so on, until all descendants of A are unfolded. Because *cc* is acyclic and without loops, this construction eventually terminates. At this point we say A is *fully unfolded.* The initial state is fully unfolded if all subjects in $\text{SUB}^0$ are fully unfolded.

If *cc* contains loops, we first eliminate the loops and fully unfold the initial state. Then, for every A in the resulting state such that $cc(\tau(A), \tau(A))$, we let A create A′ of type $\tau(A)$. The intention here is that the $\tau(A)$ children of A will be simulated by A itself rather than by A′. Why then create A′ at all? The reason for this goes back to the motivation underlying our attenuating restriction on create-rules for loops in *cc*; that is, it is possible this create operation may generate copiable control tickets for A in dom(A).

We now formally define this construction.

*Definition* 13. Given any initial state 0 with an acyclic attenuating scheme derive the *fully unfolded state* u as follows:

(1) Let $cc' = cc - \{\langle a, a\rangle \mid a \in TS\}$.
(2) Mark all subjects in $\text{SUB}^0$ as folded.
(3) **While** there exists a folded subject A **do**
    Mark A as unfolded
    **For** all *b* such that $cc'(\tau(A), b)$ **do**
      Let A create B of type *b*
      Call B the *b*-surrogate of A
      **If** B is a subject mark **then** mark it as folded
(4) **For** all subjects A in the resulting state **do**
    **If** $cc(\tau(A), \tau(A))$ **then** let A create a subject of type $\tau(A)$

Clearly each create operation in this construction is authorized by *cc*, so u is a derivable state. Because of the absence of cycles and loops in *cc*′, this construction is guaranteed to terminate.

LEMMA 14. *The construction of Definition* 13 *terminates.*

PROOF. We need to show that step 3 terminates. Consider $A \in SUB^0$. The descendants of A generated by step 3 form a tree with A at the root and each created entity a child of its creator. Because each subject creates only one child of each type, each node in the tree has a finite number of children. If we follow a path in this tree from the root to any of A's descendants, the types of entities encountered in this path must all be different; otherwise $cc'$ contains a cycle or loop. Since all nodes in this path, except the last one, must be subjects, the maximum length of such a path is $|TS| + 1$. Hence the depth of the tree is finite. □

Next we define a mapping to relate each entity that can be created to the entity in state u that simulates it.

*Definition* 15. Given any initial state with an acyclic attenuating scheme, for every derivable state h define the *surrogate* function $\sigma$: $ENT^h \rightarrow ENT^u$ as follows:

(1) If $A \in ENT^0$, then $\sigma(A) = A$.
(2) If A creates B and $\tau(A) \neq \tau(B)$, then $\sigma(B) = \tau(B)$-surrogate of $\sigma(A)$.
(3) If A creates B and $\tau(A) = \tau(B)$, then $\sigma(B) = \sigma(A)$.

It is evident that $\sigma$ preserves types; that is, $\tau(\sigma(A)) = \tau(A)$.

We demonstrate the construction of Definition 13 in Figure 2a. Let $cc(a) = \{a, b\}$ and $cc(b) = \{b\}$. Let A be a subject of type $a$ in the initial state. In constructing u, A creates a child B′ of type $b$. B′ is the $b$-surrogate of A, while A itself is the $a$-surrogate for A. Also A creates A′ of type $a$ and B′ creates B″ of type $b$. Each edge in Figure 2a connects a created subject and its creator. Now assume that the actual creates that take place are as shown in Figure 2b, where all $A_i$'s are of type $a$ and all $B_i$'s of type $b$. By Definition 15 we have the following values for $\sigma$:

$$\sigma(A_1) = \sigma(A_2) = \sigma(A_3) = \sigma(A_4) = \sigma(A) = A,$$
$$\sigma(B_1) = \sigma(B_2) = \sigma(B_3) = \sigma(B_4) = \sigma(B_5) = B'.$$

Thus A will simulate itself and all the $A_i$'s while B′ will simulate all the $B_i$'s.

Let us see how the create operations of Figure 2b are simulated by the create operations of Figure 2a. The creation of $B_1$ and $B_2$ by A are both simulated by the creation of B′. Now both $A_1$ and $A_2$ are mapped to A by $\sigma$. This indicates that the creation of $A_1$ and $A_2$ should be simulated by A creating itself. Although this is somewhat curious, we can indeed pretend that A creates itself in Figure 2a. To do so, we must show that all tickets required by $cr(a, a)$, with A playing the role of the creator and the created subject, are present in dom(A). This follows from the fact that $cr(a, a)$ is attenuating. When A creates A′, by Definition 2 whatever A′ gets, A also gets. Moreover whatever A gets for A′, A also gets for itself. But this can be interpreted as A getting the tickets A/x:c for $a$/x:c or *self*/x:c in *LEFT* ∪ *RIGHT* of $cr(a, a)$. Thus we can pretend that A creates itself.

This leads to the following property.

LEMMA 16. *For a system with an acyclic attenuating scheme, if A creates B, then tickets that would be introduced by pretending that $\sigma(A)$ creates $\sigma(B)$ are present in $dom^u(\sigma(A))$ and $dom^u(\sigma(B))$.*

PROOF. If $\tau(A) \neq \tau(B)$, then $\sigma(A)$ indeed creates $\sigma(B)$ in constructing u from the initial state (step 3 of Definition 13). If $\tau(A) = \tau(B)$, then $\sigma(A) = \sigma(B)$. In constructing u then $\sigma(A)$ creates A′ of type $\tau(A) = \tau(\sigma(A))$ (step 4 of Definition 13). By Definition 2 all tickets that would be introduced by pretending that $\sigma(A)$ creates itself are thereby present in $dom^u(\sigma(A))$. □

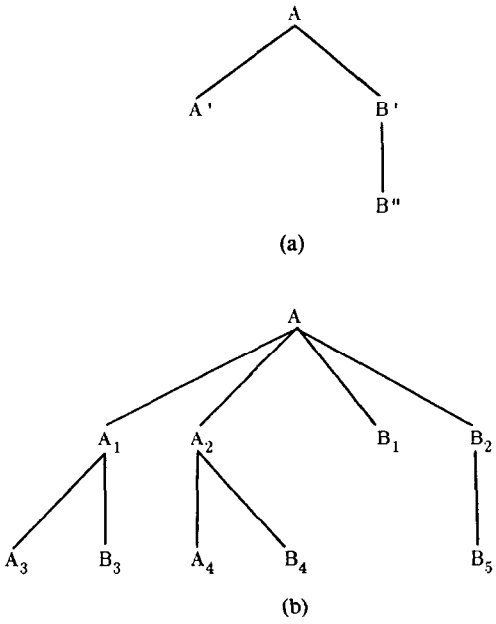


FIG. 2. The surrogate function: (a) appending surrogates to A; (b) the actual descendants of A.

Lemma 16 is crucial because it suggests that in constructing the fully unfolded state u we have managed to account for all possible create operations.

5.2 MAXIMAL STATES. We are now ready to prove the central result of this paper. Our objective is to show that every history for a given system can be simulated by a history without create operations applied to the fully unfolded state of Definition 13.

THEOREM 17. *For a system with an acyclic attenuating scheme, for every history H that derives h from the initial state there exists a history G,* without create operations, *that derives g from the fully unfolded state u such that*

$$(\forall A, B \in SUB^h)[flow^h(A, B) \subseteq flow^g(\sigma(A), \sigma(B))].$$

PROOF. By Lemma 12 we may assume that H is in canonical form, that is, that all create operations occur first, followed by demand operations and then by copy operations. G is obtained from H by replacing the individual transitions of H as follows, while preserving the relative order:

(1) Ignore all create operations.
(2) Replace "A demands B/x:c" by "$\sigma$(A) demands $\sigma$(B)/x:c".
(3) Replace "copy A/x:c from B to C" by "copy $\sigma$(A)/x:c from $\sigma$(B) to $\sigma$(C)".

We first establish the following assertions.

(I) Every transition in G is legal.
(II) $A/x:c \in dom^h(B) \Rightarrow \sigma(A)/x:c \in dom^g(\sigma(B))$.
(III) For every i, $link_i^h(A, B) \Rightarrow link_i^g(\sigma(A), \sigma(B))$.

Assertion III follows trivially from II and is crucial to the second part of the proof. Assertions I and II are proved by induction on the number of copy operations in H.

*Basis case.* Let there be no copy operations in H, so H consists of creates followed by demands while G consists entirely of demands.

*Assertion I.* By construction every operation "A demands B/x:c" in H is replaced by "$\sigma$(A) demands $\sigma$(B)/x:c" in G. Since $\sigma$ preserves types, the demand operation in G is legal.

*Assertion II.* Without copy operations there are only three ways by which A/x:c can appear in $\text{dom}^h(B)$. If A/x:c $\in \text{dom}^0(B)$, then $\sigma(A) = A$ and $\sigma(B) = B$; so Assertion II is trivially true. If A/x:c $\in \text{dom}^h(B)$ because of a create operation in H, Assertion II follows from Lemma 16. Finally, if A/x:c $\in \text{dom}^h(B)$ because of a demand operation, then Assertion II follows from the corresponding demand operation in G.

*Induction step.* Assume Assertions I and II are true for every history with k copy operations and consider a history H with k + 1 copy operations. Since H is in canonical form, it consists of an initial sequence H′, with k copy operations followed by a single copy operation. Let h′ be the state derived by H′. Let G′ be the required modification of H′. By induction hypothesis and assertion I, G′ is a history. Let g′ be the state derived by G′. Let the final operation of H be "copy A/x:c from B to C". By construction the final operation of G is "copy $\sigma$(A)/x:c from $\sigma$(B) to $\sigma$(C)".

*Assertion I.* For the final operation of H to be legal the following conditions must be true for some i:

(1) A/xc $\in \text{dom}^{h'}(B)$.
(2) $\text{link}_i^{h'}(B, C)$.
(3) $\tau(A/x:c) \in f_i(\tau(B), \tau(C))$.

By induction hypothesis and Assertion II it follows that the first two conditions above imply, respectively, that

(1) $\sigma(A)/xc \in \text{dom}^{g'}(\sigma(B))$,
(2) $\text{link}_i^{g'}(\sigma(B), \sigma(C))$.

Since $\sigma$ preserves types it follows from the third condition above that

(3) $\tau(\sigma(A)/x:c) \in f_i(\tau(\sigma(B)), \tau(\sigma(C)))$.

So the three conditions required to authorize the final operation of G are true in state g′ and the final operation in G is legal.

*Assertion II.* h differs from h′ at most by A/x:c $\in \text{dom}^h(C)$. By construction the final operation of G ensures that $\sigma(A)/x:c \in \text{dom}^g(\sigma(C))$. This completes the induction step.

It remains to prove that $(\forall A, B \in \text{SUB}^h)[\text{flow}^h(A, B) \subseteq \text{flow}^g(\sigma(A), \sigma(B))]$. We do so by showing that for every $\text{path}^h$ from A to B there is a $\text{path}^g$ from $\sigma$(A) to $\sigma$(B) with the same capacity as the $\text{path}^h$ from A to B. The proof is by induction on the number of links. For the basis case consider a $\text{path}^h$ from A to B of length 1; that is, $\text{link}_i^h(A, B)$. By Assertion III we have $\text{link}_i^g(\sigma(A), \sigma(B))$. Since $\sigma$ preserves types, the basis case is true. Assume the hypothesis is true for every $\text{path}^h$ of length k and consider a $\text{path}^h$ from A to B of length k + 1. Then there is some C with a $\text{path}^h$ from A to C of length k and $\text{link}_j^h(C, B)$. By induction hypothesis there is a $\text{path}^g$ from $\sigma$(A) to $\sigma$(C) with the same capacity as the $\text{path}^h$ from A to C. By Assertion III we have $\text{link}_j^g(\sigma(C), \sigma(B))$. Since $\sigma$ preserves types, it follows there is a $\text{path}^g$ from $\sigma$(A) to $\sigma$(B) with the same capacity as the $\text{path}^h$ from A to B. □

The essence of Theorem 17 is that all histories applied to the initial state can be simulated by histories without create operations applied to the fully unfolded state u. Let #u be the no-creates maximal state that results from u as the initial state. We have the following corollary.

COROLLARY 18. *For a system with an acyclic attenuating scheme #u is a maximal state.*

PROOF. From Theorem 17 and definition of #u, for every history H that derives state h from the initial state

$$(\forall A, B \in SUB^h)[flow^h(A, B) \subseteq flow^{\#u}(\sigma(A), \sigma(B))].$$

In particular, $(\forall A, B \in SUB^0)[flow^h(A, B) \subseteq flow^{\#u}(A, B)]$, so $h \preceq_0 \#u$. □

Note that $flow^{\#u}$ also provides a bound on the flow involving subjects created subsequent to the initial state in terms of their surrogates. That is,

(1) For $A, B \in SUB^0$, $flow^*(A, B) = flow^{\#u}(A, B)$.
(2) For $A, B \in SUB^h - SUB^0$, $flow^*(A, B) \subseteq flow^{\#u}(\sigma(A), \sigma(B))$.

If we wish to compute flow*(A, B) more precisely in the second case, we can apply the unfolding construction to a state in which A and B have been created.

Clearly the #u state is derivable. To derive #u from u requires time polynomial in $|SUB^u|$. For each subject $A \in SUB^0$ the construction of u from the initial state introduces a constant number of new subjects determined by $\tau(A)$. Thus the entire computation is polynomial in $|SUB^0|$. We take this as evidence that the computation is tractable. We mention that $|SUB^u|$ may exceed $|SUB^0|$ by a factor exponential in $|TS|$. In the worst case the straightforward algorithms for computing $flow^{\#u}$ will then be exponential in $|TS|$. This will happen only if *cc* is highly nonsparse. At any rate this exponential factor involves $|TS|$ rather than $|SUB^0|$ and may be tolerable.

## 6. *Conclusion and Discussion*

This paper has focused on the problem of balancing generality and analyzability in a protection model. We defined the Schematic Protection Model (SPM) with the key idea of strong typing of entities. We demonstrated that for acyclic attenuating schemes analysis of systems specified in SPM is tractable. The importance of this result is underscored by our conjecture that most, if not all, schemes of practical interest will turn out to have an acyclic attenuating formulation. This conjecture is based on our failure to construct any realistic scheme that cannot be formulated as an acyclic attenuating scheme. In particular, the schemes discussed in Section 3 and in [16] and [18] are all either acyclic attenuating or can be easily reformulated to be acyclic attenuating.

SPM offers a significant advantage over the access matrix from a design viewpoint. SPM provides a "high-level" structure in contrast to the "low-level" structure of the access matrix. The much richer structure of SPM makes for more convenient specification of policies. Even without formal analysis this makes it easier to formulate and understand a SPM specification of a dynamic authorization policy.

In comparing the generality of SPM and the access-matrix model of Harrison et al., [5], one obvious difference is the lack of revocation in SPM. We have bypassed the issue of revocation in SPM by appealing to the restoration principle that allows only those revocation policies in which revocation itself can be undone. This approach has the advantage that our results will not depend on the correct

working of a revocation mechanism. At any rate, SPM must be compared with the access matrix without revocation, that is, the monotonic access matrix [4]. Any SPM scheme can be expressed in the latter formalism by specifying the rules of the scheme for the copy, create, and demand operations by a straightforward construction. On the other hand, in the access-matrix formulation the copy flag is not necessary for a copy operation. Also the authorization for a copy operation can be made to depend on tickets outside the domains of the two subjects in question. So it appears that the access-matrix formulation is able to express directly rules that contradict the spirit of SPM. But there may be ways of stating the same rules in some indirect manner in SPM. The exact relationship between SPM and the monotonic access matrix appears to be a difficult question for which we do not have a precise answer as yet.

In comparison with the take–grant model, SPM is much more general, as evident from Section 3. We have shown that the take–grant model can be specified as a particular SPM scheme. We note that SPM is an outgrowth of the author's earlier work on the Schematic Send–Receive (SSR) model [15–17]. SSR itself is based on Minsky's send–receive transport model [13]. SSR adopts some simplifying assumptions and has a set-theoretic formulation in contrast to Minsky's formulation, which has a production-rule flavor.

The analysis developed in this paper is based on the worst-case concept of maximal state. In practice it is desirable to analyze systems under assumptions about the behavior of specific subjects. For instance in the departmental example of Scheme IV, essentially any state is safe because of the discretionary power of the department head. It would be interesting to analyze such a system under the assumption that the department head does not exercise his or her discretionary power to give access to internal documents to outsiders. In SPM, analysis with such behavioral assumptions can be reduced to worst-case analysis. If the department head will not exercise his or her discretionary power, we may assume that the value of $f(head, out)$ is $\phi$ rather than $\{idoc/x\}$. SPM, being a model rather than a mechanism, does not insist that behavioral restrictions built into a scheme be enforced at run time. Any restriction imposed by the scheme can be implemented by one or more of the following options:

(1) Enforce the restriction at run time.
(2) Assume subjects will honor the restriction.
(3) Prove subjects will honor the restriction.

The second alternative allows us to incorporate behavioral assumptions as part of a scheme.

In this paper we have been primarily concerned with the analysis aspect of SPM. Regarding specification, we demonstrated a few simple examples. More extensive case studies are discussed in [16] and [18]. Further work is needed in this area. In particular the SPM formulation of a policy is not unique. We saw this in Section 3.3 where the take–grant model was specified first as an acyclic nonattenuating scheme and then as an acyclic attenuating scheme. We need some formal understanding of what it means for two specifications to be equivalent. We also need a methodology for developing a specification in SPM. Finally even though we have chosen to bypass the issue of specifying policies for revocation and deletion by appealing to the restoration principle for analysis purposes, we do need a formalism for specifying these policies.

In addition to specification and analysis of a policy, there is the all important issue of implementation. The goal regarding implementation is that a protection

model should permit a variety of implementations with attendant trade-offs for a given policy. That there are alternate specifications for the same policy in SPM is a possible advantage of SPM, since the specification can then be tailored to suit a particular mechanism. There is also the question of designing a mechanism that implements the SPM framework in toto. In a tightly coupled environment such a mechanism should be no more complex than some of the existing capability-based systems [9]. We foresee no conceptual hurdles in building such a mechanism. Of course, plenty of details need to be worked out and our beliefs must be validated experimentally. Extending the mechanism to a loosely coupled distributed environment is a more challenging endeavor.

ACKNOWLEDGMENT. Although the model has undergone substantial generalization, the basic ideas were developed in the author's Ph.D. dissertation at Rutgers University. The author takes this opportunity to acknowledge the vital guidance of his thesis advisor Professor Naftaly Minsky. Also Professor Ann Yasuhara, a member of the thesis committee, contributed substantially to the clarity and correctness of the work. Finally the author is indebted to the referees for several valuable suggestions.

REFERENCES

1. DENNING, D. E. *Cryptography and Data Security*. Addison-Wesley, Reading, Mass., 1982.
2. DENNIS, J. B., AND VAN HORN, E. C. Programming semantics for multiprogrammed computations. *Commun. ACM 9*, 3 (Mar. 1966), 143–155.
3. GRAHAM, G. S., AND DENNING, P. J. Protection–Principles and practices. In *Proceedings of AFIPS Spring Joint Computer Conference*, vol. 40. AFIPS Press, Reston, Va. 1972, pp. 417–429.
4. HARRISON, M. A., AND RUZZO, W. L. Monotonic protection systems. In *Foundations of Secure Computation*. R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, Eds. Academic Press, Orlando, Fla., 1978.
5. HARRISON, M. A., RUZZO, W. L., AND ULLMAN, J. D. Protection in operating systems. *Commun. ACM 19*, 8 (Aug. 1976), 461–471.
6. JONES, A. K., LIPTON, R. J., AND SNYDER, L. A linear time algorithm for deciding security. In *Proceedings of the 17th Symposium on the Foundations of Computer Science* (Houston, Tex., Oct. 25–27). IEEE Computer Society Press, Silver Spring, Md., 1976, pp. 33–41.
7. LAMPSON, B. W. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information and System Sciences* (Princeton, N.J., Mar.). Princeton Univ., Princeton, N.J., 1971, pp. 437–443.
8. LANDWEHR, C. E. Formal models for computer security. *ACM Comput. Surv. 13*, 3 (Sept. 1981), 247–275.
9. LEVY, H. M. *Capability-Based Computer Systems*. Digital Press, Bedford, Mass., 1984.
10. LINDEN, T. A. Operating system structures to support security and reliable software. *ACM Comput. Surv. 8*, 4 (Dec. 1976), 409–445.
11. LIPTON, R. J., AND SNYDER, L. A linear time algorithm for deciding subject security. *J. ACM 24*, 3 (July 1977), 455–464.
12. LOCKMAN, A., AND MINSKY, N. Unidirectional transport of rights and take-grant control. *IEEE Trans. Softw. Eng. SE-8*, 6 (Nov. 1982), 597–604.
13. MINSKY, N. H. Selective and locally controlled transport of privileges. *ACM Trans. Program. Lang. Syst. 6*, 4 (Oct. 1984), 573–602.
14. SALTZER, J. H., AND SCHROEDER, M. D. The protection of information in computer systems. *Proc. IEEE 63*, 9 (Sept. 1975), 1278–1308.
15. SANDHU, R. S. Design and analysis of protection schemes based on the send-receive transport mechanism. Tech. Rep. DCS-TR-130, Rutgers Univ., New Brunswick, N.J., 1983.
16. SANDHU, R. S. The SSR model for specification of authorization policies: A case study in project control. In *Proceedings of the 8th International Computer Software and Applications Conference* (Chicago, Ill., Nov. 7–9). IEEE Computer Society Press, Silver Spring, Md., 1984, pp. 482–491.
17. SANDHU, R. S. Analysis of acyclic attenuating systems for the SSR protection model. In *Proceedings of the 1985 IEEE Symposium on Security and Privacy* (Oakland, Calif., Apr. 22–24). IEEE Computer Society Press, Silver Spring, Md., 1985, pp. 197–206.

18. SANDHU, R. S., AND SHARE, M. E. Some owner based schemes with dynamic groups in the schematic protection model. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy* (Oakland, Calif., Apr. 7–9). IEEE Computer Society Press, Silver Spring, Md., 1986, pp. 61–70.
19. SNYDER, L. Formal models of capability-based protection systems. *IEEE Trans. Comput. C-30*, 3 (Mar. 1981), 172–181.

RECEIVED JUNE 1985; REVISED APRIL 1987; ACCEPTED OCTOBER 1987