Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 44, No. 1, February 1992
Printed in Belgium

Undecidability of Safety for the Schematic Protection Model with Cyclic Creates

RAVINDERPAL SINGH SANDHU

Department of Information and Software Systems Engineering, George Mason University, Fairfax, Virginia 22030-4444

Received October 19, 1988; revised February 20, 1990

In the schematic protection model subjects are classified into protection types. Creation is authorized by a can-create binary relation on types. It is shown that with arbitrary cycles in can-create safety is undecidable. Whereas it has been previously shown safety is decidable for acyclic can-create. It is also shown that safety remains undedicable even if all creates are attenuating in that tickets (capabilities) given to a subject on its creation are attenuated copies of tickets available to its parent. This contrasts with decidable safety for attenuating cycles of length one. It appears safety is decidable for the practically useful cases while undecidability results from undue laxity in authorizing creation. © 1992 Academic Press, Inc.

1. INTRODUCTION

The need for access controls or protection arises in any computer system in which multiple users share information and physical resources. These systems are viewed as consisting of *subjects* and *objects*. Active entities such as users are subjects, whereas passive entities such as text files are objects. Protection is enforced by ensuring that subjects can execute only those operations which are authorized by *privileges* in their *domains*. We regard subjects and objects as mutually exclusive and use *entity* to denote either a subject or object. The key difference is that subjects possess priviliges whereas objects do not.[1]

The *protection state* of a system is defined by the privileges possessed by subjects in their domains at a given moment. We understand state to mean protection state. *Inert privileges* authorize operations which do not modify the state, e.g., reading a file. *Control privileges* authorize operations which modify the state, e.g., user X authorizes user Y to read file Z. Control privileges authorize incremental changes in the protection state and define the dynamics of authorization. Once the initial state is established the protection state evolves due to autonomous actions of subjects constrained by control privileges. The challenge is to construct the initial state so all reachable states conform with the policy the security administrator wishes to implement.

[1] Subjects are often defined to be subset of objects. This amounts to calling what we call entities as objects and coining some other term for entities which are not subjects.

0022-0000/92 $3.00

Copyright © 1992 by Academic Press, Inc.
All rights of reproduction in any form reserved.

A useful *protection model* must provide a formalism for specifying the dynamics of the protection state. This is usually done by stating rules which prescribe the authorization for making localized incremental changes in the state. We call such a collection of rules an (*authorization*) scheme. The scheme embodies the policy for dynamic authorization. In order to understand the formal specification of a scheme it must be possible to determine the cumulative global effect of authorized incremental changes in the protection state. That is, given the initial protection state and the authorization scheme, we need to characterize protection states that are reachable. The *safety problem* poses the question: Is there a reachable state in which some subject possesses a specific privilege which it did not previously possess? It is the fundamental analysis question which a protection model must confront. Since subjects are usually authorized to create new subjects and objects, the system is unbounded and it is not certain that such analysis will be decidable let alone tractable without sacrificing generality.

Analysis issues were first formalized by Harrison, Ruzzo, and Ullman [2] in context of the well-known access matrix model [1, 5]. The matrix has a row for each subject and a column for each entity. The (I, J) cell contains symbols called *rights* authorizing subject I to perform operations on entity J. In this model an authorization scheme is defined by a set of commands. Each command has a condition part and a body. The condition specifies rights required to exist in the matrix before the body can be executed for its actual arguments. The body consists of a sequence of primitive operations. The primitive operations enter or delete a right from a cell of the matrix or create a new row or column or destroy an existing row or column. A right is said to be *leaked* if it can be entered into a cell of the access matrix where it did not previously exist. The safety problem poses the question whether or not some specific right can be leaked. Harrison, Ruzzo, and Ullman showed that this problem is undecidable in general [2]. It was further shown by Harrison and Ruzzo [3] that safety remains undecidable even if the condition can test for rights in at most two cells of the matrix and the primitive operations are restricted to be monotonic.[2]

In retrospect it is not too surprising that analysis is undecidable in this general setting. More disappointing is the lack of interesting special cases of the access matrix model with tractable safety, as evident from the following quote [2].

> It would be nice if we could provide for protection systems an algorithm which decided safety for a wide class of systems, especially if it included all or most of the systems that people seriously contemplate. Unfortunately, our one result along these lines involves a class of systems called "mono-operational," which are not terribly realistic. Our attempts to extend these results have not succeeded, and the problem of giving a decision algorithm for a class of protection systems as useful as the $LR(k)$ class is to grammar theory appears very difficult.

In response to this situation we proposed the *schematic protection model* (SPM) to balance the inherently conflicting goals of generality versus tractable safety analysis

[2] That is, it is possible to enter a right in a cell or create new subjects and objects, but deletion of a right from a cell or destruction of subjects and objects is not permitted.

[14]. SPM classifies subjects and objects into protection types. The dynamic component of a protection state consists of tickets (capabilities). The key idea is that the rules comprising the authorization scheme are specified in terms of protection types. In particular creation is authorized by a can-create binary relation on types. It has been previously shown that analysis is decidable provided the can-create relation is acyclic [14]. In this paper we show that with arbitrary cycles in can-create safety becomes undecidable. This gives us a natural demarcation between decidable and undecidable safety in SPM.

Undecidability results are disappointing since they reflect inherent limitations. But in this case our disappointment is mitigated by the conjecture that most, if not all, SPM specifications of practical interest will satisfy the constraints of [14]. This is demonstrated by the examples of [12–14] and our failure to find any realistic policy which cannot fit within the assumptions of [14]. It appears that decidability is obtained for the most useful cases while undecidability follows from undue laxity in authorizing subject creation, which can be easily prevented.

The paper is organized as follows. Section 2 reviews SPM. Section 3 shows that safety is undecidable with arbitrary cycles in can-create. The proof is by reduction from Post's correspondence problem. Use of Post's problem for this purpose was motivated by its successful application in showing undecidable safety for the monotonic access-matrix [3]. Our construction is quite intricate because of the local nature of authorization rules in an SPM scheme. In Section 4 we show that safety remains undecidable even if all create operations are required to be attenuating. Roughly speaking this requirement stipulates that tickets given to a created subject on creation should be attenuated copies of tickets available to the creator. This is a natural restriction, the significance of which is highlighted by the fact that with attenuating loops (cycles of length one) safety is decidable [14]. Section 5 concludes the paper.

2. The Schematic Protection Model

The key notion in SPM is that of protection types, henceforth referred to simply as types. The domain of a subject consists of two parts: a static type-dependent part defined by the authorization scheme and a dynamic part consisting of tickets. The intuitive concept of types is that instances of the same type are treated uniformly in the authorization scheme. The scheme is defined 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. SPM entities are strongly typed; that is, an entity's type cannot change.

Tickets are dynamic privileges of the form U/x, where U identifies some unique entity and the right symbol x authorizes the possessor of this ticket to perform some operation(s) on U. Tickets can only be obtained by rules specified in the authorization scheme. We use the neutral term ticket rather than capability to

avoid the impression that SPM tickets are necessarily represented at run-time as capabilities. SPM tickets have only one right symbol. For convenience we abbreviate a set of tickets for the same entity by letting U/xyz denote $\{U/x, U/y, U/z\}$. We understand U/x to denote the ticket U/x as well as the set $\{U/x\}$ as determined by context. This allows us, for instance, to write $\{U/x, V/yz\}$ or $U/x \cup V/yz$ interchangeably.

2.1. *Types and Right Symbols*

The first step in defining a scheme is to specify disjoint sets of object types TO and subject types TS. Their union T is the entire set of entity types. The intention is that protection types identify classes of entities which have common properties with respect to the authorization policy. For subjects this might be membership in a department or a distinguished position of authority in a group such as project leader. For objects this might be a classification such as an internal document or a public document. By convention types are named in lowercase and entities in uppercase. The type of an entity U is denoted by $\tau(U)$. An entity of type u is often referred to as an u entity.

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 passive role of inert rights with respect to the protection state, the symbols in RI require no interpretation for safety analysis. 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 U/x and U/xc is that the former cannot be copied from one domain to another, whereas the latter may be, provided certain additional conditions to be defined shortly are true. It follows that the presence of U/xc in a subject's domain subsumes the presence of U/x but not vice versa. We use $x:c$ to signify 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. We understand U/xyc to denote U/xc and U/yc; that is, the copy flag applies to each symbol in the string. If the copy flag occurs in the middle of a string it applies to privileges to the left of it but not those to the right; for instance, $U/xcyz$ denotes U/xc and U/yz.

The type of a ticket $U/x:c$ is written as $\tau(U/x:c)$ and we define it to be the ordered pair $\tau(U)/x:c$. That is, the type of a ticket is determined by the type of 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 $\tau(U/x)$ and $\tau(U/xc)$ are different ticket types. This is an important distinction because of the role of the copy flag. The entire set of *ticket types* is $T \times R$.

T, R, and $T \times R$ constitute the basic sets of an authorization scheme. The remaining components are defined in terms of functions and relations involving these basic sets. SPM requires that T and R be finite, so a scheme is defined by finite sets, relations, and functions. Before considering the details we note SPM is monotonic

in that there are no facilities for revocation of tickets or deletion of entities. This is a reasonable assumption for analysis purposes in most cases, by accepting the restoration principle [14]. This principle requires that whatever can be revoked can be restored; i.e., revocation can always be undone. Then for the worst case we can assume revocation does not occur. In this way SPM side steps the issue of specifying revocation policies.

In SPM there are three operations which change the protection state: *cypy*, *demand*, and *create*. Demand is not used in the construction of this paper and is mentioned here only for the sake of completeness. Demand in now actually known to be formally redundant [15].

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 $link_i$ defined by control rights and its associated filter function f_i which is a component of the scheme.

A link predicate takes two subjects, say U and V, as arguments and evaluates to true or false. If true it establishes a connection from U to V which can be used to copy tickets from the domain of U to the domain of V. Link predicates are defined in terms of the presence of some combination of control tickets for U and V in the domains of U and V. Formally a link predicate $link_i(U, V)$ is a conjunction or disjunction, but not negation, of terms from the following collection for $x \in RC$

$$U/x \in \mathrm{dom}(U),\ U/x \in \mathrm{dom}(V),\ V/x \in \mathrm{dom}(U),\ V/x \in \mathrm{dom}(V) \text{ or } \textbf{true},$$

where $\mathrm{dom}(U)$ is the set of tickets possessed by subject U. The idea is that link predicates are evaluated by examining the domains of the two subjects of concern and that too only with respect to presence of control tickets for these two subjects. To emphasize this property we say link predicates are local. That the definition of a link should depend only on presence and not absence of tickets is a well-known principle for protection [10]. In SPM a finite collection of local link predicates are defined in a scheme. Some examples of these are given below.

1. The take–grant link of [4, 7, 16] defined as $V/g \in \mathrm{dom}(U) \vee U/t \in \mathrm{dom}(U)$.
2. The take and grant links of [8] defined as $U/t \in \mathrm{dom}(U)$ and $V/g \in \mathrm{dom}(U)$, respectively.
3. The send–receive link of [9, 11] defined as $V/s \in \mathrm{dom}(U) \wedge U/r \in \mathrm{dom}(V)$.
4. The broadcast link of [14] defined as $U/b \in \mathrm{dom}(U)$.
5. The universal link of [14] defined as **true**.

For a given state, if $link_i(U, V)$ is true we say there is a $link_i$ from U to V. We emphasize that existence of a link is necessary but not sufficient for copying tickets

The final condition required for authorizing a copy operation is specified by a filter function f_i: $TS \times TS \rightarrow 2^{T \times R}$ for each predicate $link_i$. The interpretation is that $Y/x : c$ can be copied from dom(U) to dom(V) if and only if all of the following are true for some $link_i$, where the types of U, V, and Y are u, v, and y, respectively:

$$Y/xc \in \text{dom}(U) \wedge link_i(U, V) \wedge y/x : c \in f_i(u, v).$$

Note that Y/xc is required in dom(U) whether we are authorized to copy Y/xc or Y/x by the filter function. In this manner the copy flag, link predicates, and filter functions together authorize copying. We emphasize there is a different filter function for each link predicate.

Filter functions are a powerful tool for specifying policies. They impose mandatory controls which are inviolable and constrain the discretionary behavior of individual subjects. Some sample values for $f_i(u, v)$ are $T \times R$, $TO \times RI$, and ϕ. SPM imposes no assumptions regarding the role of U and V in a copy operation from U to V. It is equally acceptable that copying take place at the initiative of U or V alone or require both to cooperate. This is consistent with a worst-case scenario for analysis in which complete cooperation between subjects is assumed.

2.3. *The Create Operation*

The create operation introduces new subjects and objects in 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? 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 u are authorized to create entities of type v if and only if $cc(u, v)$. It is often convenient to regard cc as a function cc: $TS \rightarrow 2^T$.

The tickets introduced by a create operation are specified by a create-rule cr for every pair in cc. SPM create-rules are local in that the only tickets introduced are for the creating and created entities in the domains of these two entities. The motivation is that creation should immediately have only a local incremental impact on the state. We emphasize there is a different create-rule for each pair in cc.

Let subject U of type u create entity V of type v, so U is the parent and V the child. If V is an object the only tickets that can be introduced by the crteate-rule are inert tickets for V in U's domain. So if v is an object type the create-rule $cr(u, v)$ is specified as a subset of $\{child/x : c \mid x : c \in RI\}$, where *child* is a special symbol. The interpretation is that U gets $V/x : c$ if and only if $child/x : c \in cr(u, v)$. If V is a subject the create-rule must also specify tickets to be placed in V's domain. So if v is a subject type the create-rule $cr(u, v)$ has two components $cr_p(u, v)$ and $cr_c(u, v)$ which respectively specify tickets to be placed in the parent and child domains. Tickets for the parent and child are identified by the special symbols *parent* and *child*, respectively. That is $cr_p(u, v)$ and $cr_c(u, v)$ are subsets of $\{parent/x : c, child/x : c \mid x : c \in R\}$. The interpretation[3] is that parent U gets $U/x : c$, provided

[3] Slightly different interpretations of the create-rules are discussed in Section 4.

parent/$x : c \in cr_p(u, v)$, and gets $V/x : c$, provided *child*/$x : c \in cr_p(u, v)$. Similarly the child V gets $U/x : c$, provided *parent*/$x : c \in cr_c(u, v)$, and $V/x : c$, provided *child*/$x : c \in cr_c(u, v)$.

2.4. *Summary of SPM*

In summary SPM requires the security administrator to specify an authorization scheme by defining the following finite components.

1. A set of entity types T partitioned into subject types TS and object types TO.
2. A set of right symbols R partitioned into inert rights RI and control rights RC.
3. A collection of local link predicates $\{link_i\}$.
4. A filter function f_i: $TS \times TS \to 2^{T \times R}$ for each predicate $link_i$.
5. A can-create relation $cc \subseteq TS \times T$. Equivalently, cc: $TS \to 2^T$.
6. A local create-rule for each pair in cc.

A *system* is specified by defining an authorization scheme and the initial protection state, i.e., the initial set of entities and the initial distribution of tickets. Thereafter the protection state evolves by copy and create operations. In SPM we say a right x is leaked if a ticket with the x right can be entered in a domain where it previously did not exist. The safety problem poses the question whether or not a particular right x can be leaked.

3. Undecidability of Safety for SPM

Let $x[1] \leftrightarrow y[1]$, $x[2] \leftrightarrow y[2]$, ..., $x[n] \leftrightarrow y[n]$, be a finite sequence of pairs of nonempty strings over some alphabet Σ not containing the symbols "(" and ")". Post's correspondence problem asks if there is a finite sequence of integers $i_1 i_2 \cdots i_k$ such that $x[i_1]\, x[i_2] \cdots x[i_k] = y[i_1]\, y[i_2] \cdots y[i_k]$. It is one of the classic undecidable problems and remains undecidable even if the solution sequence is constrained to begin with the first pair, that is, $i_1 = 1$ (see [6] for instance). Consider the following variation of Post's problem.

1. Augment Σ by introducing two new symbols "(" and ")".
2. Replace each symbol $z \in \Sigma$ in the strings $x[i]$, $y[i]$ by the string (z).

It is obvious that any solution sequence for the modified strings is also a solution sequence for the original problem and vice versa. We call the class of problems obtained by this replacement as Post's problem with parenthesis, abbreviated pcp( ).

In this section we reduce pcp( ) to the safety problem for SPM systems. Each string in pcp( ) has at least three symbols. Moreover in a sequence of x strings or y strings no two consecutive symbols are identical. These properties are exploited in our construction and proof.

For a given instance of pcp( ) we construct the *corresponding SPM system* as follows. Let $x[i, l]$ signify the lth position in $x[i]$ for $l = 1 \cdots l_i$. Similarly let $y[i, m]$ signify the mth position in $y[i]$ for $m = 1 \cdots m_i$. We define a subject type for each position in each string as follows.

1. $TS = \{x[i, l], y[i, m] \mid i = 1 \cdots n,\ l = 1 \cdots l_i,\ m = 1 \cdots m_i\}$.

The symbols at these positions are denoted by $\text{sym}(x[i, l])$ and $\text{sym}(y[i, m])$.

We simulate a pair of strings $x[i] \leftrightarrow y[i]$ in the following way. Subjects of type $x[i, l]$ are authorized to create subjects of type $x[i, l+1]$, for $l = 1 \cdots l_i - 1$. Subjects of type $x[i, l_i]$, which simulate the end position of $x[i]$, are authorized to create subjects of type $y[i, m_i]$ to simulate the end position of $y[i]$. Preceding positions of $y[i]$ are simulated by creating them backwards. That is, subjects of type $y[i, m]$ are authorized to create subjects of type $y[i, m-1]$, for $m = m_i \cdots 2$. To simulate a paired sequence of x and y strings, subjects of type $x[i, l_i]$ are authorized to create subjects of type $x[j, 1]$. The latter in turn can create the pair of strings $x[j] \leftrightarrow y[j]$. This leads to the following definition of can-create, stated for convenience as a function.

2. For $i = 1 \cdots n$

$$\text{For } l = 1 \cdots l_i - 1,\ cc(x[i, l]) = \{x[i, l+1]\}$$

$$cc(x[i, l_i]) = \{y[i, m_i]\} \cup \{x[j, 1] \mid j = 1 \cdots n\}$$

$$\text{For } m = m_i \cdots 2,\ cc(y[i, m]) = \{y[i, m-1]\}$$

$$cc(y[i, 1]) = \phi.$$

This definition of cc is cyclic since subjects of type $x[i, l]$ can indirectly create subjects of type $x[j, k]$ and vice versa.

Points 1 and 2 above are critical to the construction, particularly the idea of simulating the $x[i]$ and $y[i]$ strings respectively by a forward and backward sequence of creates, and connecting these strings at their end points. We can visualize cc as shown in Figs. 1 and 2. Each circle represents the subject type indicated alongside. Each edge represents the direction in which creation is authorized. Figure 1 pictures cc vertically. It shows that a $x[i, 1]$ subject can create a $x[i, 2]$ subject and so on to form a sequence corresponding to the string $x[i]$. The $x[i, l_i]$ subject at the end of this sequence can create a $y[i, m_i]$ subject as well as $x[j, 1]$ subjects for $j = 1 \cdots n$. The $y[i, m_i]$ subject can create a $y[i, m_i - 1]$ subject and so on to form a sequence corresponding to the string $y[i]$. In the same way each $x[j, 1]$ subject can "grow" a pair of $x[j]$ and $y[j]$ strings. This pattern can be repeated indefinitely. If we follow a path such as the one straight below $x[i, 1]$ we can visualize a sequence of x strings paired with y strings as shown in Fig. 2. This gives us a horizontal view of cc ignoring the fan-out at the $x[i, l_i]$ subjects. This view is particularly useful for understanding the construction and proof.

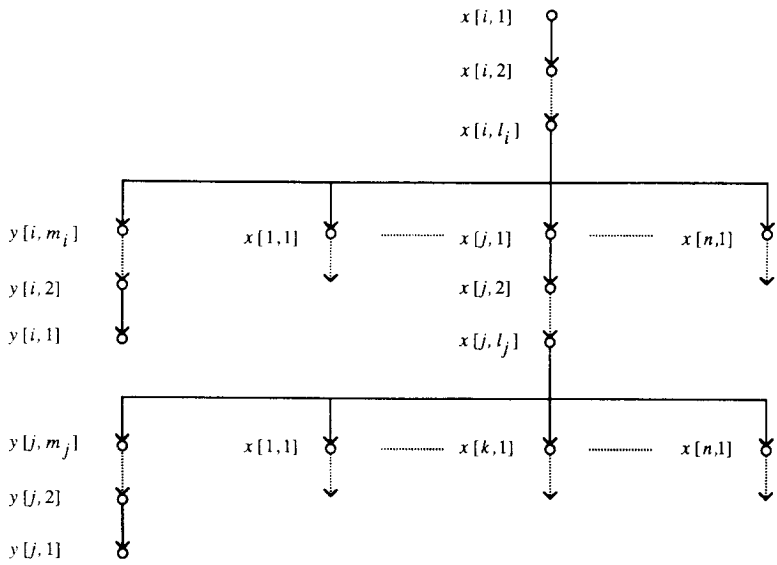

FIG. 1. Vertical view of can-create.

We can set up the create-rules to introduce appropriate links between adjacent subjects in Fig. 2. This is almost enough to simulate a sequence of x strings $x[i_1]\,x[i_2]\cdots x[i_k]$ paired with a corresponding sequence of y strings $y[i_1]\,y[i_2]\cdots y[i_k]$. There is, however, a missing connection between the end points of the y strings, for instance between the $y[i, m_i]$ and $y[j, 1]$ subjects. Establishing this missing connection properly is the main complication, particularly because we want to prevent a similar connection from occurring between the $y[i, m_i]$ and $y[k, 1]$ subjects. The key idea here is that the $y[i, m_i]$ subject is a direct child of the $x[i, l_i]$ subject, while the $y[j, 1]$ subject is a closer descendant of the $x[i, l_i]$ subject than the $y[k, 1]$ subject is. By carefully specifying the create-rules and filter functions we are able to establish the desired connection between the $y[i, m_i]$ and $y[j, 1]$ subjects by a sequence of copy operations while preventing a similar connection between the $y[i, m_i]$ and $y[k, 1]$ subjects. Once we are able to

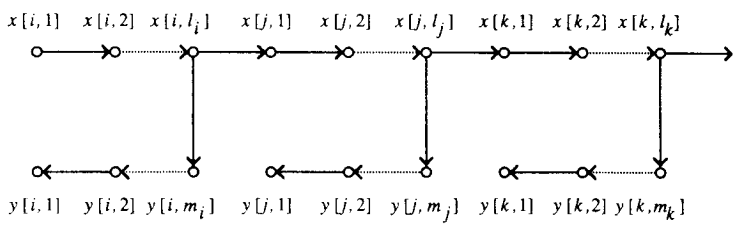

FIG. 2. Horizontal view of can-create.

ensure that the missing connections are properly established it remains to match the sequence of x strings and y strings subject by subject. This is done backwards by testing for a match at the end points of these two sequences. If a match exists it permits the subjects at the immediately preceding positions of the two sequences to be tested for a match and so on working our way back to the starting positions. This is the intuition behind the rest of the construction.

We define the following right symbols and link predicates with mnemonic significance as indicated.

3. $RC = \{a : c,\ e : c,\ m : c,\ p : c,\ r : c,\ t : c\}$, $RI = \{l : c\}$

Read as: a—adopt, e—end, l—leak, m—match, p—predecessor, r—refer, t—test.

4.

$link_s(U, V) \equiv U/p \in \mathrm{dom}(V)$	Successor link
$link_p(U, V) \equiv V/p \in \mathrm{dom}(U)$	Predecessor link
$link_e(U, V) \equiv U/e \in \mathrm{dom}(V) \vee V/e \in \mathrm{dom}(U)$	End link
$link_r(U, V) \equiv V/r \in \mathrm{dom}(U)$	Refer link
$link_a(U, V) \equiv U/a \in \mathrm{dom}(V)$	Adopt link
$link_t(U, V) \equiv U/t \in \mathrm{dom}(V) \wedge V/t \in \mathrm{dom}(U)$	Test link
$link_m(U, V) \equiv U/m \in \mathrm{dom}(V) \wedge V/m \in \mathrm{dom}(U)$	Match link
$link_l(U, V) \equiv U/m \in \mathrm{dom}(V) \wedge V/mr \in \mathrm{dom}(U)$	Leak link

Recall that links which do not exist in the initial state can be established by create-rules as a side effect of creation or by copy operations.

We now describe the role of the various links in our construction. Predecessor and successor links are established by the same right symbol p but in opposite directions. That is if $U/p \in \mathrm{dom}(V)$ we have $link_s(U, V)$ and $link_p(V, U)$. In this case $\tau(U)$ corresponds to the position that immediately precedes the position corresponding to $\tau(V)$ in a sequence of x or y strings. The $link_s$'s are established in a pattern corresponding to the left to right direction in Fig. 2 with $link_p$'s in the opposite direction. For the most part $link_p$ and $link_s$ are established by creation. The exceptions are $link_p$'s and $link_s$'s corresponding to the missing connections of Fig. 2, which are established by copy operations. The end, refer, and adopt links are used for this purpose. The end link connects an $x[j, l_j]$ subject to a $y[j, m_j]$ subject which it creates. The refer link has a non-empty filter function only from an $x[j, 1]$ subject to a $y[j, 1]$ subject. It is established by a sequence of copy operations over $link_s$'s from $y[j, m]$ subjects to $y[j, m+1]$ subjects, a copy operation over a $link_e$ from a $y[j, m_j]$ subject to a $x[j, l_j]$ subject, and finally a sequence of copy operations over $link_p$'s from $x[j, l]$ subjects to $x[j, l-1]$ subjects. The refer link is then used to set up an adopt link from a $y[i, m_i]$ subject to a $y[j, 1]$ subject, which in turn is used to establish the above-mentioned missing connections. The test and match links are used to match a paired sequence of x strings and y strings working backwards from the end points. These bidirectional links have non-empty filter functions only between $x[i, l]$ and $y[j, m]$ subjects. A test link between X and Y can be converted to a match link if and only if $\mathrm{sym}(\tau(X)) = \mathrm{sym}(\tau(Y))$. A match link in turn is used to establish a test link by copy operations between subjects to

which X and Y have $link_p$'s. To begin this process test links are introduced at the end points by the create-rules for $cc(x[i, l_i], y[i, m_i])$. Definition of the leak link amounts to requiring the match link and the refer link. The leak link can only be used to copy a $x[1, 1]/l$ ticket from a $x[1, 1]$ subject to a $y[1, 1]$ subject. It reduces the problem of finding a solution to a given instance of pcp( ) to that of establishing a leak link from a $x[1, 1]$ subject to a $y[1, 1]$ subject.

The above considerations motivate the following definitions for the filter functions. Values not explicitly defined are empty by default.

5. For $i = 1 \cdots n$,
   For $l = 1 \cdots l_i - 1$,
   $f_p(x[i, l+1], x[i, l])$
   $= \{y[k, q]/t \mid k = 1 \cdots n, q = 1 \cdots m_k\} \cup y[i, 1]/rc$
   For $m = 1 \cdots m_i - 1$,
   $f_p(y[i, m+1], y[i, m]) = \{x[k, q]/t \mid k = 1 \cdots n, q = 1 \cdots l_k\}$
   For $j = 1 \cdots n$,
   $f_p(x[i, 1], x[j, l_j]) = \{y[k, q]/t \mid k = 1 \cdots n, q = 1 \cdots m_k\}$
   $f_p(y[i, 1], y[j, m_j]) = \{x[k, q]/t \mid k = 1 \cdots n, q = 1 \cdots l_k\}$

   For $i = 1 \cdots n$,
   For $l = 1 \cdots l_i - 1$,
   $f_s(x[i, l], x[i, l+1]) = x[i, l]/tc$
   For $m = 1 \cdots m_i - 1$,
   $f_s(y[i, m], y[i, m+1]) = y[i, m]/tc \cup y[i, 1]/rc$
   For $j = 1 \cdots n$,
   $f_s(x[i, l_i], x[j, 1]) = x[i, l_i]/tc \cup y[i, m_i]/ac$
   $f_s(y[i, m_i], y[j, 1]) = y[i, m_i]/tc$

   For $i = 1 \cdots n$, $f_e(y[i, m_i], x[i, l_i]) = y[i, 1]/rc$

   For $i = 1 \cdots n$, $f_r(x[i, 1], y[i, 1]) = \{y[j, m_j]/a \mid j = 1 \cdots n\}$

   For $i = 1 \cdots n$, $j = 1 \cdots n$, $f_a(y[i, m_i], y[j, 1]) = y[i, m_i]/p$

   For $i = 1 \cdots n$, $j = 1 \cdots n$, $l = 1 \cdots l_i$, $m = 1 \cdots m_j$,

$$f_t(x[i, l], y[j, m]) = \begin{cases} x[i, l]/m & \text{if } sym(x[i, l]) = sym(y[j, m]) \\ \phi & \text{otherwise} \end{cases}$$

$$f_t(y[j, m], x[i, l]) = \begin{cases} y[j, m]/m & \text{if } sym(x[i, l]) = sym(y[j, m]) \\ \phi & \text{otherwise} \end{cases}$$

   For $i = 1 \cdots n$, $j = 1 \cdots n$, $l = 1 \cdots l_i$, $m = 1 \cdots m_j$,

$$f_m(x[i, l], y[j, m]) = \begin{cases} \{x[k, l_k]/tc \mid k = 1 \cdots n\} & \text{if } l = 1 \\ x[i, l-1]/tc & \text{otherwise} \end{cases}$$

$$f_m(y[j, m], x[i, l]) = \begin{cases} \{y[k, m_k]/tc \mid k = 1 \cdots n\} & \text{if } m = 1 \\ y[j, m-1]/tc & \text{otherwise} \end{cases}$$

   $f_l(x[1, 1], y[1, 1]) = x[1, 1]/l.$

The create-rules ensure that each subject has the mc and tc tickets for itself and

introduce test and end links at the end points of a pair of $x[i] \leftrightarrow y[i]$ strings. They also establish the predecessor and successor links. Finally they introduce copiable refer, adopt, and predecessor tickets required to establish the missing connections of Fig. 2. The formal statement of these rules is given below.

6. For $i = 1 \cdots n$, $l = 1 \cdots l_i - 1$,
$cr_p(x[i, l], x[i, l+1]) = \phi$
$cr_c(x[i, l], x[i, l+1]) = child/mtc \cup parent/p$
For $i = 1 \cdots n$, $m = m_i \cdots 2$,
$cr_p(y[i, m], y[i, m-1]) = child/p$
$$cr_c(y[i, m], y[i, m-1]) = \begin{cases} child/mrtc & \text{if } m = 2 \\ child/mtc & \text{otherwise} \end{cases}$$
For $i = 1 \cdots n$, $j = 1 \cdots n$,
$cr_p(x[i, l_i], x[j, 1]) = \phi$
$cr_c(x[i, l_i], x[j, 1]) = child/mtc \cup parent/p$
For $i = 1 \cdots n$,
$cr_p(x[i, l_i], y[i, m_i]) = child/act$
$cr_c(x[i, l_i], y[i, m_i]) = child/mptc \cup parent/et$

Points 1 through 6 enumerated above define the corresponding SPM scheme for a given instance of pcp( ). To complete the construction we need to specify the initial state, which we define as follows.

7. The initial state consists of a single subject $X_1[1, 1]$, whose type is $x[1, 1]$, with $\mathrm{dom}(X_1[1, 1]) = X_1[1, 1]/lmtc$.

The safety question is whether l can be leaked in the SPM system defined by points 1 through 7 above.

Naming conventions. It is useful to name subjects so their types are evident from their names to the extent possible. Subjects with names of the form $X[i, l]$ or $X_k[i, l]$ are understood to be of type $x[i, l]$. Similarly subjects with names of the form $Y[i, m]$ or $Y_k[i, m]$ are understood to be of type $y[i, m]$. When we use X, X', or X'' as the names of subjects we understand that these are of types $x[i, l]$ for unspecified i and l. Similarly subjects with names Y, Y', or Y'' are understood to be of types $y[j, m]$ for unspecified j and m.

The if part of the reduction is straightforward from the construction. The main steps are summarized below.

THEOREM 1. *If an instance of pcp( ) has a solution then l can be leaked in the corresponding SPM system.*

Proof. Let $1 i_2 \cdots i_k$ be a solution so $x[1]\, x[i_2] \cdots x[i_k] = y[1]\, y[i_2] \cdots y[i_k]$. Starting with $X_1[1, 1]$ construct a paired sequence of x and y strings, as indicated in Fig. 2, in the order $1 i_2 \cdots i_k$. Establish the missing $link_s$'s as outlined earlier. The create-rules introduce $link_t$'s between $X_{i_k}[i_k, l_{i_k}]$ and $Y_{i_k}[i_k, m_{i_k}]$ at the end points

of these sequences. Copy $X_{i_k}[i_k, l_{i_k}]/m$ from $X_{i_k}[i_k, l_{i_k}]$ to $Y_{i_k}[i_k, m_{i_k}]$ and vice versa via $link_t$'s. Use the resulting $link_m$'s in conjunction with $link_p$'s to establish $link_t$'s between the predecessors of $X_{i_k}[i_k, l_{i_k}]$ and $Y_{i_k}[i_k, m_{i_k}]$. Repeat this procedure to propagate $link_m$'s to the beginning of the paired sequence, i.e., between $X_1[1, 1]$ and $Y_1[1, 1]$. This is possible because $\mathrm{sym}(\tau(X)) = \mathrm{sym}(\tau(Y))$ all along. Finally copy $Y_1[1, 1]/rc$ from $Y_1[1, 1]$ to $X_1[1, 1]$, which, in conjunction with $link_m$, sets up a $link_l$ from $X_1[1, 1]$ to $Y_1[1, 1]$. So $X_1[1, 1]/l$ can be leaked from $X_1[1, 1]$ to $Y_1[1, 1]$. ∎

As is usual in reduction proofs the more difficult part is to show the converse property. The fact that no consecutive identical symbols occur in a sequence of x strings or y strings in pcp( ) is crucial to the proof.

THEOREM 2. *If l can be leaked in the corresponding SPM system the given instance of pcp( ) has a solution.*

Proof. The only lc ticket is $X_1[1, 1]/lc$ in the initial domain of $X_1[1, 1]$, so l can be leaked only by copying $X_1[1, 1]/l$ from $X_1[1, 1]$ to some $y[1, 1]$ subject, say $Y_1[1, 1]$, via a $link_l$. This amounts to requiring $link_m$ and $link_r$ from $X_1[1, 1]$ to $Y_1[1, 1]$. A $link_m$ between X and Y subjects can be established only by mutual exchange of each other's m tickets using $link_t$'s, which requires $\mathrm{sym}(\tau(X)) = \mathrm{sym}(\tau(Y))$. The $link_t$'s in turn are established by create-rules or copy operations. Creation establishes $link_t$'s only at the end points of x and y strings, in which case we also have $link_e(X, Y)$. (Because each string in pcp( ) has at least three symbols $X_1[1, 1]$ and $Y_1[1, 1]$ are not at the end points, so this case does not apply to them.) Otherwise we must somehow copy Y/t to X and vice versa. X can obtain Y/t in one of two ways shown in Figs. 3(a) and 3(b). Each edge depicts the link indicated by the label in the middle. In Fig. 3(a) Y/tc can be copied from Y to Y' to X', and then Y/t can be copied from X' to X. In Fig. 3(b) Y/tc can be copied from Y to Y' and from Y' to X. Y can similarly obtain X/t in one of two ways shown in Figs. 3(a) and 3(c). However, the situations of figures 3(b) and 3(c) cannot occur simultaneously. This is because f_s is non-empty only if $\tau(Y)$ and $\tau(Y')$ correspond to successive positions in a sequence of y strings and similarly for $\tau(X)$ and $\tau(X')$ with regard to x strings. Now for pcp( ) it is not possible to simultaneously have

$$\mathrm{sym}(\tau(X)) = \mathrm{sym}(\tau(Y')) \wedge \mathrm{sym}(\tau(Y)) = \mathrm{sym}(\tau(X')).$$

Therefore a $link_t$ can be established by copy operations only in the situation of Fig. 3(a) which requires a $link_m$ between X' and Y'. By applying this argument inductively it follows $link_m$ ($X_1[1, 1]$, $Y_1[1, 1]$) implies there exist X' and Y' with $link_e(X', Y')$ and a successor path (sequence of $link_s$'s) from $X_1[1, 1]$ to X' and from $Y_1[1, 1]$ to Y' such that:

1. Identical words are formed by the symbols corresponding to the subject types traversed by the successor path from $X_1[1, 1]$ to X' and from $Y_1[1, 1]$ to Y'.

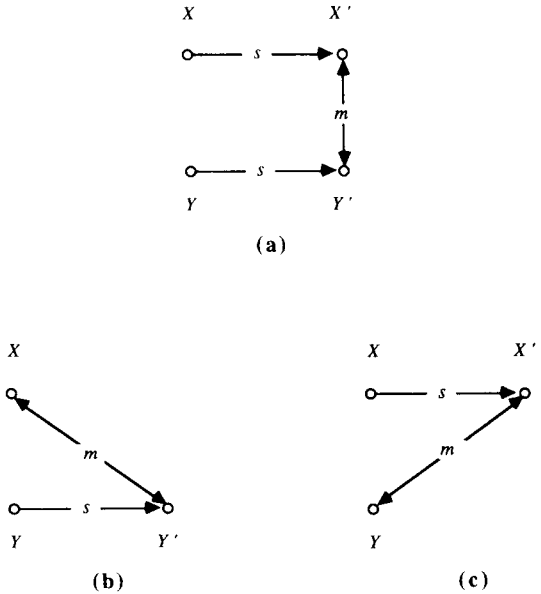

FIG. 3. Establishing a test link.

2. The types of subjects traversed by the successor path from $X_1[1, 1]$ to X' correspond to a sequence of x strings, while the types of subjects traversed by the successor path from $Y_1[1, 1]$ to Y' correspond to a sequence of y strings.

It remains to show that the sequences of x and y strings asserted above must be pairwise identical so they form a solution to pcp( ). Now all subjects are ultimately descendants of $X_1[1, 1]$. So for every subject there is a chain of its ancestors going back to $X_1[1, 1]$. Let $\omega(Y[i, 1])$ signify the closest $x[i, 1]$ ancestor of $Y[i, 1]$. It is evident that $link_r(X_1[1, 1], Y_1[1, 1])$ implies $X_1[1, 1] = \omega(Y_1[1, 1])$. This property ensures the desired result. Note that there is a unique successor path from $X_1[1, 1]$ to X' because $link_s$'s from $x[i, l]$ subjects are established only by creation. On the other hand there may be multiple successor paths from $Y_1[1, 1]$ to Y'. To see this consider that the $y[j, 2]$ subject in Fig. 2 may have several $y[j, 1]$ children, to all of whom the $y[i, m_i]$ subject can have $link_s$'s. However, this fan-out to multiple successor paths converges at the $y[j, m_j]$ subject. It follows that all successor paths from $Y_1[1, 1]$ to Y' traverse the same y strings. Moreover a $link_s$ from $Y[i, m_i]$ to $Y[j, 1]$ can be established by copy operations only if the creator $X[i, l_i]$ of $Y[i, m_i]$ is also the creator of $X[j, 1] = \omega(Y[j, 1])$. But then if we have X' and Y' with $link_e(X', Y')$ and successor paths from $X_1[1, 1]$ to X' and from $Y_1[1, 1]$ to Y', these paths must traverse pairwise identical sequences of x and y strings. So if the symbols in these sequences form the same word we have a solution to pcp( ). ∎

4. Attenuating Create-Rules

In this section we show that safety remains undecidable even if all create operations are required to be attenuating. The general idea behind the attenuating restriction is that tickets given to a child at the moment of creation should be somehow derived from tickets available to its parent. Then, at least with respect to tickets, the child will be "no more powerful" than its creator. Of course in SPM the type of the child may give it more power in the scheme than its parent has. For example consider the broadcast link $link_b(U, V)$ defined by $U/b \in \mathrm{dom}(U)$. That is, a subject with the *self*/*b* ticket has a broadcast link to everybody in the system. The motivation for the attenuating restriction is that the child should get the broadcast ticket only if the parent also has the broadcast ticket. However, the broadcast links from the child may be much more powerful than broadcast links from the parent as determined by f_b. The attenuating restriction was profitably used in our earlier work where it was shown that safety is decidable with loops (cycles of length one) in can-create provided the create-rules for loops are attenuating [14]. For loops the child and parent are of the same type so by attenuating the tickets we were really ensuring the child is "no more powerful" than its parent.

The safety algorithm of [14] is based on the observation that without creates safety is easily determined in polynomial time by simply executing copy operations until the state stabilizes. Cration is accommodated by breaking the analysis into two phases, as follows.

1. From the initial state construct an augmented state by create operations alone.
2. Compute the no-creates stable state from the augmented state of phase 1.

This strategy works provided we have a method for constructing a suitable augmented state. We have just shown in Section 3 that there is no such method in general. On the other hand for acyclic can-create there is a straightforward method for phase 1. Let each subject create one entity of every type it is authorized to create. Repeat this procedure for all children and so on. Clearly then phase 1 is guaranteed to terminate if and only if *cc* is acyclic. This method correctly analyzes safety because if a subject creates two entities of the same type there is no difference between them as far as the scheme is concerned. So from a worst-case viewpoint it suffices to create just one. In [14] this method was extended to handle attenuating loops by letting each subject create one child of its own type, if so authorized, and then simply ignoring the child for further creation in phase 1. This demonstration was important because attenuating loops were required to simulate models such as take-grant [7] in SPM.

With acyclic *cc* the tree of descendants created by a subject has finite depth, although its breadth is unbounded. As indicated above unbounded breadth is of no consequence for safety because multiple children of the same type add no power. The previous section shows that in general the tree cannot be truncated at some

computable depth. For attenuating loops however the analysis of [14] allows us to truncate the tree very easily. This naturally leads us to consider whether this idea can be generalized to cycles of length greater than one. In this section we show that this is not possible.

The first component of the attenuating restriction is that a newly created subject should not get more tickets than its creator. In our notation this is stated as follows.

I. $cr_c(u, v) \subseteq cr_p(u, v)$.

This requirement seems almost necessary for any meaningful notion of attenuating. However, it is not enough by itself. For instance $cr_c = cr_p = \{child/b\}$ satisfies this condition and introduces a broadcast ticket for the child in the child and parent domains. However, the latter is totally useless. We clearly need to relate *self*/*b* in the child domain to *self*/*b* in the parent domain. We can achieve this in a number of ways. Perhaps most straightforward is to interpret create-rules as upper bounds on the tickets which will actually be introduced, as follows.

II. Only those *parent*/$x : c$ and *child*/$x : c$ tickets, for which *self*/$x : c$ ticket is already present in the creator's domain prior to the create operation, will be actually introduced by the create-rule.

Note that in this case *parent*/$x : c$ in $cr_p(u, v)$ has no effect. This formulation is conservative in that tickets introduced by creation are truly derived from self tickets in the parent domain. We can even be more strict and require the strongly conservative formulation below.

II′. Only those *parent*/$x : c$ and *child*/$x : c$ tickets, for which *self*/xc ticket is already present in the creator's domain prior to the create operation, will be actually introduced by the create-rule.

In this case the parent is required to posses the xc ticket for itself before the create operation even if the create-rule only introduces x tickets. The actual formulation chosen in [14] was the following non-conservative one. It stipulates that if a ticket for the created subject is placed in the creator's domain the creator should also get the corresponding ticket for itself.

II″. If *child*/$x : c \in cr_p(u, v)$ then *parent*/$x : c \in cr_p(u, v)$.

With this formulation the parent may actually possess new tickets for itself as a result of creating a child. In a sense it makes creation attenuating after the fact rather then forcing it to be attenuating before the fact.

It turns out it really does not matter which of these forulations we chose. Safety is undecidable in general for all cases. We modify the create-rules for the scheme of Section 3 to be attenuating as follows.

6′. For $i = 1 \cdots n$, $l = 1 \cdots l_i - 1$
$cr_p(x[i, l], x[i, l+1]) = parent/aemprtc \cup child/aemprtc$
$cr_c(x[i, l], x[i, l+1]) = child/aemprtc \cup parent/p$
For $i = 1 \cdots n$, $m = m_i \cdots 2$
$cr_p(y[i, m], y[i, m-1]) = parent/aemprtc \cup child/aemprtc$
$cr_c(y[i, m], y[i, m-1]) = child/aemprtc$
For $i = 1 \cdots n$, $j = 1 \cdots n$
$cr_p(x[i, l_i], x[j, 1]) = parent/aemprtc \cup child/aemprtc$
$cr_c(x[i, l_i], x[j, 1]) = child/aemprtc \cup parent/p$
For $i = 1 \cdots n$
$cr_p(x[i, l_i], y[i, m_i]) = parent/aemprtc \cup child/aemprtc$
$cr_c(x[i, l_i], y[i, m_i]) = child/aemprtc \cup parent/et$

To accommodate the conservative variations of the attenuating restriction we modify the initial state of the construction of the previous section to be as follows.

7′. The initial state consists of a single subject $X_1[1, 1]$, whose type is $x[1, 1]$, with $\mathrm{dom}(X_1[1, 1]) = X_1[1, 1]/aelmprtc$.

With this initial state the three formulations of the attenuating restriction are all equivalent.

It is clear that with the create-rules of 6′ and the initial state of 7′ every subject will possess the *aemprtc* tickets for itself and for its children. We now show that this has no effect with respect to leaking *l*.

THEOREM 3. *l can be leaked in the original SPM system of Section* 3 *if and only if it can be leaked in the system obtained by modifying the create-rules to* 6′ *and the initial state to* 7′.

Proof. Since every ticket in the original system is also present in the modified system, if *l* can be leaked in the original system it can also be leaked in the modified system. For the converse we show the additional tickets in the modified system are of no consequence. In the modified system every subject possesses *self/aemprtc* for itself and *child/aemprtc* for every child. Consider the different rights in turn. In both systems only *child/ac* tickets for $y[i, m_i]$ subjects can be copied and the other *ac* tickets in the modified system do not establish significant links, i.e., links with non-empty filter functions. Similarly *ec* tickets cannot be copied in either system and introduce the same significant links between a $x[i, l_i]$ subject and its $y[i, m_i]$ child. The *self/mc* ticket exists for every subject in both systems. Since *child/m* : *c* tickets cannot be copied the only significant $link_m$'s that may result due to these are between some $X[i, l_i]$ and its $Y[i, m_i]$ child. However, we still need to copy $X[i, l_i]/m$ from $X[i, l_i]$ to $Y[i, m_i]$ in which case we can also copy $Y[i, m_i]/m$ from $Y[i, m_i]$ to $X[i, l_i]$ in the original system. Similar arguments can be made for the *prtc* rights. ∎

5. Conclusion

We have shown that safety for SPM systems with cyclic can-create relations is in general undecidable. This complements our earlier result that with acyclic can-create safety is decidable [14]. In [14] we also showed that safety remains decidable with loops (cycles of length one) in can-create provided the create-rules for loops are attenuating. We believe that attenuating loops cover all practical systems that might be considered, and have been unable to formulate a realistic SPM system for which this restriction cannot be met. In Section 4 we have shown that extending the attenuating restriction to all create-rules still leaves the safety problem undecidable. An interesting open question is whether or not SPM schemes with non-attenuating loops have a decidable safety problem.

The SPM framework has a rich structure and numerous theoretical questions can be posed. The ones of greatest of greatest interest at the moment pertain to the modeling power of SPM. In particular how does SPM compare in power with the monotonic access matrix of Harrison and Ruzzo [3]? It is quite straightforward to express an SPM system in the latter formulation. Whether the converse is true is a crucial open question. We would be pleased if it turns out to be the case. Then SPM could be viewed as an alternate formulation of the monotonic access matrix but with richer structure and a natural demarcation between its decidable and undecidable cases.

Acknowledgments

I am grateful to my colleague Timothy Long of Ohio State University for comments on a draft of this paper. I am also indebted to the referees. Their comments have greatly improved the paper.

References

1. G. S. Graham and P. J. Denning, Protection—Principles and practice, *in* "Proceedings of the AFIPS Spring Joint Computer Conference," Vol. 40, pp. 417–429, 1972.
2. M. H. Harrison, W. L. Ruzzo, and J. D. Ullman, Protection in operating systems, *Comm. ACM* **19**, No. 8 (1976), 461–471.
3. M. H. Harrison and W. L. Ruzzo, Monotonic protection systems, *in* "Foundations of Secure Computations" (R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, Eds.), Academic Press, New York, 1978.
4. A. K. Jones, R. J. Lipton, and L. Snyder, A linear time algorithm for deciding security, *in* "17th IEEE Symposium on the Foundations of Computer Science," pp. 337–366, 1976.
5. B. W. Lampson, Protection, *in* "5th Princeton Symposium on Information Science and Systems," pp. 437–443, 1971; reprinted in "ACM Operating Systems Review," Vol. 8, No. 1, pp. 18–24, 1974.
6. H. R. Lewis and C. H. Papadimitriou, "Elements of the Theory on Computation," Prentice-Hall, Englewood Cliffs, NJ, 1981.
7. R. J. Lipton and L. Snyder, A linear time algorithm for deciding subject security, *J. ACM* **24**, No. 1 (1977), 455–464.
8. A. Lockman and N. Minsky, Unidirectional transport of rights and take-grant control, *IEEE Trans. Software Eng.* **SE-8**, No. 6 (1982), 597–604.

9. N. MINSKY, Selective and locally controlled transport of privileges, *ACM Transactions on Programming Languages and Systems* **6**, No. 4 (1984), 573–602.
10. J. H. SALTZER AND M. D. SCHROEDER, The protection of information in computer systems, *Proc. IEEE* **63**, No. 9 (1975), 1278–1308.
11. R. S. SANDHU, "Design and Analysis of Protection Schemes Based on the Send-Receive Transport Mechanism," Ph.D thesis, Department of Computer Science, Rutgers University, 1983.
12. R. S. SANDHU, The SSR model for specification of authorization policies: A case study in project control, *in* "8th IEEE International Computer Software and Applications Conference," pp. 482–491, 1984.
13. R. S. SANDHU AND M. E. SHARE, Some owner based schemes with dynamic groups in the schematic protection model, *in* "IEEE Symposium on Security and Privacy," pp. 61–70, 1986.
14. R. S. SANDHU, The schematic protection model: Its definition and analysis for acyclic attenuating schemes, *J. ACM* **35**, No. 2 (1988), 404–432.
15. R. S. SANDHU, The demand operation in the schematic protection model, *Inform. Process. Lett.* **32**, No. 4 (1989), 213–219.
16. L. SNYDER, Formal models of capability-based protection systems, *IEEE Trans. Comput.* **C-30**, No. 3 (1981), 172–181.

Printed by Catherine Press, Ltd., Tempelhof 41, B-8000 Brugge, Belgium