DOCUMENT RESUME

ED 090 983 IR 000 594

AUTHOR	Hsiao, D. K.; And Others
TITLE	A Model for Data Secure Systems (Part 1).
INSTITUTION	Ohio State Univ., Columbus. Computer and Information Science Research Center.
SPONS AGENCY	Office of Naval Research, Washington, D.C.
REPORT NO	OSU-CISRC-TR-73-8
PUB DATE	Feb 74
NOTE	45p.
EDRS PRICE	MF-$0.75 HC-$1.85 PLUS POSTAGE
DESCRIPTORS	Computers; *Conceptual Schemes; Confidentiality; *Data Bases; *Information Retrieval; Information Storage; Models; Program Descriptions; *Security
IDENTIFIERS	*Access; Extended Logical Data Base; Privacy; Protection

ABSTRACT

A description is provided of a conceptual model for a data secure system. The discussion first offers a formal working vocabulary and next, using the intuitive idea of a dichotomy between permissible and impermissible accesses, formalizes the idea with an Extended Logical Data Base and with protection specifications and patterns. These specifications and patterns encompass the traditional identification, authentication, and verification procedures by recognizing that these procedures can be compared solely on the basis of their answers to specific access attempts. A limited calculus of protection patterns is offered, suggesting both comparative and generative operators, and a variety of protection specifications is defined and demonstrated. Finally, a demonstration is made of the fact that it is possible to protect any accessible data in a data base with a proposed protection specification which is independent of the structure and implementation of the data base. (Author)

TECHNICAL REPORT SERIES

COMPUTER & INFORMATION SCIENCE RESEARCH CENTER

THE OHIO STATE UNIVERSITY COLUMBUS, OHIO

OSU-CISRC-TR-73-8

A MODEL FOR DATA SECURE SYSTEMS (Part I)

by

D. K. Hsiao, D. S. Kerr
and E. J. McCauley III

Work performed under Contract N00014-67-A-0232-0022,

Office of Naval Research

The Computer and Information Science Research Center
The Ohio State University
Columbus, Ohio 43210

February 1974

Preface

The project on Data Security and Data Secure Systems was formed in the fall of 1972 and funded on March 1, 1973. The principal investigator of the project is Dr. David K. Hsiao, Associate Professor of Computer and Information Science. There are five Graduate Associates and Assistants on the project, R.I. Baum, N. Kaffen, E.J. McCauley, C.J. Nee, and S. Peden. Presently, Dr. Douglas S. Kerr, Associate Professor of Computer and Information Science, is serving as an investigator on the project.

Five major research and experimentation efforts are underway. We intend to issue a series of technical reports at the milestones of these efforts. Although early reports may be preliminary, we believe that they can serve as position papers for the research being pursued. These five research and experimentation efforts are listed as follows:

(1) A data secure system based on the theory of security deadlock.

(2) Theoretical foundations for context protection and consistent control in data secure systems.

(3) A data secure computer (hardware) architecture.

(4) Design and certification of data secure system kernels.

(5) A system for experimenting with access control mechanisms.

The Office of Naval Research (contract N00014-67-A-0232-0022,), and the Office of Science Information Service, National Science Foundation (Grant No. GN534.1) are acknowledged for their support in the preparation of this study.

The report is published by the Computer and Information Science Research Center of The Ohio State University which consists of the staff, graduate students, and faculty of many University departments and laboratories. The research was administered and monitored by The Ohio State University Research Foundation.

Abstract

A multi-level model for data secure systems is proposed. In this model the relevant issues in data security, such as integrity, privacy protection and controlled information sharing, can be studied, on the one hand; and the conventional procedures such as identification, authentication, authorization, and compartmentalization can be characterized, on the other hand. Furthermore, the model allows different problems in data security to be considered at a level of abstraction appropriate to the specific issue and procedure under study. The highest level is conceptual. In it, "patterns of protection" (intuitively, the ways the users may access the data) can be defined in formal and unambiguous ways. The intermediate level of the model is structural. Here, the primitives to be utilized in the realization of the patterns of protection defined in the higher level will be specified. The most important feature of this level is that the critical functions of an access control mechanism are no longer carried out by complex, and thus potentially unreliable programs, but are inherent in the basic structure of the system by the utilization of deadlocks. When a user attempts an unpermitted access, he deadlocks with a "pseudo-user" and cannot proceed. Thus, the demonstration of system correctness involves the certification of a limited number of small, single-purpose modules and the verification of the correctness of the user/pseudo-user interaction. On the lowest level, a system to illustrate the utility and practicality of the model will be created. Overall, the research should suggest a modelling and design technique for a demonstrably complete and correct system for providing logical access control in a shared data base system.

Our present plan for this research consists of three studies. The first is to complete our abstract model of data secure systems and develop a general theory of data security as proposed. In particular, we emphasize the structural level of the model. It is hoped that this level of modelling can reveal the inner working of the access control mechanism based on the theory of deadlock. With a good understanding of its inner working, the access control mechanism can then be properly designed and implemented. The application of the theory of deadlock to access control is new. We believe that this is the first application. Traditionally, system designers attempt to avoid and circumvent the system deadlocks which tie up system

resources and utilities. However, in our case, we deliberately tie up resources and utilities as a means to deadlock penetrators of the system. Obviously, these resources and utilities are logical resources such as files, records and fields and functional utilities such as data access and manipulations. Such deadlock is called security deadlock. One of the basic requirements is that no authorized use of and access to the data base will cause a security deadlock and any unauthorized use or access will cause an immediate security deadlock. This requirement will be met.

The Part I of this report deals only with the conceptual model.

Table of Contents

I. Introduction

> We now realize that perhaps the most important and powerful computer applications will involve the use of a computer as an _extension_ of the human intellect rather than as a replacement for it.
>
> [Zschau]

Computer applications are moving towards this goal with increasing speed. One of the ways in which computer systems can extend man's intellect is through their capability to store and retrieve data. As such data base becomes larger, more complete and more common, the importance of protection of the data grows. Without adequate protection, no user will be willing to entrust his sensitive data to the system. Further, no user can be sure that the data base is an accurate reflection of reality unless we provide sufficient protection of the data base. Running counter to the security requirements for protection is a need to provide for sharing of data so that users can "build on the work of others." Without sharing of data, there is little intellectual value in the accumulation of data. Thus, one of the most difficult and perplexing problems is how to provide adequate protection and still allow flexible sharing.

A data base is a collection of data structured in a way to facilitate some aspect of its use, such as query answering, update, report generation, etc. The exact structure chosen depends on precisely what the intended use of the data base is. Increasing amounts of work have been done in support of a separation of the physical and logical structure of the data base. Such a separation allows users to be concerned with the logical content of the data base rather than its physical representation. Data base computer systems must have some protection from unauthorized use and destruction of the data base. Conventionally, this is done by means of procedures which check the legitimacy of certain user actions, monitor the interaction between the user and the system and enforce the control of the user's access to the data.

In order to build a model which will allow us to study data security in comparative isolation at a level of abstraction appropriate to each specific study, we will need a multi-level model.

The highest and most general level, the conceptual model, lets us focus on the specification and description of _protection patterns_ which constitute the fundamental concepts for data security and integrity requirements in data secure systems. Conventional procedures such as identification, verification, authentication, authorization and compartmentalization should be easily characterized in terms of protection patterns. Furthermore, the roles of these procedures in data secure systems should be clearly delineated. With the conceptual model, we can then study the relationships among these procedures and their effects on data secure systems. New procedures and consistant variations may be formed when new security and integrity problems arise.

To provide a means of rigorously characterizing the various patterns of protection, the conceptual model will be a formal model. We define a protection relation:

$$D \times U \times A \rightarrow \{\text{permit, deny}\}$$

where D is the set of logical data in the data base
U is the set of users
A is the set of possible access types

The method of specification of which members of D, the set of logical data, are involved is potentially the most difficult part for us to do. Nevertheless, there are solutions which we shall present in later sections.

A protection pattern will be a prescribed collection of protection relations. For consistency, the protection patterns themselves are part of the logical data base. They may be manipulated by suitably authorized users in much the same way as the rest of the data base. The conceptual model is, therefore, also concerned with the establishment of the rules by which the patterns of protection can be maintained in a consistent and complete way. For example, a given user should not be able to modify the patterns which affect him, for this would be tantemount to no security at all. The major thrust of the conceptual model is aimed to provide a well defined, easily understood model for the characterization of logical data base protection.

The second level, the structural model, lets us examine the problems associated with use of the data base and computer system under the patterns of protection defined in the conceptual model. It is in the structural model that the protection mechanisms of the data secure system are characterized. It is also planned that the characterizations will show clearly the working of the mechanisms.

The structural model is mainly concerned with the logical implementation of the conceptual model in terms of a set of primitives by which any protection pattern defined in the conceptual model may be carried out. Mechanisms needed for regulating the implementation of the protection patterns are called protection mechanisms. The structural approach of this model will enable the reader to gain enough insight into the design of the mechanisms and to understand the working of the primitives in relation to the mechanisms.

A crucially important fact of the structural model is how the patterns of protection in the conceptual model are to be presented in the structural model as a subset of primitives. The most promising mechanisms under consideration for regulating the patterns are deadlock-based. The deadlock-based mechanisms create for each real (user) process* a pseudo-process which does not exert any effect on the system, other than deadlocking with its process if the process attempts an illegal access. Deadlock-based protection mechanisms have the advantage that checking of whether or not the access attempt is legal is an implicit and intrinsic feature of the structure of the process/pseudo-process relationship. This feature is in contrast to other schemes which require an explicit checking for each request. Implicit checking requires almost no overhead. Furthermore, it is possible to verify the correctness of the working mechanisms by verifying their intrinsic structural features and by applying the theory of deadlock.

The structural model is therefore process-oriented with each active user corresponding to a process and the related pseudo-process. The data base activities of the user can thus be viewed as requests of his process and pseudo-process for data base resources. If the request is illegal, the process and the pseudo-process will deadlock each other resulting in no activity for the user.

In deadlock-based protection mechanisms, many of the conventional procedures, like verification and authentication become more a feature of the basic structure of the system, than of explicit programs. There is considerable advantage over more traditional approaches because we can prove the correct interactions of a limited number of processes more easily than we can prove the correctness of a set of potentially complex programs.

* The process referred to here has the same meaning as a process in the MIT-Multics or a task in the IBM 370 VS2/Release 2.

To convincingly demonstrate that the process and pseudo-process interlocking on data base items will create a deadlock situation if and when the process makes an illegal request, we have the well-developed theory of system deadlock to rely on. Thus, sufficient and necessary conditions in which a system deadlock will occur are known. We only need to apply the system deadlock theory to security deadlock situation.

At the third level, the implementation model is concerned with practical aspects of actually implementing the model described by the other levels. It will be an experimental data secure system for the demonstration of well conceived concepts in the conceptual model and carefully developed mechanisms in the structural model.

II. The Conceptual Model

One of the problems that plague us in attempts to study protection is the lack of a suitable theoretical framework to allow for the comparison and analysis of the many different ideas for providing protection for data base systems. The goal in building the conceptual model is to provide such a framework and to demonstrate its utility.

We shall try to use a more formal approach with the hope of arriving at more precise definitions of the model. Nevertheless, there are definitions which must remain intuitive and somewhat 'undefined.'

Defn: The Physical Data Base is the underlying physical reality of a data base, e.g., a reel of magnetic tape, a deck of cards, a collection of disc tracks, etc.

Defn: The Logical Data Base is the set of all elements of information contained in the PDB. Furthermore, the elements of information in LDB are referred to as logical data.

Defn: A Data Base System(s) is (are) the collection of computer programs, procedures, and components etc. for the creation, use and maintenance of the Logical Data Base on the Physical Data Base. Examples of Data Base Systems are many, such as IMS, CICS, TDMS, etc.

Informally, the Logical Data Base is the collection of all possible "answers" obtainable or extractable by the Data Base System from the Physical Data Base in response to "questions" by users of the Data Base System.

Assumption: It is possible to enumerate completely every piece of logical data contained in the Physical Data Base.

This assumption often goes unstated since it appears so obvious. If we accept the assumption, however, it is not clear for many data base systems how to form the "answers" since the enumeration is hardly an efficient way to extract them. Nevertheless, the basis for this assumption is that if an element of data is capable of being "found" in the Physical Data Base by the Data Base System, then it can be enumerated. If it can be enumerated, then it can be used for the formulation of an "answer' in Logical Data Base.

The Logical Data Base will occupy the central position in any discussion of protection. A major lack in most Data Base System protection capabilities is the ability to protect only the Physical Data Base, rather than the Logical Data Base. This lack of more subtle protection makes many existing protection capabilities inadequate for modern, multi-user, integrated Data Base Systems. The inadequacy will be greatly multiplied since as the current research in Data Base Systems suggests, there will be in the future even greater difference and separation between the Physical Data Base and the user's view of it, the Logical Data Base.

Now we define more formally some of the terms which have been frequently used in data base system technology.

Defn: A User will be the generic term for any agent which attempts to use the Logical Data Base in some way.

Defn: Access is any activity by a user which requires logical data from the Logical Data Base and which demands a completion.

Defn: An access is denied if it is indefinitely delayed (never completed); otherwise, it is permitted.

Access may be further subdivided into access types, such as read, write, execute, search, retrieve, etc. We shall leave the exact connotations of any particular access type unspecified at this point.

Let us now define some of the terminology more leisurely. The interaction of a user with the Data Base System can be viewed as a series of requests and replies, with the system making some requests and the user making others. The first dialog is the identification procedure. The identify and access types of the user are established and retained by the system for future use. Once the identity of the user is known, the purpose of a data base system is to use the information in the Logical Data Base in the reply to the user's requests. For every such request the system performs an authentication procedure to determine whether the request should be permitted or denied, utilizing the information established and retained in the identification procedure. The system then carries out authenticated requests. As more sophisticated data base systems are developed where the separation of Logical Data Base and Physical Data Base is evident, more of the burden of searching for the information in the Logical Data Base which satisfies the user's request is placed on the system.

To allow users to access only certain parts of a data base, the division of the data base into logical regions may be requested. Furthermore, the logical data base may take on different apparent regions for different users. The study of logical divisions, their access requirements, and their physical organizations is termed compartmentalization, in which parts of the data base are separated logically and/or physically. For particularly sensitive parts of the data base, it may be necessary to further check on the legitimacy of attempted accesses. These verification procedures may be as simple as asking the user "ARE YOU SURE?", or may be much more elaborate. It is well to compartmentalize even if especially sensitive data is not involved. With these "firewalls" the entire data base is less likely to be affected if some untoward event (such as physical damage, accidental destruction, or illegal access) occurs.

It is also reasonable to consider the information used in the identification, authentication, compartmentalization and verification procedures as part of the data base which may be manipulated in the same manner as the rest of the data base. An authorization procedure is the only means by which this information is created and maintained. In particular, certain users (say, the creator) of logical data can authorize other users of the Data Base System to access the data by exercising the authorization procedure.

With this discussion, we have the following important notions.

Defn: Security is the prevention of unauthorized use of the data.

Defn: Integrity is the prevention of unauthorized or accidental destruction or modification of the data.

Defn: Access Control is the process of determining the authorized users of the Data Base System (and thus the Logical Data Base), and of determining which accesses may be permitted and which should be denied.

Defn: Contamination is a breach in integrity; Penetration is a violation of security. Both of these are caused by the user of the Data Base System and known collectively as interference.

Defn: A Protection Mechanism (or simply, protection) is an attempt to provide security and integrity by means of access control and interference prevention.

What has been characterized is a general data base system with access control and interference prevention. Practical systems embody the procedures discussed above in various forms reflecting the needs and purposes of the system. Any data base system must have some security and control if the user is to place any confidence in the data stored and retrieved. What distinguishes a data secure system is that the security and integrity are integrated into the system, not hung on as an extra module or two. The protection mechanism of the data secure system is logically complete and can be convincingly demonstrated to operate correctly and effectively even when under strong attack by well equipped (skillful and knowledgeable) penetrators.

The preceding discussion has laid the formal basis for the conceptual model and has also outlined in a somewhat less less formal way the concepts of most contemporary Data Base System protection mechanisms. Let us now first go strongly into the theoretical aspects of the conceptual model, and then show how such a theory may be applied by demonstrating how some fairly diverse protection mechanisms may be described by the model.

Defn: The Extended Logical Data Base (ELDB) is a set of triples

$$(u,a,d)$$

where

u is a user identifier,

a is an access-type identifier,

d is the identifier of an element of the Logical Data Base.

The Extended Logical Data Base is formed by making a triple for every possible access to the Logical Data Base by every possible user. Thus, the Extended Logical Data Base is a complete characterization of all possible accesses to the data base. Clearly, the Extended Logical Data Base has an immense number of elements for even the most trivial cases. Our goal is to suggest methods which deal with aggregates of Extended Logical Data Base elements. The Extended Logical Data Base forms the foundation for any discussion of interference prevention and access control.

Let us consider the components of these triples in some detail. Not too much need be said about the user identifier, since its meaning and importance are obvious. However, our definition of access was quite broad so as to include all actions in which information from the Logical Data Base is used. Thus, we must consider the connotations of particular access types. A servicable definition is, "Each access type is a program which effects a particular variety of access..." [PopeG73]. The access type identifiers in the triples are, thus, program identifiers. These access programs range from basic hardware operations through supervisor services to more elaborate user created programs. Finally, we have made the assumption that every element of the Logical Data Base can be identified and assigned a unique name. These names are the data identifiers of the Extended Logical Data Base.

<u>Defn</u>: A protection specification is a relation
$p: S \to \{\text{permit}, \text{deny}\}$, where $S \subseteq \text{ELDB}$.

Given $x \in S$, $p(x) = $ deny will indicate that under this protection specification, the access is denied and $p(x) = $ permit that the access is permitted.

Intuitively, a protection specification is an assertion about the protection of the data base system. We do not require that a user who is creating protection specifications have global knowledge of the system. He can make the specification cover only that Extended Logical Data Base subset about which he is cognizant.

Defn: A protection pattern is a set, P, of protection specifications such that every triple of the Extended Logical Data Base is in the domain of at least one specification.

Defn: A protection pattern, P, is consistant if every triple of the Extended Logical Data Base which lies in the domain of more than one protection specifications of P is mapped onto the same value by each of the protection specification in whose domains the subset is contained. A protection pattern which is not consistant will be said to be inconsistant.

We shall assume that protection patterns are consistant unless specifically indicated otherwise. (Later, we shall suggest methods to resolve inconsistant specifications). In a consistant protection pattern it does not matter which protection specification is applied to an element for which several are applicable because by definition if access to the element is, for example, denied as a result of the application of one of the protection specifications (i.e., $p_i(x)$ = deny), the access to the same elements will still be denied no matter which other protection specifications are applied. We may simply refer to $P(x)$, rather than some particular $p_i(x)$.

Thm: A consistent protection pattern partitions the Extended Logical Data Base into those accesses which are permitted (may be completed) and those which are denied (indefinitely delayed).

This theorem is the heart of the conceptual model. It formalizes the notion that of all the "things" that the users might try to do, the protection mechanism, at any instant, partitions the "things" into a set which are allowed, and a set which are not allowed. This formalism lets us consider all deterministic protection mechanisms in the same framework because the result of any protection mechanism is to return a binary (permit or deny) result to a user access attempt. Let us demonstrate how some of the existing protection mechanisms may be characterized by the model.

III. Examples - Modeling Existing Protection Mechanisms

Let us demonstrate how some of the existing protection mechanisms may be characterized by the model.

Example 1: Modeling Protection Mechanisms Based on an Access Matrix [GrahG71, GrahG72].

The access matrix used by the protection mechanisms may be conceptualized (as depicted in [GrahG72]) in the following figure:

		Subjects			Objects			
					Files		Devices	
		S_1	S_2	S_3	F_1	F_2	D_1	D_2
Subjects	S_1		block wakeup		read write		seek	seek
	S_2			stop		update		
	S_3				delete			

Fig. 1 - Portion of an access matrix

The access attributes (permitted access types, in our terminology) of subject (user, in our terminology) S_i towards object O_j are contained in the (i,j) entry of the access matrix. For ease of discussion, let us consider a system environment in which we are only concerned with accesses towards files (e.g., F_1 and F_2 in Fig. 1). Further, let there by only four different access types: read, write, update, and delete as shown in Fig. 1.

Each entry of the table in Fig. 2 is a protection specification. Collectively, the protection specifications form a protection pattern which partitions the data base into accessible and inaccessible files.
In other words, the table depicted in Fig. 2 is the protection pattern. This example shows many things. First, the conceptual model can describe the access matrix in a natural, even trivial way. Second, the Extended Logical Data Base has an impossibly large number of elements. Finally, it

(u, a, d) $\rightarrow$	$\{permit, deny\}$
(S_1, R, F_1)	permit
(S_1, W, F_1)	permit
(S_1, U, F_1)	deny
(S_1, D, F_1)	deny
(S_2, R, F_1)	deny
(S_2, W, F_1)	deny
(S_2, U, F_1)	deny
(S_2, D, F_1)	deny
(S_3, R, F_1)	deny
(S_3, W, F_1)	deny
(S_3, U, F_1)	deny
(S_3, D, F_1)	permit
$(S_2, U. F_2)$	permit
$(S_2, D. F_2)$	deny
(S_3, R, F_2)	deny
(S_3, W, F_2)	deny
(S_3, U, F_2)	deny
(S_3, D, F_2)	deny
(S_1, R, F_2)	deny
(S_2, W, F_2)	deny
(S_1, U, F_2)	deny
(S_1, D, F_2)	deny
(S_2, R, F_2)	deny
(S_2, W, F_2)	deny

Fig. 2 - The Portion of an Access Matrix as Characterized in the Protection Pattern

should be noted that access matrix based protection mechanisms as discussed in [GrahG71] were mainly concerned with the protection in operating systems where the file-like objects were comparatively few in number and were the same for every user (i.e., protection was limited to the Physical Data Base). In general neither of these two assumptions is true for Data Base Systems, which deal with the Logical Data Base.

Example 2: Modeling Protection Mechanisms Based on Capability Lists.

In essence, a capability list (c.list) is one row from the access matrix. Many different definitions for c.lists exist. We, rather arbitrarily, will use the one found in [DennJ66]:

> Each capability in a c.list locates by means of a pointer some computing object and indicates the actions that the computation may perform with regards to that object. Among these capabilities there are really several [memory] segment capabilities, which designate segments that may be referenced by the computation and that give by means of access indicators an indication of the kind of reference permitted. . .

Although the definition of c.list confines the protection to memory protection, we will allow more liberal interpretation of the term 'segment.' Let us consider these segments as files. The characterization of a c.list based protection mechanism by the conceptual model is again relatively trivial. Considering only the files, let us first characterize access matrix in Fig. 1 as a c.list.

C.list for S1

R,W; Pointer to F_1
null

C.list for S2

null
U; Pointer to F2

C.list for S3

D; Pointer to F1
null

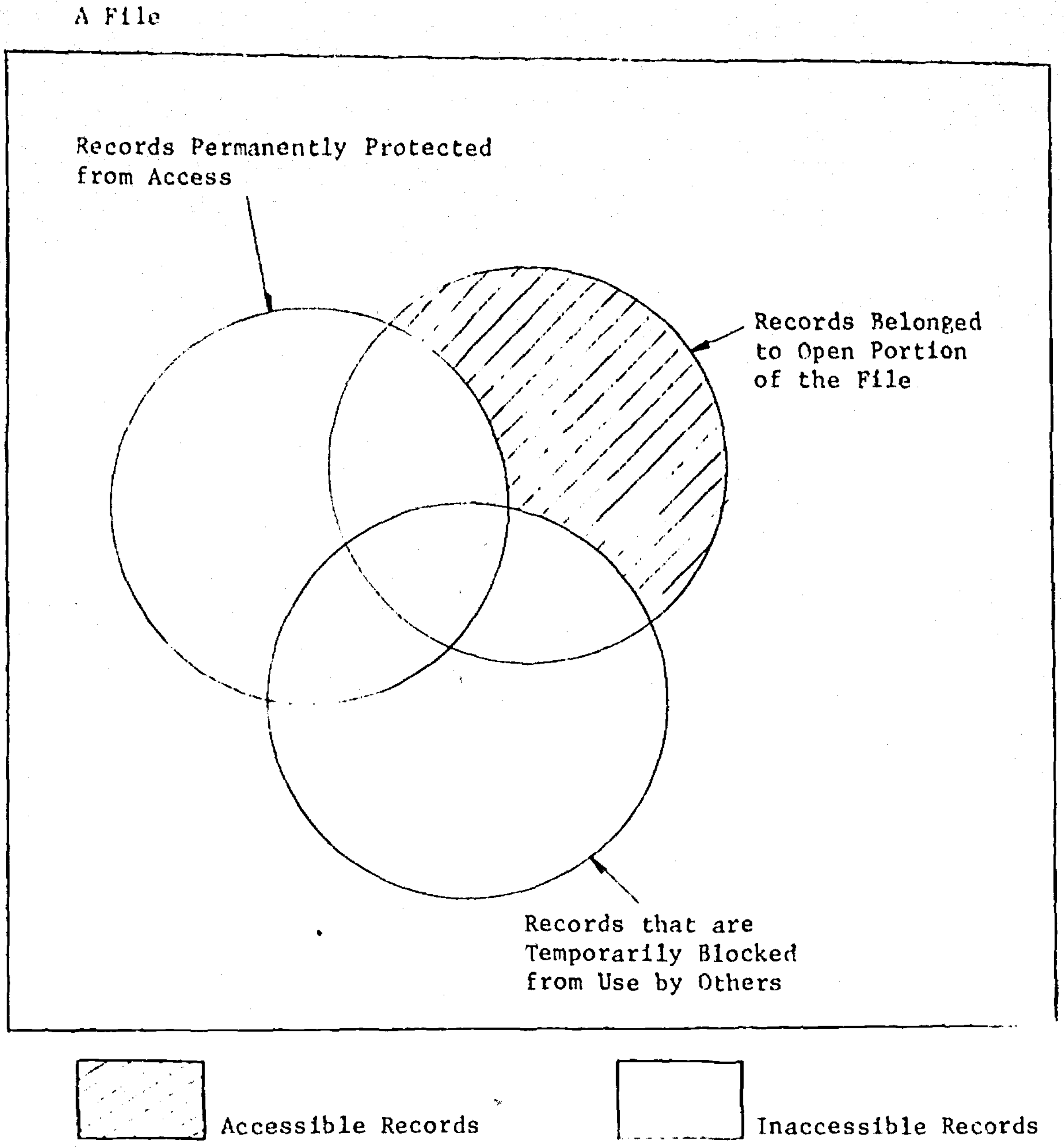
Fig. 3 - The Rows of an Access Matrix as Characterized by Capability Lists (c.list)

The c.list system may be modified to compress out the null entries. However, this has no effect on the concept of either the c.list or the conceptual model. To map c.lists to the protection pattern of the conceptual model, we follow the following procedure. Everywhere that the c.list gives the user a capability to reference some segment, locate the Extended Logical Data Base element (consisting of identifiers of the user, access type and segment) corresponding to that segment. There will be such an element because we have explicitly defined the Extended Logical Data Base to consist of all possible accesses. Next assign the value 'permit' to the element (i.e., P(x) = permit). In other words, we have just formed a protection specification of the element to which access is permitted. When all the c.lists have been exhausted, we have obtained all the possible elements to which accesses are allowed. Finally, assign all the other non-referenced elements with the value deny (i.e., P(x) = deny). Thus, the non-referenced segments are not accessible. We have now a protection pattern in the conceptual model corresponding to the c.list.

Example 3: Modeling Protection Mechanisms Based on Authority Items.

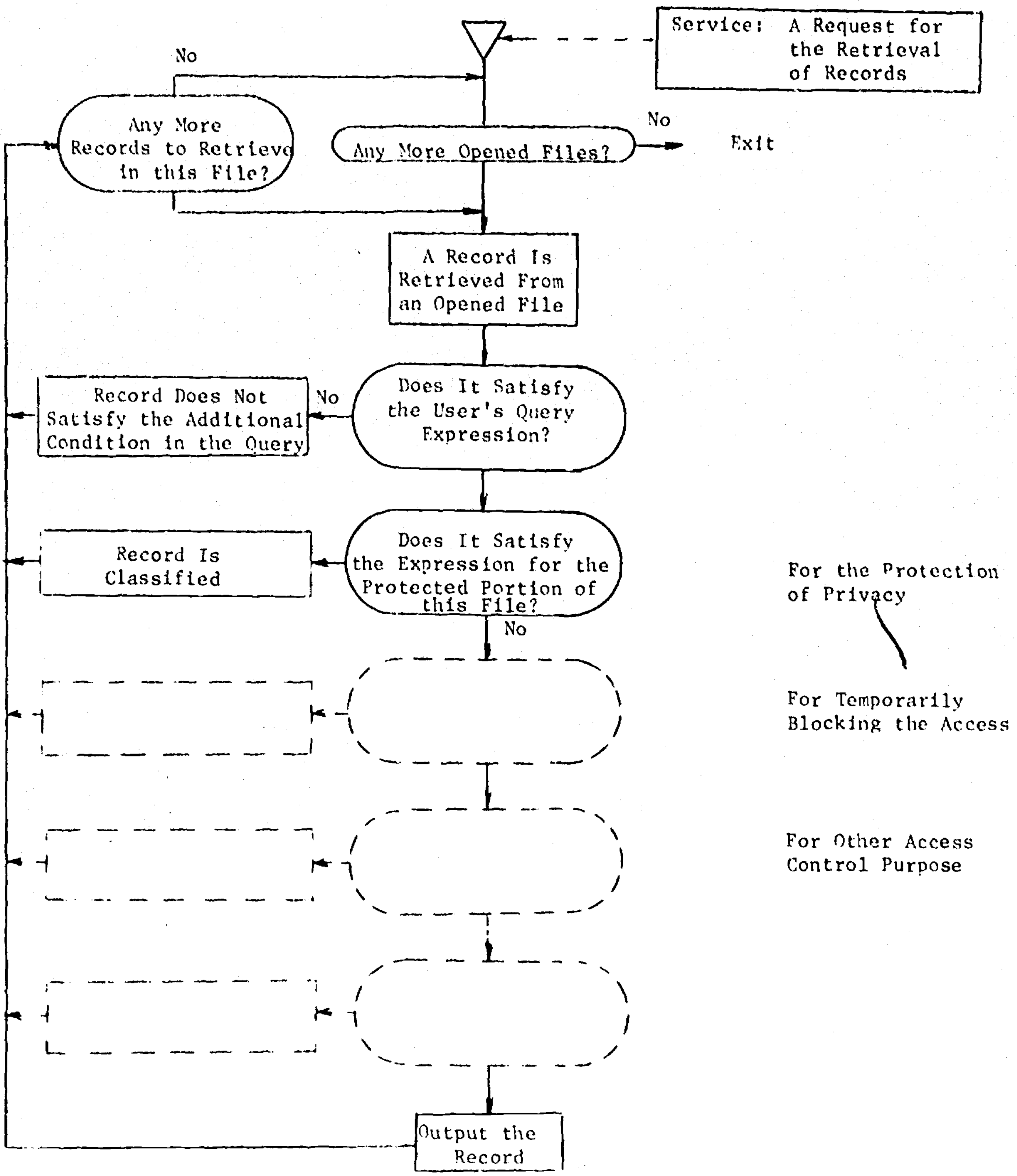
Let us consider one of the first systems to introduce logical access control, subfile protection and user-created control procedures [HsiaD68a, HsiaD68b]. The basic system protection mechanism uses capability lists which are called authority items. One authority item is associated with each user. The file of authority items is itself maintained much like other files on the system. The system is notable in its ability to offer protection of arbitrary subfiles. These subfiles are defined not by physical parameters but rather by logical descriptions, such as Boolean and arithmetic expression of key words and symbolic names. Within the authority item, logical expressions indicate for each file which records are inaccessible, which are temporarily blocked, which are presently opened for use, etc. As a record is retrieved in response to a user query, it is processed against these logical expressions to determine whether it should be output to the user. Figures 4 and 5 (from [HsiaoD68b]) illustrate these system features. The user is permitted to create a procedure associated with the file which he owns. This procedure would be invoked whenever access to the file is initiated by any user.

Such procedures can be arbitrarily complex, the only constraint being that the procedure return a 1 or 0 to the system indicating whether the access to the file is to be permitted or denied. This idea is expanded upon in [HoffL70, HoffL71] and subsequently became known as formularies.


A File
Records Permanently Protected from Access
Records Belonged to Open Portion of the File
Records that are Temporarily Blocked from Use by Others
Accessible Records
Inaccessible Records

The sets of records in a file specified by three types of logical description.

Fig. 4



Service: A Request for the Retrieval of Records
No
Any More Records to Retrieve in this File?
Any More Opened Files?
No
Exit
A Record Is Retrieved From an Opened File
Record Does Not Satisfy the Additional Condition in the Query
No
Does It Satisfy the User's Query Expression?
Record Is Classified
Does It Satisfy the Expression for the Protected Portion of this File?
No
For the Protection of Privacy
For Temporarily Blocking the Access
For Other Access Control Purpose
Output the Record

Fig. 5 - Expression Validations in Record Access

Example 4: Modeling Protection Mechanisms Based on Formularies.

Perhaps the most difficult protection mechanisms to be characterized in the conceptual model are the ones based on the user-defined authorization/verification procedures. These procedures are termed formularies in [HoffL70, HoffL71] are are also adopted in [CODAS71]. Essentially, a user could create whatever procedures he would like, and is allowed to have such procedures invoked at various phases of data base activity. Thus, access to data is determined by these procedures. One reason for the difficulty in characterizing procedural mechanisms in the conceptual model is the lack of definition of the procedure (i.e., formulary) itself. It is also not clear what is the environment for the invocation of the procedures. By this we mean what are data that are accessible to the procedure in order for the procedures to determine whether to permit or deny an access attempt.

Fortunately, the conceptual model is only descriptive in nature. Thus, we are not concerned with how a decision to permit or deny access is reached; rather, we only want to know what such a decision is. In effect, we could build the protection pattern equivalent to a given procedure (formulary) by "running" (or executing) the procedure on every element of the Extended Logical Data Base. Intuitively, then, we can characterize procedures (formularies) in terms of the corresponding conceptual model protection patterns.

IV. Four Types of Protection Specifications

Throughout the preceding discussion we have ignored two areas because we lacked the theoretical tools to deal with them. First, how do we resolve the protection of subsets of the Extended Logical Data Base which are assigned different protection by different protection specifications? Second, how do we get an orderly characterization of the creation, alteration and destruction of protection specifications and patterns? In order to treat these questions in a thorough way, we again develop some definitions.

<u>Defn</u>: The <u>user extraction operator</u> U returns for an element x of the Extended Logical Data Base the user identifier of the element.

$$U(x) = \text{user identifier}$$

<u>Defn</u>: The <u>access type extraction operator</u> A returns for an element x of the Extended Logical Data Base the access-type identifier of the element.

$$A(x) = \text{access-type identifier}$$

<u>Defn</u>: The <u>data extraction operator</u> D returns for an element x of the Extended Logical Data Base the data identifier of the element.

$$D(x) = \text{data identifier}$$

We also adopt the following notations. Lower case x and y will denote elements of the Extended Logical Data Base. Lower case p and q will denote protection specifications while upper case P, Q and R, possibly with subscripts, will denote protection patterns. Recall that the difference between a pattern and a specification is that the domain of a specification is a subset of the Extended Logical Data Base while the domain of a pattern is the entire Extended Logical Data Base. Finally, the set of all possible protection patterns over a given Extended Logical Data Base will be denoted by ψ.

Now we make some definitions of relations between protection patterns.

<u>Defn</u>: Two protection patterns, P and Q are <u>equal</u>, $P = Q$ if $\forall x\ (P(x) = Q(x))$.

In order to define a partial ordering on ψ we will say that for a, b $\in$ {deny, permit} $a \le b$ means a = deny or b = permit. Then we say that <u>a pattern P is a restriction of a pattern Q</u>, $P \le Q$, if $\forall x\ (P(x) \le Q(x))$. Informally, $P \le Q$ says that

a) There are some of the Extended Logical Data Base subsets which are permitted under Q but denied under P.
b) There are no subsets which are permitted under P but denied under Q.

We can conceptualize restriction as a partial ordering on the "strength" of the patterns, that is

a) $P \le P$ for any protection pattern.
b) $P \le Q$ and $Q \le P$ inplies that $P = Q$.
c) $P \le Q$, $Q \le R$ implies $P \le R$.

It is also possible to define the dual concept, <u>P is an expansion of Q</u>, $P \ge Q$, meaning $Q \le P$.

Given two protection patterns P and Q we define the <u>greatest lower bound (glb) of P and Q</u>, S_L = glb (P,Q) as follows

$$S_L(x) = \begin{cases} \text{deny, if } P(x) \text{ or } Q(x) = \text{deny}. \\ \text{permit, if } P(x) = Q(x) = \text{permit}. \end{cases}$$

A similar definition for the <u>least upper bound (lub)</u>, S_U = lub (P,Q) is also possible.

The glb and lub can also be defined for specifications.

$$s_L(x) = \begin{cases} \text{deny if } p(x) \text{ or } q(x) = \text{deny}. \\ \text{permit if } p(x) = q(x) = \text{permit}. \\ \text{undefined if } p(x) \text{ and } q(x) \text{ are undefined}. \end{cases}$$

The glb and lub characterize two constructive operations, since neither is necessarily equal to either P or Q. Thus both glb and lub represent possibilities for resolution of the protection of subsets in protection patterns which are not consistant. Another facet of this resolution is that it is how we can create a consistant (global) protection pattern from potentially not consistant (local) protection specifications. In attempting to make a consistant pattern, we may be fortunate enough that the individual specifications are consistant and that they cover the entire Extended Logical Data Base. It is more likely that one or both of these conditions does not hold. The question is, then, "How do we solve these problems?" First, let us consider the situation in which some of Extended Logical Data Base subsets are in the domain of no protection specification. Here, an essentially arbitrary decision on whether to permit or deny such accesses must be made. Such a decision creates, in effect, a protection specification covering these subsets. The problem of two or more specifications giving the same subset different inconsistant protections is far more difficult. The best working hypothesis is to suspend any access and refer such problems to a higher authority. This restrictive strategy corresponds to taking the lub of the contending specifications because if they differ it is because one says to permit access, and another says to deny it. Intuitively, it seems best to suspend the access while waiting for the higher authority to decide, because such actions can be simply reversed. On the other hand, once access has been incorrectly permitted, there is little that can be done to reverse the action. Such a higher authority may well be one of the users. It should be remarked that the resolution of inconsistant protection specifications is really the more philosophical question of how do we arbitrate between two conflicting requirements in system design. Each of the specifications is an assertion about the desired protection of the system. For this reason, no mechanical procedure for such resolution was suggested. The method suggested represents a compromise. By suggesting that some decisions can only be made by mechanisms and people outside the model, we endow it with great flexibility. Those decisions about which no controversy exists will be made mechanically and routinely.

To have a complete system we introduce two artifices which we shall employ later.

Defn: The completely protected data element, e_0 is defined such that for every triple (u, a, e_0) in the Extended Logical Data Base, $P((u,a,e_0))$ = deny.

Defn: The completely accessible data element, e_1 is defined such that for every triple $\forall(u, a, e_1)$ in the Extended Logical Data Base, $P((u,a,e_1))$ = permit.

Both definitions hold for any protection pattern, P. They give us a known property upon which we can depend.

All of the above formalism would be rather uninteresting if it could not be related to the practical considerations of system design. Recall that we are attempting to create a formal model for description and discussion of a wide variety of protection schemes. It is easy to lose sight of this and fall into a pedantic discussion of esoteric properties of the model. We have developed the notion of the protection pattern as the central descriptive means. By showing protection patterns to be partially ordered under restriction we were able to motivate the glb and lub as operators.

Now, we shall consider a more basic question, how do we create and modify the specifications which make up the patterns.

Defn: A context-free protection specification is a protection specification which does not depend upon the previous access attempts (permitted or denied) by any of the users governed by the specification.

We shall initially restrict our discussions to such memoryless specifications, since they are the most basic type.

The most primitive specification gives the protection of a single Extended Logical Data Base element, (u, a, d):

$$\text{TYPE.1}((u,a,d),\begin{Bmatrix}\text{deny}\\\text{permit}\end{Bmatrix}).$$

It should be obvious that, formally, this single operator is sufficiently powerful. Other operators will thus be measured against this one for flexibility. However, it should also be obvious that this operator is an awkward way to specify the protection of more than a few elements. Moreover, to create such specifications, the user must know explicitly all the users and access types to be governed by the specifications, an undesirable feature which violates our assertions about global knowledge.

The next step in flexibility is to allow the specification to cover, a subset, S, of Extended Logical Data Base, rather than a single element,

$$\text{TYPE.2}(S, \begin{Bmatrix} \text{deny} \\ \text{permit} \end{Bmatrix})$$

We observe that in naming a subset there is the practical problem of describing the subset to the protection system. Certainly, we shall not want to describe the set by enumerating its elements since we are no better off than with the TYPE.1 protection specification. We can ameliorate this problem by restricting the type of subset to one which can be described by parameterization. For example, the set definition $\{x | x \geq k\}$ implicitly creates a set selection function with parameter k indicating whether or not the element is a member of the subset. Although we would be somewhat premature in specifying the set selection functions which we shall use at this point of discussion, we nevertheless stipulate that whenever S is used in a protection specification TYPE.2, there is a set selection function of a few parameters associated with S. We shall continue this assumption throughout the rest of our discussion on the conceptual model.

The TYPE.2 specification enables the user to choose S such that every element of S has the same protection. What we would now desire is to relax this by allowing the user to specify that some subset S_1 is to be protected "like" some other subset S_2. First we must define what we mean by "like". Let f be a function from S_1 into S_2. Then for $x \varepsilon S_1$ we define p(x) to be p(f(x)) which is already defined since $f(x) \varepsilon S_2$ and p(y) is defined for all $y \varepsilon S_2$.

Since f maps S_1 into S_2, the specification can actually be simplified to

$$\text{TYPE.3 } (S_1, f).$$

A example of one such function is

$$f_{d_2}((u, a, d_1)) = (u, a\ d_2)$$

With such a function we can permit and deny the same set of accesses to two different data elements. That is, if $p(u, a, d_2)$ = permit, then $p(u, a, d_1)$ = permit. For the same user u the data element d_1 is protected with the same access attributes as the data element d_2. Obviously, we would like to extend this idea to other elements of the triple, for example, to give one user access privileges identical to those of another user, or to say that users may have a_2 access to some data element only if they have a_1 access to the element. The following general function can be used for all of these situations.

$$f\ (x;\ u_i,\ a_j,\ d_k)$$

$$= \left(\begin{Bmatrix} U(x), \text{ if } u_i = \text{null} \\ u_i, \text{ otherwise} \end{Bmatrix}, \begin{Bmatrix} A(x), \text{ if } a_j = \text{null} \\ a_j, \text{ otherwise} \end{Bmatrix}, \begin{Bmatrix} D(x), \text{ if } d_k = \text{null} \\ d_k, \text{ otherwise} \end{Bmatrix} \right)$$

where $x \in$ Extended Logical Data Base.

Let us consider the following special cases of f for $x = (u_o, a_o, d_o)$, an element in Extended Logical Data Base.

Case 1:

$$f\ (x;\ \text{null},\ \text{null},\ d_1) = (U(x),\ A(x),\ d_1).$$

But $U(x) = u_o$
$A(x) = a_o$

so that

$$f_1\ (x;\ \text{null},\ \text{null},\ d_1) = (u_o,\ a_o,\ d_1)$$

Intuitively, this function indicates that for the user u_o the data element d_o can be accessed in the same a_o manner as data element d_1.

Case 2:

$$f\ (x;\ u_2,\ \text{null},\ \text{null}) = (u_2,\ A(x),\ D(x))$$
$$= (u_2,\ a_o,\ d_o)$$

This function says that the user u_o will have the same a_o access as the user u_2 has to the element d_o.

Case 3:

$$f\ (x;\ u_1,\ a_1,\ \text{null}) = u_1,\ a_1,\ d_o)$$

This function says to make user u_o's a_o access to data d_o the same as user u_1's a_1 access to d_o.

It is, of course, possible to define many such functions. But first let us show how some common protection requirements can be translated into protection specifications of TYPE.3. Typical systems have a default protection for newly created objects. One of the strengths of our model is that a wide variety of methods can be used to achieve the same end. One way to get the default protection would be to explicitly specify whether each access was to be permitted or denied, using TYPE.1 specifications. Another way would be to group the permitted and denied accesses into subsets and use TYPE.2 specifications. This subset grouping would be fairly easy since, in general, the default rules are quite simple. The most natural way is to introduce another artifice. For each user we shall consider a default data object, d_{default}, such that the protection of newly created objects is "like" that of the default object, unless otherwise specified. More formally we have the following specification:

$$\text{TYPE.3}\quad (\{X \mid D(x) = d_{\text{new}}\},\ f\ (x;\ \text{null},\ \text{null},\ d_{\text{default}}))$$

That is, give every access of the form $(u,\ a,\ d_{\text{new}})$ the same protection as the corresponding access $(u,\ a,\ d_{\text{default}})$. The default specification can be changed, should the user desire, effecting only subsequently created objects. We can now close a potential loophole. The tacit assumption was made that the function mapped elements to other elements whose protection was defined. If we use this default idea for each newly created object, there will be no accesses for which the protection is undefined.

Let us consider one more type of specification. If we allow multiple valued mappings some additional possibilities occur. A user can create specifications that say "permit this access if any of these other accesses is denied." We shall use the following new specification

TYPE.4 (S , F , OP)

where S is an Extended Logical Data Base subset,

F is a finite set of functions (f_1, f_2, ...) mapping the Extended Logical Data Base into itself, i.e.,

$$f_i: \text{ELDB} \rightarrow \text{ELDB}, \text{ and}$$

OP is glb or lub.

The specification operates as follows:

$$p(x \varepsilon S) = OP(p(f_1(x)), p(f_2(x),\ldots)$$

recall that the glb will deny an access if it was denied under any of the specifications, and lub will permit it.

It is certainly possible to define other, more elaborate, specifications. However, one reaches a point of diminishing returns in the ability to apply them.

V. The Data Base Protection

The preceding development of protection specifications made no explicit use of the fact that we are dealing with data base systems. It is here that the present work makes a sharp difference from other efforts which were primarily concerned with protection in operating systems. Though certainly no less significant, the problems of protection in operating systems are different from those of data base systems. First, even a large multi-user operating system is concerned with the protection of a small number (like hundreds or at most a few thousand) of relatively large objects , the majority of which are some kind of physical resources such as disks and memory segments. A data base system is concerned with large numbers (tens of thousands to millions) of relatively small (tens to hundreds of words) objects such as the records and fields. Second, in an operating system the objects are mostly unrelated to each other, and where such relations exist between objects they are fairly simple. In modern data base systems a large variety of relations exist between objects. Indeed, this is a fundamental purpose of data base systems, to allow the retrieval of information from many different objects based upon some relationship or affinity among those objects. Thus, a data base protection system should, even must, use those relations to provide or enhance security.

In order to emphasize the relationship among objects of data base systems, a more formal model is needed. Although the topic of data base modelling has been an extremely popular one in recent years, reflecting the very real need for such formalism, we do not need the complexity which characterizes many of the models proposed.

The following terminology and ideas are mostly derived from (HsiaD70, WongE72) and are needed to express relations among elements of Extended Logical Data Base.

We start with two undefined terms: a set A of "attributes" and a set V of values. We shall leave these undefined to allow the broadest possible interpretation.

Defn: A record r is a subset of the Cartesian product $A \times V$, in which each attribute has one and only one value. We can consider r to be a collection of ordered pairs: (an attribute, its value).

Defn: An index for record r is a set of its attribute-value pairs which collectively characterize r.

For practical reasons we usually desire to choose pairs which are succinct. We shall call the ordered pairs in the index keywords. In the discussion which follows we shall denote keywords by K_i. From the definition of index above, we can characterize r by the keywords of its index.

Defn: Every record is assigned a unique address.
In practical systems, the address may give locational information, but we shall not concern ourselves with such specifics at this point, all we need is the uniqueness.

Defn: Associated with each keywork K in record R is the address of another record with the same keyword. We shall call this the pointer of r with respect to K or briefly the K-pointer. We allow the existance of null pointers to retain the uniformity of definition.

Defn: A list L of records with respect to keyword K (or briefly a K-list) is a set of records each containing K such that

1) the K-pointers are all distinct,
2) each non-null K-pointer gives the address of a record within L and L only,
3) there is a unique record in L not pointed to by any other record containing K, called the beginning of the list,
4) there is a unique record in L with a null K-pointer, the end of the list.

We may view this organization as a directed graph. The set of nodes corresponds to the set of records. There is an edge from r to r' for each K-pointer of r that is equal to the address of r'. We can label these edges with their associated keywords. From the definition of K-lists, no cycles exist in which every edge has the same label.

Defn: A set F of records is called a file if every K-list containing one or more of these records is contained in F. Every file is assigned a unique file name.

Defn: Let $R(K_i)$ be the set of all records containing keyword K_i.

Defn: Let $A(K_i)$ be the addresses of the records in $R(K_i)$.

Clearly, for a file F of m keywords we have

$$F = \bigcup_{i=1}^{n} R(K_i)$$

We shall say that a keyword K is true for a record r if the record contains K. Thus, every Boolean function $f(K_1, \ldots, K_n)$ is either true or not true for each record. $R(K_i)$ is then the set of all records for which K_i is true. Further, we denote $\bar{R}(K_i)$ to be the set of all records for which K_i is not true.

Similarly:

$$R(K_i) \cup R(K_j) = \{\text{records for which } (K_i \text{ or } K_j) \text{ is true}\}$$

and

$$R(K_i) \cap R(K_j) = \{\text{records for which } (K_i \text{ and } K_j) \text{ is true}\}$$

Extend $Q = \{R(K_i),\ i = 1, 2, \ldots, n\}$ to a Boolean algebra $B(Q)$ by taking unions ($\cup$), intersections ($\cap$) and complements ($-$). It is then clear that any data base query can be answered by one element of $B(Q)$. Define $C_1, C_2, \ldots, C_{2^n}$ as the 2^n intersections of the form

$$\bigcap_{i=1}^{n} R^*(K_i) \text{ where } R^* = R \text{ or } \overline{R}$$

and assume the C_i are numbered so that $C_1, \ldots, C_m$ are non-empty and $C_{m+1}, \ldots C_{2^n}$ are empty.

<u>Defn</u>: $C_1, C_2, \ldots, C_m$ as defined above are called the <u>atoms</u> of $B(Q)$.

It is then easy to see the following

<u>Theorem</u>: Let $C_1, C_2, \ldots, C_m$ be the atoms $B(Q)$, then

1) $C_j \varepsilon B(Q)$ for $j = 1,2,\ldots,m$;
2) C_j and C_k are disjoint if $j \neq k$;
3) $U \varepsilon B(Q)$ implies that for each j, $U \cap C_j$ is either empty or is C_j.
4) Every $U \varepsilon B(Q)$ is the union of some of the C_j.

Intuitively, the atoms are the elementary expressions of keywords which characterize the elements of the logical data base. Thus, with the development of the atoms we have a means of explicitly characterizing the Logical Data Base. These ideas will be illustrated in the next section.

In our early assertion, we stated that it was possible to enumerate every answer to any question that the user could put to the data base. This result can be achieved by characterizing any arbitrary subset of the

Logical Data Base as a collection of atoms. Furthermore, it is possible to completely characterize a protection pattern as either the subset of permitted or of denied accesses. In other words, we can make another assertion concerning protection patterns. It is possible to specify protection patterns on any subset of the Extended Logical Data Base for the purpose of either permitting or denying the access to the subset. The assertion goes without saying that every retrievable item in the Logical Data Base may be protected since retrieval specifications are like TYPE.3 specifications. Although this statement is simple and basic, it is fundamental. It offers the complete protection of every possible element in the data base whether the elements are small fields, large files, data or programs. Again, the impact of such formalism is that it allows us to prove the assumption that we can find every piece of data in the Logical Data Base.

In order for a system to function as described, it must meet the following requirement.

Design Requirement: The only source of addresses is the K-pointers. No user can fabricate or modify addresses.

Without such a requirement, a user could supply fabricated addresses and circumvent any controls on the data base.

Now we can introduce a new type of protection specification in the conceptual model.

$$\text{TYPE.5 } (U,\ Q,\ \left\{ \begin{array}{c} \text{permit} \\ \\ \text{deny} \end{array} \right\})$$

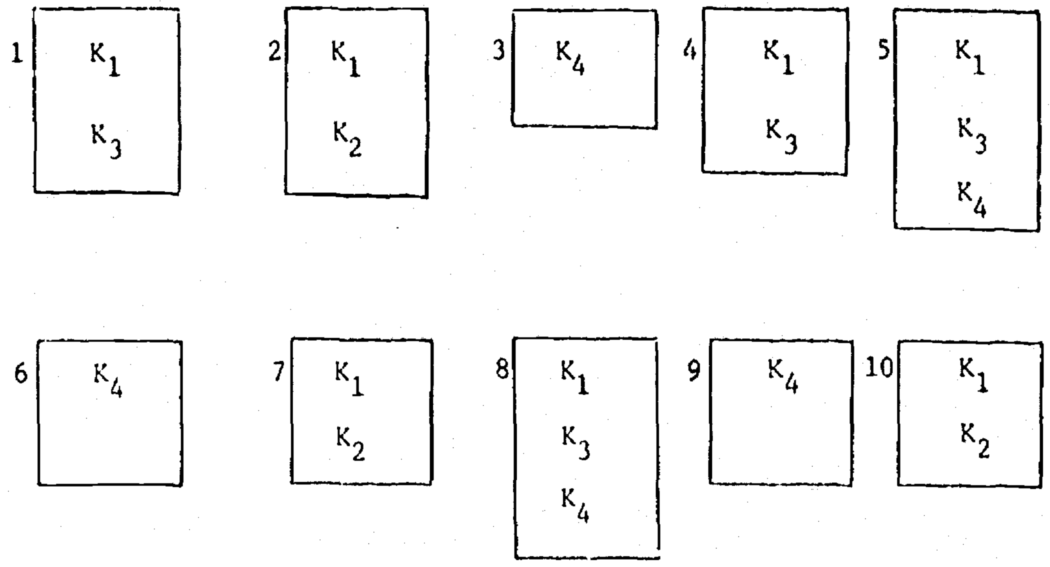
where U is a set of users
and Q is a Boolean expression of keywords.

Note that this is a logical protection specification, and that the specification is posed completely in terms of the users view of the data base, i.e., Boolean expression of keywords. It may be that the set of records satisfying Q is null, in which case the specification has no effect.

VI. An Example of Data Base Protection Specification

Let us illustrate the use of the TYPE.5 protection specification with a simple example as follows:

A small data base consists of ten records which are characterized by four different keywords. The record addresses and their keywords are depicted on Fig. 6 and the structure of the data base is depicted on Fig. 7. For the structure, we use the numbered circular node to denote the record at the address so numbered. Along with each edge directed to a node there is a keyword indicating that the node (therefore, the record) is characterized by the keyword. If there are several edges directed to a node, then the node is characterized by several keywords. For example, the record at 8 is characterized by keywords, K_1, K_3 and K_4. Obviously, Fig. 7 is a graphical representation of Fig. 6. In this discussion the directory is a special node which can only be accessed by the system, thus no keyword leading to the directory is known. In general, directory may be records; access to and protection of directories can be handled in the same way as records. However, for ease of discussion, we shall not consider the generalization in this example. The atoms of the data base are listed on Fig. 8.



$R(K_1) = 1,2,4,5,7,8,10$

$R(K_2) = 2,7,10$

$R(K_3) = 1,4,5,8$

$R(K_4) = 3,5,6,8,9$

Fig. 6 The Records of a Data Base

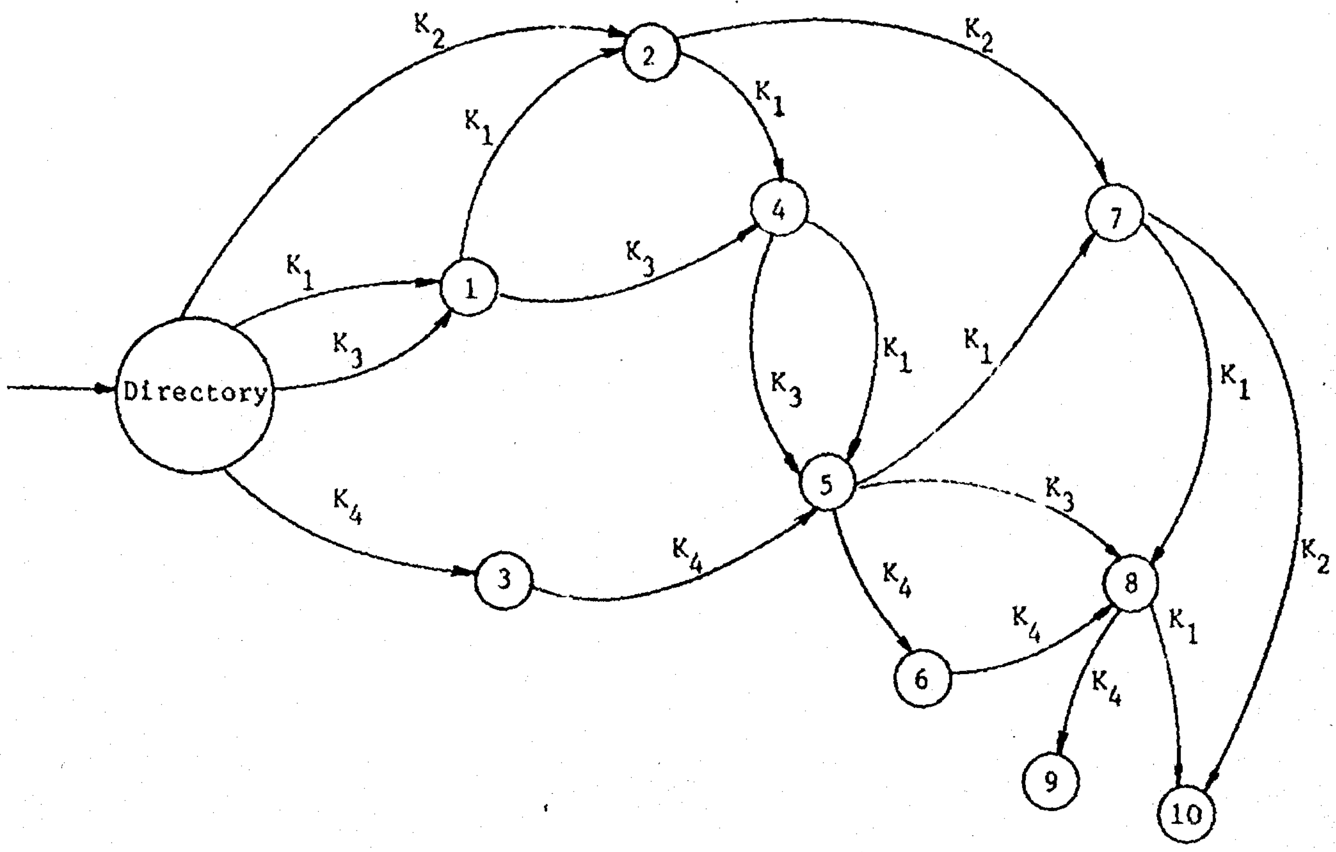


Fig. 7 The Structure of the Data Base

Atoms	Addresses of Records satisfy the atom
$K_1 \wedge \bar{K}_2 \wedge \bar{K}_3 \wedge \bar{K}_4$	1, 4
$K_1 \wedge K_2 \wedge \bar{K}_3 \wedge \bar{K}_4$	2, 7, 10
$\bar{K}_1 \wedge \bar{K}_2 \wedge \bar{K}_3 \wedge K_4$	3, 6, 9
$K_1 \wedge \bar{K}_2 \wedge K_3 \wedge K_4$	5, 8

Fig. 8 The Atoms of the Data Base

Now consider the following specification of a protection pattern:

TYPE.5 ({U1}, Q, deny)

where $Q = K_2 \wedge ((K_1 \wedge \bar{K}_4) \vee (\bar{K}_3 \wedge K_4))$.

This specification indicates that the system must deny user U1 access to any record in the data base for which Q is true. Thus, this user has only a portion of the data base for access and his view of the data base is depicted in Fig. 9 . Let us elaborate on the last remark. The Boolean expression Q can be decomposed into disjunctive normal form as follows:

$$\begin{aligned} Q &= K_2 \wedge ((K_1 \wedge \bar{K}_4) \vee (\bar{K}_3 \wedge K_4)) \\ &= (K_1 \wedge K_2 \wedge \bar{K}_4) \vee (K_2 \wedge \bar{K}_3 \wedge K_4) \\ &= (K_1 \wedge K_2 \wedge K_3 \wedge \bar{K}_4) \vee (K_1 \wedge K_2 \wedge \bar{K}_3 \wedge \bar{K}_4) \vee \\ &\quad (K_1 \wedge K_2 \wedge \bar{K}_3 \wedge K_4) \vee (\bar{K}_1 \wedge K_2 \wedge \bar{K}_3 \wedge K_4) \end{aligned}$$

By comparing the above four conjuncts derived from Q with the atoms of the data base listed in Fig. 8, we learn that only the conjunct $(K_1 \wedge K_2 \wedge \bar{K}_3 \wedge \bar{K}_4)$ is an atom. Furthermore, we learn from the same Fig. 8 that the records for which the atom is true are the records at 2, 7 and 10. Thus, we can conclude from the TYPE.5 specification that the user U1 is to be denied access to records at 2, 7 and 10. Thus, the user U1's view of his data base, as depicted on Fig. 9, does not include the records at 2, 7 and 10.

The use of atoms to partition the data base into mutually exclusive subsets for protection specification is powerful and effective. It is powerful because all retrievalbe information can be protected. Since any data retrieval is a response to the user's query and a query is a Boolean expression of keywords, the same expression can be used for protection specification. It is also effective because the atom is a logical specification which is independant of the structure and implementation of the data. The reliance on keywords is not a restriction since keywords may be either symbolic names in their most sophisticated form or numeric indentifiers in their primitive form.

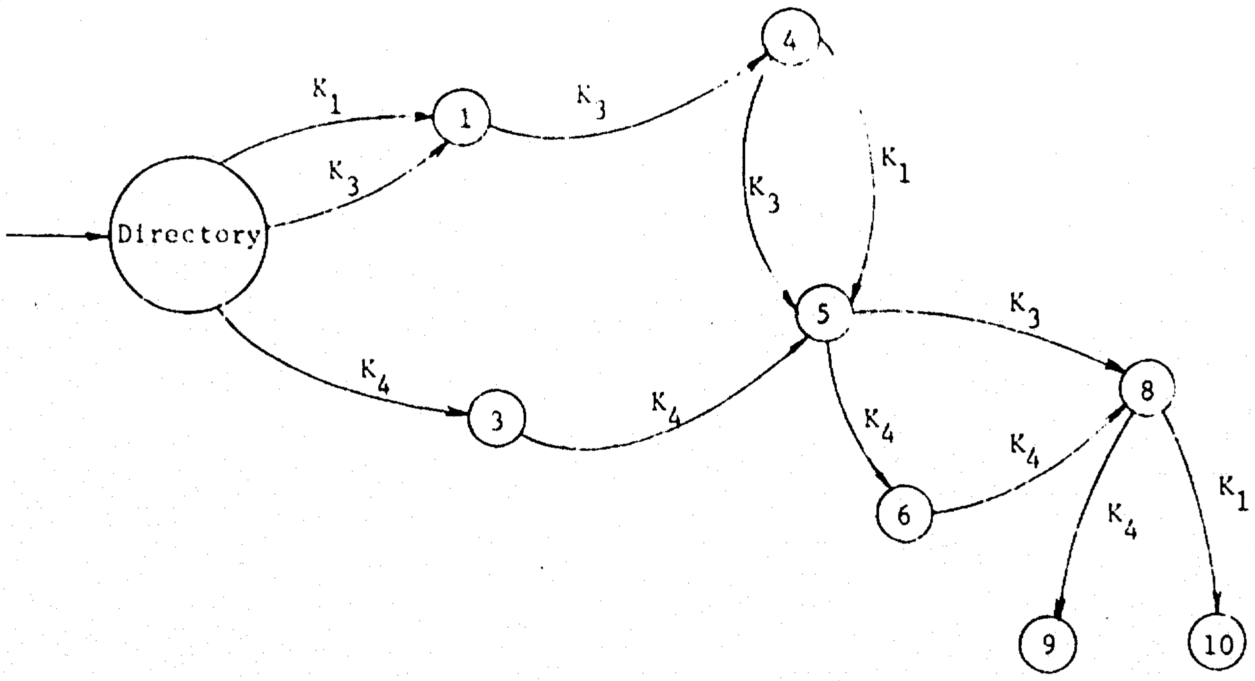


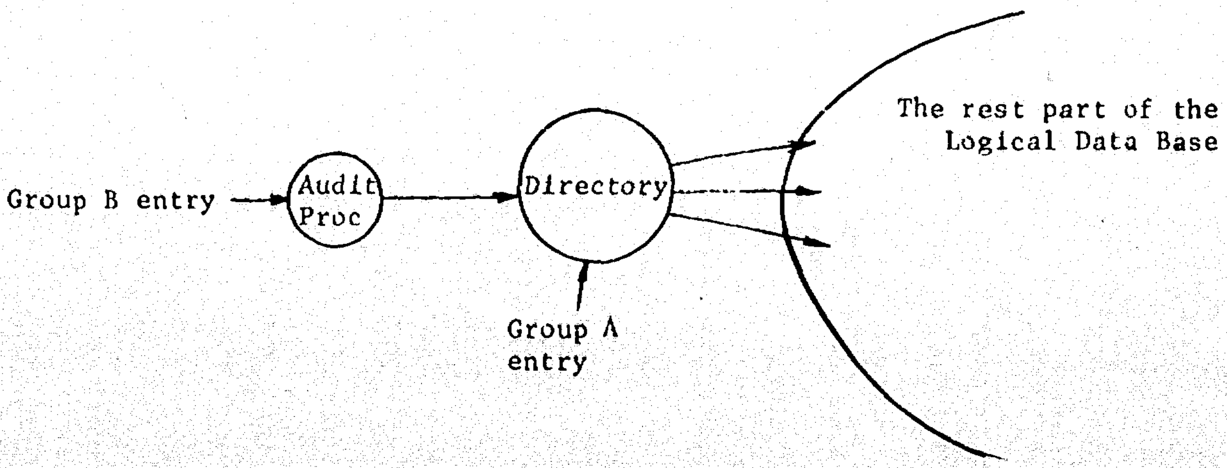
Fig. 9 The User's View of His Data Base

VII. Access Types

As we indicated earlier, to be a useful model, we should be able to accommodate a variety of different access types. The following consideration will lead to a more unified treatment of access types. For each record we allow the possibility of a special attribute-value pair, (proc. name, proc.) where proc is a procedure. When a record with such a procedure is accessed, control is passed to the procedure. The procedure uses only the information in the other attribute-value pairs in the record. To achieve uniformity of definition we shall assume that every record has a proc.name-proc pair, although it may be null. Further, such pairs always have null pointers.

Thus, the particular access type for a given node is dependent upon the path followed through the data base in getting to that node. This solves the problem posed by multiple access types to the same data element. The particular path followed is of course determined by the Boolean expression of keywords supplied by the user. However, in this way the Boolean expression not only determines the set of records to be accessed but also the types of access to be involved. The following example demonstrates the utility of this formulation.

Consider a situation in which the users can be divided into two disjoint groups, A and B. Those in group A are skilled and trustworthy and are thus to be allowed free access to the data base. Those in group B are more suspect. We therefore desire to make a record of every one of their accesses. The data base may be structured as follows



Each user in group B must make all his accesses through the auditing procedure, which has a pointer to the normal search procedure, the directory node. On the other hand, group A may access the directory node directly.

In this new formulation, the directory is not merely an entry into the data base, but also a node at which the normal search procedure is triggered. We term this the directory node. As it is shown in the example, it may be desirable to incorporate more than one entry into the data base. We shall term all these entries as gate nodes. Recalling an earlier discussion of basic terminology, we see now that the function of the identification procedure is to direct the user to the proper gate node.

VIII. Summary

Let us conclude our discussion of the conceptual model by summarizing what we have done.

The discussion was started by formal and contextual definitions of the working vocabulary. Using the intuitive idea of the dichotomy of all possible accesses between those which are at some instant permitted and those which are denied, we formalized the idea with the Extended Logical Data Base and protection specifications and patterns. These specifications and patterns encompass the traditional identification, authentication, and verification procedures by recognizing that these procedures, though possibly so complex as to preclude meaningful analysis, can be compared solely on the basis of their "answers" to specific access attempts. We developed a limited "calculus" of protection patterns, suggesting both comparative and generative operators. Finally, we defined and demostrated a variety of protection specifications. Most significantly, we demonstrate that it is possible to protect any accessible (or retrievable) data in a data base. The proposed protection specification (TYPE.5) is independent of the structure and implementation of the data base.

REFERENCES

CODAS71	CODASYL Data Base Task Group, Report to the Programming Language Committee, Rev. Apr. 71.
DennJ66	J. B. Dennis and E. C. VanHorn, "Programming Semantics for Multiprogrammed Computations," CACM, 9, 3 (March 66), 143.
GrahG71	G. S. Graham, "Ptotection Structures in Operating Systems," (M. S. Thesis), Dept. of Computer Science, University of Toronto, (Aug. 71).
GrahG72	G. S. Graham and P. J. Denning, "Protection - Principles and Practice," Proc. 1972 SJCC, 40, AFIPS Press, 417-429.
HoffL70	L. J. Hoffman, "The Formulary Model for Access Control and Privacy in Computer Systems," (PhD. Dissertation, Stanford), Stanford Linear Accelerator Center Report SLAC-117, (May 70).
HoffL71	L. J. Hoffman, "The Formulary Model for Flexible Privacy and Access Control," Proc. 1971 FJCC, 39 AFIPS Press, 587-601.
HsiaD68a	D. K. Hsiao, "A File System for a Problem Solving Facility," (PhD. Dissertations Penn.), The Moore School of Electrical Engineering, University of Pennsylvania, (May 1968) (Also available through NTIS as report AD 671-826).
HsiaD68b	D. K. Hsiao, "Access Control in an On-line File System," File Organization - Selected Papers from FILE68, an I.A.G. Conference by Student literature Ab, Lund, Sweden, 1968.
HsiaD70	D. K. Hsiao, and F. Harary, "A Formal System for Information Retrieval from Files," CACM 13, 2 (Feb. 1970), 67-73.
PopeG73	G. J. Popek, "Correctness in Access Control," Proc. 1973 ACM National Conf., 236-241.
WongE71	E. Wong and T. C. Chiang, "Canonical Structure in Attribute Based File Organization," CACM 14, 9 (Sept. 1971), 593-597.