INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms
300 North Zeeb Road
Ann Arbor, Michigan 48106
HARTSON, H. Rex, 1941-
LANGUAGES FOR SPECIFYING PROTECTION REQUIREMENTS
IN DATA BASE SYSTEMS--A SEMANTIC MODEL.
The Ohio State University, Ph.D., 1975
Computer Science

Xerox University Microfilms, Ann Arbor, Michigan 48106
LANGUAGES FOR SPECIFYING PROTECTION REQUIREMENTS
IN DATA BASE SYSTEMS—A SEMANTIC MODEL

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

by

H. Rex Hartson, B.S.E., M.S.E. (University of Michigan)
M.S. (Ohio State University)

* * * * *

The Ohio State University

1975

Reading Committee:
Professor D.K. Hsiao
Professor M.T. Liu
Professor D.J. Moore

Approved by

Advisor

Department of Computer and
Information Science
Let us be thankful; then, for all the right choices we make when we have to choose; for all the unseen influences that help us to choose right; for whatever holds us, or diverts from a course that is not our true course; for any denial of apparent advantage or present ease which constrains us toward the fulfillment of a nobler destiny.

—Edward Sandford Martin

No metaphysician ever felt the deficiency of language so much as the grateful.

—C. C. Colton, *Lacon*

This dissertation is the result of one research effort in the Project on Security and Data Secure Systems. The project 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. Dr. Douglas S. Kerr, Associate Professor of Computer and Information Science, and Dr. M. T. Liu, Associate Professor of Computer and Information Science, are investigators on the project. Presently, there are seven other Graduate Assistants in the project: R. I. Baum, D. Cohen, J. Hennings, R. S. Knablein, C. J. Nee, G. Severson, and D. Wong. In addition, there are two systems analysts in the project, A. Breene and N. Kaffen. Ms. Sandy Rich is the secretary for the project. T. Rodeheuffer, D. Schmaltz, and R. Kaiser, undergraduate students, also participate in the project activities.

I gratefully acknowledge the financial support of the Office of Naval Research through contract N00014-75-C-0573, and I thank Mr. Marvin Denicoff personally for his encouragement of the project on data secure systems. I also
want to express my gratitude to my dissertation advisor, Professor David K. Hsiao, for his help with this research. It was he who introduced me to the topic of access control. It was he who, despite my shortcomings as an author, expended a great deal of time and effort in trying to help me express myself in a readable manner. Dr. Hsiao has contributed greatly to the maturation of this work. I also thank Professors Ming-Tsan Liu and Daniel J. Moore for serving on my reading committee. Sandy Rich deserves credit for a skillful job of typing a major part of the dissertation into the computer. Special thanks go to David Wong and James Hennings for their interest in the topic and their useful suggestions. Also the PDP-10 staff is acknowledged for their help and cooperation and the Instruction and Research Computer Center for computer time to produce the index. In addition, I am indebted to Dr. James E. Rush and the Ohio College Library Center for their timely support during part of my stay in Columbus.

My deepest appreciation is reserved for my wife, Lynda. I can only admire how she competently and lovingly met the problems of raising a family, while artfully bearing with the ups and downs of a temperamental graduate student. Her unfailing encouragement sustained my efforts throughout.

A person reaches a goal like this through the help of many people. In particular my parents-in-law, Mr. and Mrs. Chester Q. Fike, have been very supportive of my efforts in many ways. I hope I can prove worthy to the confidence they have invested in me. A special thanks also goes to Mark Ebersole, for his helpful discussions at many points in the development and for his considerable effort in developing the program used to create the index, the same program that I use to produce a large index of my file of technical papers. Thanks to my many other friends and relatives who have encouraged and helped me in many real ways. I will never forget.

Lastly, it is fitting that, in this document representing a major educational milestone, consideration be given to those who, educationally, have had an important effect on me. The most important effect was from my parents, Mr. and Mrs. Earl C. Hartson, who unfortunately have not lived to share in this accomplishment. My original interest in computers is due to Miss Catherine D. Meehan, my high school mathematics teacher, who sponsored a mathematics club. That club provided the opportunity for
Joseph Marsh, James Schlee, and I to build an operational digital computer from the relays of three pin-ball machines. I have been deeply interested in computers ever since. Thank you, Miss Meehan.

Another important person in my education has been Professor Norman R. Scott, who, during our several years of pleasant association at the University of Michigan, inspired me to continue in computer science.

This work is dedicated to Lynda, Holly, and Cole.

Columbus, Ohio
August 1975

H. Rex Hartson
VITA

March 17, 1941

Born—New York, New York

1960-1967

Assistant, Digital Computer Engineering Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor, Michigan

1964

Consultant, Teaching Computer Science, Ford Motor Co., Detroit, Michigan

1965

B.S.E. (Electrical Engineering), The University of Michigan, Ann Arbor, Michigan

1965-1966

Consultant, Teaching Computer Science, Bendix Research Laboratories, Southfield, Michigan

1966-1967

Lecturer, Department of Electrical Engineering, The University of Michigan, Ann Arbor, Michigan

1967

M.S.E. (Electrical Engineering), The University of Michigan, Ann Arbor, Michigan

1967-1971

Engineer and Senior Engineer, Xerox Corporation, Research Laboratories, Webster, New York, in these groups: Computer and Information Science Area, Information Systems Division, and Corporate Exploratory Development Laboratory

1971-1972

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

1972

M.S. (Computer and Information Science), The Ohio State University, Columbus, Ohio
<table>
<thead>
<tr>
<th>Year</th>
<th>Role</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1973</td>
<td>Research Assistant, Department of Computer and Information Science</td>
<td>The Ohio State University, Columbus, Ohio</td>
</tr>
<tr>
<td>1973-1974</td>
<td>Programmer, The Ohio College Library Center</td>
<td>Columbus, Ohio</td>
</tr>
<tr>
<td>1974-1975</td>
<td>Research Associate, Department of Computer and Information Science</td>
<td>The Ohio State University, Columbus, Ohio</td>
</tr>
</tbody>
</table>

**PUBLICATIONS**


**FIELDS OF STUDY**

Major Field: Computer and Information Science

- Studies in Systems Programming. Professor David K. Hsiao
- Studies in Computer Architecture. Professor Ming-Tsan Liu
- Studies in Information Storage and Retrieval. Professor James E. Rush
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Section</th>
<th>Illustration Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.1</td>
<td>Portion of an Access Matrix</td>
<td>10</td>
</tr>
<tr>
<td>II.1</td>
<td>Overview of Access Control Processes</td>
<td>15</td>
</tr>
<tr>
<td>IV.1</td>
<td>Full vs. Partial Rejection, Case 1: Request Contained in Intersection</td>
<td>77</td>
</tr>
<tr>
<td>IV.2</td>
<td>Full vs. Partial Rejection, Case 2: Request Overlapping</td>
<td>77</td>
</tr>
<tr>
<td>IV.3</td>
<td>Partitioning for OR and ‘AND’ operations</td>
<td>79</td>
</tr>
<tr>
<td>IV.4</td>
<td>A request Not Covered</td>
<td>79</td>
</tr>
<tr>
<td>IV.5</td>
<td>States in Terms of Resource Values</td>
<td>81</td>
</tr>
<tr>
<td>IV.6</td>
<td>The Resource and State Projection of The Security Space</td>
<td>82</td>
</tr>
<tr>
<td>IV.7</td>
<td>The Final Authorized Subspace</td>
<td>82</td>
</tr>
<tr>
<td>IV.8</td>
<td>The Subspace for a User Request</td>
<td>85</td>
</tr>
<tr>
<td>IV.9</td>
<td>Projection of Security Space for Fixed State, Multi-User Case</td>
<td>85</td>
</tr>
<tr>
<td>IV.10</td>
<td>Access Conditions</td>
<td>87</td>
</tr>
<tr>
<td>V.1</td>
<td>Lexical Translation</td>
<td>95</td>
</tr>
<tr>
<td>V.2</td>
<td>Programming Language Compiler</td>
<td>95</td>
</tr>
<tr>
<td>V.3</td>
<td>Programming Language Interpreter</td>
<td>97</td>
</tr>
<tr>
<td>V.4</td>
<td>Protection Language Translation</td>
<td>97</td>
</tr>
<tr>
<td>V.5</td>
<td>Programming Language Definitions in Terms of A Common Base Language</td>
<td>98</td>
</tr>
</tbody>
</table>
B.1 The Relationship Between U and S.............................. 149
F.1 States of a Binary Three-Resource System.............. 174
F.2 Overlapping of State Subsets........................... 174
F.3 The Security Space Projection of
Authorizers and Users............................................. 176
F.4 The Security Space Projection of
Users and Resources.................................................. 176
F.5 The Security Space Projection of
Resources and States.................................................. 177
F.6 The Security Space Projection of
Users, Resources, and States........................................ 179
F.7 The Projection of Users and Resources.................. 180
F.8 The User and State Projection
For User Group U1..................................................... 180
F.9 The User and State Projection
For User Group U2..................................................... 181
F.10 The User and State Projection
For User Groups U1 and U2......................................... 181
F.11 p1 in the User and Resource Projection.............. 184
F.12 p2 in the User and Resource Projection.............. 184
F.13 p1 in the Resource and State Projection............. 185
F.14 p2 in the Resource and State Projection............. 185
F.15 p1 and p2 in the Resource and
State Projection....................................................... 186
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE AND ACKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>VITA</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>vii</td>
</tr>
<tr>
<td>ORTHOGRAPHY AND NOTATION</td>
<td>xv</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION AND BACKGROUND</td>
<td>1</td>
</tr>
<tr>
<td>The Need for Access Control</td>
<td></td>
</tr>
<tr>
<td>The Setting</td>
<td></td>
</tr>
<tr>
<td>The Need for Protection Languages at Many Levels of Sophistication</td>
<td></td>
</tr>
<tr>
<td>The Approach to Presentation</td>
<td></td>
</tr>
<tr>
<td>II. CONCEPTS AND TERMINOLOGY</td>
<td>13</td>
</tr>
<tr>
<td>Introduction</td>
<td></td>
</tr>
<tr>
<td>Separation of Policies from Mechanisms</td>
<td></td>
</tr>
<tr>
<td>Ownership</td>
<td></td>
</tr>
<tr>
<td>System Administrator</td>
<td></td>
</tr>
<tr>
<td>Subownership</td>
<td></td>
</tr>
<tr>
<td>Conditional Ownership</td>
<td></td>
</tr>
</tbody>
</table>
Sets of the Model

Users
Authorizers
Resources
Elemental Data Operations
Notation
Structuring of the Sets

System States and Conditions

Resource Values
Restrictions on Resource Values
Conditions

Introduction to Authorization and Enforcement

III. THE SECURITY SPACE AS A 5-DIMENSIONAL DISCRETE SPACE

Definition of the Space
Access Requests and Queries
Authorization Specifications
Examples of Authorization Specifications
The Role of Full Specifications
Access History Keeping

Examples of History Keeping

x
Auxiliary Program Invocation in the Security Space

Examples of Auxiliary Program Invocation

Extended Resources

interlocking for Protection During Shared Use of Resources

IV. PROCESSES OF THE MODEL .................................................. 56

Modelling the Authorization Process in the Security Space

Validation—The First Step

Granting Access Rights

Withdrawing Access Rights

Other Tasks

Modelling the Enforcement Process in the Security Space

A Dilemma in Enforcement

Implications of Overlapping User Groups—An OR Operation

Implications of Overlapping Resource Units—An AND Operation

Interpretations of the Enforcement Process

xi
The Enforcement Process as a Series of Steps
Basic Memoryless Case
Full Specification Case
Partial Rejection by the Enforcement Process
Examples of Authorization and Enforcement in the Security Space

V. THE NATURE OF PROTECTION LANGUAGES ........................ 89
Introduction
The Nature of Protection Languages
Translation of Protection Languages
Syntax and Lexical Translation
Semantic Translation
Approach to Languages
Categories of Constructs

VI. PROTECTION LANGUAGES—COMPONENTS
AND CONSTRUCTS ........................................................................ 102
A Sample Protection Language
The Protection Language Components and Their Translation into Entities of the Model
Definitional Constructs
Operational Constructs
Advanced Protection Service Constructs

xii
Semantic Specification Constructs

Summary of PLC

The Design of Protection Languages

A Note on Implementation

VII. SUMMARY........................................................................ 140

Appendix

A. A GENERAL SET NOTATION FOR CONDITIONS.................. 145
B. SETS DEFINED BY CONDITIONS.......................................... 147
C. A SAMPLE PROTECTION LANGUAGE—SYNTAX AND SEMANTICS. 150
D. FORMAL REPRESENTATION OF AUTHORIZATION AND ENFORCEMENT........................................... 162
E. EXAMPLES OF CONDITIONS............................................... 165

User Group Defining Conditions

Resource Unit Defining Conditions

Access Conditions

Classes of Conditions

F. THE ROLE OF BASIC AUTHORIZATION IN THE SECURITY SPACE—AS DEPICTED BY PROJECTIONS OF THE SECURITY SUBSPACES...................................................... 173

The Relation Between States and Conditions
The Relation Between Authorizers and Users—The Valid Subspace
The Relation Between Users and User Groups—One View of the Authorized Subspace
The Relation Between Individual Resources and Resource Units—Another View of the Authorized Subspace

G. THE ENFORCEMENT PROCESS WITH VARIABLE SEMANTICS. 187

CITED REFERENCES .................. .................. 190

INDEX .......................................................... 200
Orthography and Notation

Editing and formatting programs on the DECsystem-10 of the Department of Computer and Information Science were used to prepare the manuscript of this dissertation. The final output is, therefore, subject to the limitations of the output printer. The character set is restricted and the text is completely linearized (i.e., there can be no subscripts and superscripts in the traditional positions). To overcome the first limitation, certain mathematical symbols have been added manually. In response to the second limitation, no superscripts are used and the representation of subscripts is as follows. Subscripts are indicated by square brackets: \( A[l] \). In cases where no ambiguity arises, subscripting is simplified for easier reading by omitting the square brackets: \( A_l, b_3 \). Parentheses are used to surround n-tuples: \((a_1,a_2,\ldots,a[n])\). Parentheses are also used around arguments of functions: \( f(x) \). Sets are enclosed in braces: \( \{s_1,s_2,s_3\} \). Bibliographic references are given as a series of upper case letters and numerals within square brackets: [HARTH75].

Most mathematical symbols used in this work will be found in the following summary.

+ logical OR operator or arithmetic addition operator depending on context: \( A + B \)

& logical AND operator: \( X \& Y \)

\( \in \) set membership: \( x \in X \) (\( x \) is a member of \( X \))

\( :) \) set restriction: \( S = \{x:x>5\} \)

** arithmetic exponentiation operator: \( 2^{10} = 1024 \)
- complement or relative complement of a set: \(-A\), \(B-C\)

\(\neg\) logical NOT or negation operator: \(\neg B\)

\(\Rightarrow\) material or logical implication: \(p\Rightarrow q\)

\(\iff\) logical equivalence: \(p\iff q\)

\(\rightarrow\) mapping of a function: \(f: X \rightarrow Y\)

\(<-\) replacement or assignment

\(=\) equivalence of functions: \(f(x) = g(y)\)

\(\cap\) set intersection operator: \(A \cap B\)

\(\cup\) set union operator: \(A \cup B\)

\(\emptyset\) the null set

\(0\) the number zero

\(\prod_{i=1}^{n}(A[i])\) Boolean algebraic conjunction or logical AND:

xvi
\[ \sum_{i=1}^{n} \text{Boolean algebraic disjunction or logical OR:} \]
\[ \sum_{i=1}^{n} (A[i]) \]
\[ \sum \text{an alphabet of symbols} \]
\[ \sum^* \text{the set of all strings over the alphabet } \sum \]
\[ \times \text{cartesian product of sets: } \times (B[i]) = B1 \times B2 \times \text{...} \times B[n] \]

In deference to the equal rights of women there are two symbols which require explanation. The terms "him" and "his," when used in reference to an unknown person, are to be construed to be a general reference to a person, either masculine or feminine.
CHAPTER I

INTRODUCTION AND BACKGROUND

The personal life of every individual is based on secrecy, and perhaps it is partly for that reason that civilized man is so nervously anxious that personal privacy should be respected.
—Anton Chekhov, The Lady with the Dog

Security is having a good infield behind you.
—Charlie Brown, by Charles M. Schultz

The Need for Access Control

Lack of control over present day information systems presents a problem with explosive potential. It is a problem which could conceivably touch nearly every person in every nation. The burgeoning problem of the very real information explosion has induced the use of computers to store, organize, analyze, and disseminate information in government, education, business, and industry. Such widespread use of information systems and data bases introduces some serious issues regarding the control of their use. People need only to read the news to be aware of the existence of some of these issues. To most of these individuals there is little need to motivate the case for privacy and security controls in computing and data base systems. The implications of such controls in business computing are enormous, as illustrated by a recent insurance
INTRODUCTION AND BACKGROUND
THE NEED FOR ACCESS CONTROL

imbroglio [PODCG73]. However, nowhere is there more danger from improper use of computers than in the government. Here, security and privacy pose serious problems. Security is a key issue in areas such as defense and foreign affairs. Privacy is probably a more important difficulty with data bases of information on arrests and crime; credit organizations [FISCL73, PATRR74], mailing list activities, and medical information systems [WAREW74, GABRE74]. The impact of these issues is underscored by the continual attention they receive from legislators, civil rights groups [FLATL75], and data processing professionals [ARNSC75]. Nor is the problem confined to one country. For example, Britain [HANLJ74] is very concerned and Sweden [SWEDE73] put the first national data act into effect in July 1973. The United States followed with a limited privacy act in late 1974 [FRENN75]. Access control, which plays a major role in this dissertation, will be shown to provide mechanisms necessary to achieve security and privacy in computing and data base systems.

It is virtually certain that the need for access control will increase with time. In many ways, the future of computer systems is the future of large data base systems. Recent developments in computer technology are already causing manifestations of this trend. Larger, faster, and less expensive secondary storage devices give rise to memory hierarchies of physical media. No longer can the storage medium, and its often sensitive contents, be physically carried away as it could in the case of magnetic tape; now storage is on-line. Every computer system is now an on-line data base system. Multi-programming systems are designed to facilitate the sharing of resources. Therefore, the data is sharable. Often, however, it is desirable to restrict certain types of access to data under some conditions, in order to preserve security, privacy, and integrity of that data. Therefore, access control is an absolutely essential requirement of computer systems of the future. Without proper access control, the user will be reluctant to participate in a sharable data base.

In the preceding paragraphs the terms security, privacy, and access control have been used without definition. Universally accepted definitions do not exist, but it is possible to make a distinction among these terms based on usage in the literature. For example, Ware [WAREW67] differentiates between privacy and security on the basis of what is being protected. National defense secrets are subject to security; a personal dossier is subject to privacy or confidentiality. His distinctions appear to be drawn on the grounds of what kind of data is being
protected, the reasons why it is being protected, and the consequences of a failure of that protection—especially as seen from a military, industrial, economic, political, or social viewpoint.

Others make a distinction between privacy and security based wholly on the type of operation performed [CLIEN74]. To these people, security is the prevention of unauthorized destruction or modification of data, and privacy is the control of illegal reading or disclosure of data. Integrity is very close to security except that integrity often connotes prevention of accidental damage to data. In this work access control will be defined as the restriction, regulation, and monitoring of resource usage and in particular of the transmission of data and transformations upon data in a shared resource computer system. As such, access control is the general machinery which can be used to achieve privacy, confidentiality, security, integrity, and protection. Within this work, all of these terms will be used almost interchangeably with "access control" as it is defined above. The term "access control" is often preferred because it describes the actual phenomenon which is taking place without carrying out any implication regarding the purpose or consequences of that phenomenon. It is also a desirable term because "control" implies checking and regulation, not necessarily just an arbitrary approval or denial. The basis of a particular way the access control is accomplished will occasionally be referred to as a "model of protection." A resource secure system is a computer system with access control capability which serves as an instrument for specification and enforcement of the protection policies (security requirements) of the individual users and of the computer system administrator.

Although it is clear that "technical solutions to the problems of computer abuse are necessary, but not sufficient" [PARKD73], only the technical and not the social, legal, or economic issues are treated here.

The next section will enlarge this perspective by briefly describing the setting in which the development of this dissertation takes place.

**The Setting**

Within this section the purview of the dissertation will be delimited. Some related topics will be briefly mentioned, to indicate how this work fits into the larger
INTRODUCTION AND BACKGROUND

THE SETTING

context of the data security milieu. However, this section is neither a history of the state-of-the-art nor a comprehensive survey of the current field. For a more complete treatment the reader is referred to the literature for several well-written surveys and bibliographies [BEARC72, WEISH74, BROWP72, HOFFL69, BERGJ72, HARRA67].

Computer security has many faces. The administrative face of security concerns itself with locks, keys, guards, badges, doors, fire protection, and physical access of personnel into computer areas. Within the computer, protection may be classified into physical protection (generally a control of memory, the containers of data), logical protection (protection of the contents), and information protection (protection of the meaning of or knowledge of the contents). The sphere of interest dealt with here is almost exclusively limited to logical protection. Since the logical domain is one step abstracted from the physical domain and deals in "name spaces," the work is one step abstracted from the physical environment and the potential for more general results is afforded. A model of protection (or its implementation) will probably interface with an operating system, very roughly speaking, at the input/output handling and resource allocation level. The data management sub-system of an operating system may provide the interface between the logical space and the machine's physical space.

The class called physical protection also includes such concerns as wire-tapping, electromagnetic emission surveillance, between-the-lines entries on channels, and "piggy back" entry (an illegal computer connected between the legitimate user and his own computer) [BEARC72, PETEH67]. Because every logical scheme must have foundation in physical mechanisms, hardware and firmware (microprogrammed) approaches to support designs at the logical level are an important part of the current work in the field of access control.

The subject of information protection has overt application with regard to statistical data bases [HOFFL73b]. These obvious cases have generated some limited solutions [HANSM71, ASTIA70] for preserving the privacy of individuals in data bases used for social science research, surveys, and other statistical studies. However, there are more profound implications of information protection [BAUMR75, SAVID67] involving inference of semantics. Many of the related problems, all of which are outside of the methods discussed in this dissertation, are very difficult and largely unsolved. The reader is referred to [MINSW68]
for more about the general problem of semantic information processing.

This section is concluded with a discussion of a few lateral issues which could apply to any of the classes of protection. First, the question of identification of users will be treated as a completely separate issue, part of a different field of interest. User identification is a keystone to every access control system. If an interloper can successfully masquerade as an authorized user, for example, by forging an identification or stealing a password, the rest of the protection system is surely obviated. No control mechanism will daunt him from executing operations which are actually unauthorized, but which it believes are authorized. It shall be assumed that the identification function has been thoroughly resolved.

In actuality, the identification process, however, has not been developed to a satisfactory degree of reliability [BEARC72]. One class of identification techniques requires the user to carry a key, badge, magnetic stripe card [MAGNE73], or punched card to insert into the terminal. Another type of recognition is based on memorized passwords or user identification numbers. In addition to the concept of a simple, fixed group of characters, a password may be specified as a procedure. For each entry to the system the user employs the procedure (e.g., some function of the current date plus every other character of his last name) to compute a password.

In extremely sensitive situations, verification of identification can be made by a call-back feature such as that found in Allen Babcock's RUSH [BABCJ67] system. Before usage of the sensitive data the computer (or the operator) can call the authorized user on a separate number. Such a procedure could be very bothersome, and the level of added safety is questionable.

The most promising (with regard to accuracy and reliability) class of identification means of the future depends on a measurement and recognition of patterns of personal physical characteristics. Methods in this class are also more likely to be expensive, because they involve more complex processes, often forms of pattern recognition. Fingerprint identification is an example [ELECM73]. The approach developed by KMS Security Systems of Roseville, Michigan [MURPJJ73b], and a few others, uses holographic correlation. The Fingerscan [FRENN74] by Colspan Corporation uses automatic fingerprint reading technology developed for the F.B.I. The Identimat 2000 of Identimation
INTRODUCTION AND BACKGROUND

THE SETTING

Marketing Company of Northvale, New Jersey, uses measurements of the human hand [SECUR73, MURPJ73b] for identification. Another example is voiceprint recognition as offered by Threshold Technology, Inc. of Cinnaminson, New Jersey, [MURPJ73a, MURPJ73b] and Voiceprint Laboratories, Inc. of Somerville, New Jersey.

As well as solutions to the identification problem, it also will be assumed that there is no "back door" or "sneak entrance" through which any user can surreptitiously by-pass the authorization and enforcement processes. For example, if a system is claimed to have only one point at which user access to resources may occur, but someone finds a way around it by finding a way to get into some supervisor mode, then an interface to a resource security system at that point would not guarantee system protection. A security system must be assumed to be connected at a point where all attempts to use protected resources are subject to its scrutiny.

Another lateral topic, cryptography [BEARC72, VANTD69, SKATR69, FEISH73, BARAP64, CARRJ70, TURNR72] is not of interest in this work. Except for its use as a technique for secure communication of sensitive data, coded scrambling can be an inefficient way to achieve protection. High overhead is often needed to code and decode for transmission of large quantities of data. Decoding and recoding is usually required, even for simple procedures such as sorting and merging of files.

Lastly, the economic considerations of cost and efficiency are not directly part of this work. However, such considerations were carefully taken into account throughout the development, and many decisions made during the modelling phase were motivated by these factors.

The Need for Protection Languages at Many Levels of Sophistication

Due to the sheer size and great complexity of the database on the one hand and the heterogeneous usage patterns and elaborate protection requirements on the other hand, there is considerable difficulty in developing effective interfaces between the users/authorizers of the data base and the data base systems. Nevertheless, without effective
**INTRODUCTION AND BACKGROUND**

**THE NEED FOR PROTECTION LANGUAGES**

User-data-base-system interfaces and authorizer-data-base-system interfaces the capability of advanced data base systems will never be fully utilized. This research on language and language translators for security is aimed to develop such effective interface.

One very important function of these interfaces is the establishment of data security policies and the specification of protection requirements for all or part of the data base. The means for communicating these policies and requirements to the system is a protection specification language, which can be a complete language on its own or the security-oriented part of a more extensive data base management language.

The complexity of the problem of designing a language for protection is reduced by proposing the approach of developing a family of languages, each member of which is aimed for a level of sophistication of the protection system users. The user's level of sophistication is determined by the complexity of protection specifications that he needs as an authorizer, as well as by his skill and experience. The work reported here is the basis for such an approach to protection specification languages. For example, the novice user most likely needs a language for protecting his own files in a very simple way. Thus, his language may be non-procedural and interpretive with simple constructs and relatively free syntax. The system administrator may be responsible for deciding the distribution of ownership and sharing of authorization responsibility, for determining the conditions for keeping access history, and for specifying the procedures for user authentication, alarm and recovery, threat monitoring, journaling, and audit trails. His language is therefore likely to be elaborate and will include procedurally oriented elements with rigid syntax.

One of the requirements in developing a family of protection languages for different levels of sophistication is that the underlying operations resulting from the sequences written in and interactions composed of these language statements must be invariant. In other words, whether the statements are from one level of sophistication or from another level, the language processor for that level will always translate them into a set of primitive security operations which the protection system understands explicitly and unambiguously.

The purpose of the next three chapters is to present a model of primitive data security operations and rules into which all the levels of language constructs can be
translated. Fundamental to the model is the concept of security space within which dynamic access decisions based on conditions (that characterize the system states) are made. Because protection language constructs usually will utilize groupings of definitions, entities, and operation types, the elements of the model are intended to deal with aggregations of system objects.

The need for protection languages has only recently been apparent. It is reasonable that most efforts in this field were concentrated on the enormously complex problems associated with the development of large-scale data bases and prototype protection systems. Therefore, it is logical for recognition of a need for protection languages to succeed those developments. But some of the weaknesses of the early protection systems are directly related to overlooking the implications of the complete spectrum of protection language constructs. This can be found in, for example, [SALTJ74]:

The second serious weakness in the current MULTICS design is the complexity of the user interface. . . . A variety of defaults have been devised to reduce the number of explicit choices which need be made in common cases. . . . The defaults merely hide the complex underlying structure, however, and are not helpful to the user with an unusual protection requirement, who must figure out for himself how to accomplish his intentions amid a myriad of possibilities, not all of which he understands. . . .

The solution to this problem lies in better understanding the nature of a typical user's mental description of protection intent, and then devising interfaces which permit more direct specification of that intent.

In spite of this emerging need, work in the area of languages for specifying data base protection requirements has been very limited. In only a few cases protection related expressions have been integrated into existing programming languages [MORRJ73, MILLJ73, SUMMR74, FERNE74]. These expressions deal with very rudimentary protection considerations. An example of elementary considerations of protection in a data base definition language can be found in [CODAS71]. Furthermore, there has been virtually no published work on languages specifically for protection specifications in data base systems having access control capabilities. The primary purpose of this present work is
INTRODUCTION AND BACKGROUND
THE NEED FOR PROTECTION LANGUAGES

to point out this need and to provide a foundation for future developments in this area.

A second motivation of this work is the belief that an emphasis on a spectrum of languages for specifying protection needs will reflect a broad variety of data security requirements and thereby bring into focus the shortcomings of access control capabilities of existing systems. There are, in fact, significant shortcomings with existing approaches to access control in computer-based data systems. Many current systems of protection are limited by not allowing a broad enough range of dependencies for making access decisions. At the other end of the scale access decisions can be made through very flexible procedures [HOFFFL71] which are difficult to define, control, and authenticate. These procedures, which often must have their own very powerful access privileges, are capable of harming the data base themselves. An approach lying in neither extreme is required. The approach must allow great flexibility by including rules of access as an explicit and manipulable part of the system, but must do so in a highly structured and precisely controlled manner.

The conceptual basis for common operating and data base systems with protection capability is the access matrix [LAMPB71, GRAH72] in which rows correspond to users and columns to data and other protected resources. Elements of the matrix are access rights of users with respect to the resources. Capability lists [DENNJ66, LAMPB69, GRAH71, FABBR74] are access matrices with emphasis on the rows. The authority item [HSIAD68] is predominantly a type of capability list. Access lists [DALER65, ORGAE72] are access matrices with emphasis on the columns. Access lists have the advantage of being directly related to resources and able to control access with respect to a given resource. This is useful, for example, for temporarily blocking all access to a prescribed resource. However, there are serious disadvantages. For example, once a user is defined to MULTICS (a system employing access lists), he must remain so defined on a permanent basis. Because there is no way of finding all the places in which a user's name occurs in access lists of data objects and in programs which initialize those access lists, it is exceedingly difficult to remove all references to a user's name. The approach taken here has a strong capability list flavor, but emphasizes the advantages of each approach. An example of an access control matrix is seen in figure 1.1.
INTRODUCTION AND BACKGROUND
THE NEED FOR PROTECTION LANGUAGES

<table>
<thead>
<tr>
<th>U (The set of Users)</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>READ ONLY</td>
<td>READ ONLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U2</td>
<td>READ, WRITE</td>
<td>READ, WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U3</td>
<td>READ, WRITE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U4</td>
<td></td>
<td></td>
<td>EXEC ONLY</td>
<td></td>
</tr>
</tbody>
</table>

Figure I.1 – Portion of an Access Matrix
As in figure I.1, access rights are represented as a fixed value of either "allow" or "deny" for a given user with respect to a particular resource and a given operation. More specifically, this type of representation is two-valued function (allow or deny) of three variables (user, resource, and access operation). In such systems, an access decision is solely dependent on these three variables, resulting in a very limited range of dependency. Yet, many protection requirements call for access decisions to depend on quite a number of other things [HOFFL69, FRIET70, LAMPB71, CONWR72, BRAND73], such as the ones illustrated in the following examples.

Example 1. (Access Decision Is Based on the Access History of Other Resources.) User A may write in file F only if he has not read from file G.

Example 2. (Access Decision Is Based on the Dynamic State of the System.) User B may open file H only at a time when the data base in which the file resides is in the LOCK state.

Example 3. (Access Decision Is Based on the Prescribed Usage of the Resource.) When a given user calls a SORT routine to sort a particular file, its rights to read data on his behalf are greater than the user's rights, provided the SORT routine does not return the data to the user.

Example 4. (Access Decision Is Based on the Current Value of the Resource.) A given user may not read the salary field of any personnel record for which the salary value is greater than 15,000.

Example 5. (Access Decision Is Based on the Value of Certain Internal System Variables.) No access may be made by anyone in a particular group of users except between 7:00 a.m. and 7:00 p.m. unless via special terminal 72.

Most existing models and systems, with the exception of one early work [OMENR71], have the disadvantage that access decisions are made a priori, on the basis of an external set of access rules, and stored as "yes" or "no" in the access matrix. In the work reported here a model is offered which internalizes the rules of access as an essential part of the model. In the model to be proposed elements of the access matrix are replaced with these rules which will be evaluated.
INTRODUCTION AND BACKGROUND
THE NEED FOR PROTECTION LANGUAGES

dynamically at access time. These rules are embodied as conditions, which are descriptions of system state subsets.

The Approach to Presentation

The modelling and development herein is intentionally general and intended not to be dependent on current hardware configurations, nor to be tied to present day main or secondary memory technology, nor to be unduly influenced by existing operating system or data base system designs, nor to be restricted to contemporary styles of data and file structures. Also, the model is in no way dependent on a secret. Rather, it is based on the principle that security must prevail even when every user fully understands how the security system works.

The emphasis in presentation is on the intuitive, the conceptual, and the descriptive. Occasional small doses of mildly set-theoretic formalism are included (mostly in the appendices) for added precision. Thus, the reader may expect a sequence of motivations, definitions, and demonstrations rather than theorems and proofs.
CHAPTER II

CONCEPTS AND TERMINOLOGY

Words are not crystal, transparent, and unchanged; they are the skin of living thoughts, and may vary greatly in color and content according to the circumstances and time in which they are used.
—Oliver Wendell Holmes

The way to be safe is to never feel secure.
—H. G. Bohn, Handbook of Proverbs

Introduction

Chapters II, III, and IV are an intuitive presentation of an access control model. These chapters contain the most significant content of the dissertation. In a dissertation professed to be about languages one may wonder why a model is required or, at least, why a model plays such a dominant role. The reason is that languages by themselves have little meaning. The model is constructed to supply a complete context for discussion of the languages, to support and give substance to the languages. The paradigm presented here will be used as a semantic foundation for a family of protection languages in much the same sense that Dennis' common base language model provides a base for the fundamental constructs of programming languages [DENNJ72]. In order to fulfill this role the model must characterize all aspects of logical protection as completely as possible. In other words, a model of semantics for protection
languages must be comprehensive. The study of all the processes of protection is believed to be the best way to provide for a complete consideration of the languages and for the broadest possible choice of constructs in their design. Thus, the reader will find the model to play a strong role, though the dissertation is fundamentally about protection languages.

As a semantic model of protection languages, the model herein is descriptive of what constructs, mechanisms, and processes are required in a semantic base for protection languages, but not how these may be realized in a given system environment. That is to say that this is a conceptual and not an implementational model. The information contained in the tuples of the model may be stored in many possible ways in a given implementation.

The diagram in figure II.1 will serve to introduce, without prior definition, most of the important terms which the rest of this chapter will undertake to define and explain. This general view will help to illustrate the role of protection languages and their application in access control. The left side is generally the domain of the authorizer, since it is used to make authorizations. This side is termed the syntactic side, since it is concerned with designing and using languages to express policies to the system. Here, authorizers (1) encode (4) their access control policies (2) using protection languages (3) at various levels of sophistication, producing authorization specifications (5). Each authorization is translated by the authorization process (6) into some internal representation of the system's access control information (7). Before storing it as such the authorization process (6) performs a validation check to make sure the authorizer owns the resources affected by his authorizations. As this check is really the enforcement of "ownership" as a data operation, the enforcement process (13) is used in a "feedback loop" to make this check. The internal representation may have one of the many forms mentioned in chapter I—an access matrix, a capability list, authority items, an access control list, or some modified form of access matrix. The internal representation provides the connection and communication between the two domains.
Figure II.1 - Overview of Access Control Processes
The right side of figure II.1 is called the semantic side, since it deals with the meaning of the authorizations and how they affect access requests. This side represents the domain of the user, because it contains the mechanisms by which protection is enforced on user requests. Here, a user (8) encodes (11) his access needs (9) via query languages (10), producing access requests (12). The term query language is broadly used here to mean any formal way to communicate the access needs to the system. The enforcement process (13) must use information from the internal representation (7), the request itself (12) and the state of the system (14) to make access decisions (15) whether to permit or deny the logical access (16). In general, one or more accesses by the system itself may be required to obtain the necessary system state information. The logical access is the completion of the act of passing the data between the data base and the user.

Separation of Policies from Mechanisms

From the beginning, programmable computing devices have been based on the principle of separation between policies and mechanisms. Policies are externally defined, broad tasks to be accomplished by users of the computing system. For example, the algorithm of a program is a precise, formal statement of the user's policy. On the other hand, a mechanism is a set of primitive functions with which implementations of policies can be built. For example, the individual instructions in a computer instruction set may be so considered. While no single instruction can do any of the broad tasks of the user, the set of instructions should not impose inherent limits on the kinds of tasks (policies) the user can implement. Although this distinction is well understood at the program instruction level, it is often ignored at the system level. A notable exception to this neglect is the work on HYDRA, described in [JONEA73, WULFW74]. Application of this principle in the model of protection proposed here lends much of the motivation for its broad generality, leaving to higher level policy many decisions heretofore built into protection mechanisms. Protection language constructs (called semantic parameters) will be introduced (in chapter V) to allow the specification of these high level protection policies.
Ownership

A data base system which can make access decisions must analyze each access request with respect to some information about access authority. These authorizations must be stored in the system and must come from certain individuals who have the power to give others permission to access parts of the data. The power to grant access authority to a specific part of the data base is related to the ownership of that part of the data base. The important concept of ownership will be treated in a way so as to give it a more general meaning than it derives from traditional definitions [HSIAD68, BARRD67, DENNJ66]. What does it mean to say that a user is the owner of a particular resource or data unit? It usually means that he created the data or he entered the data into the system. The important implication is that he has the legal authority to grant access privileges (with respect to that data) to other users. He also has the authority to share his ownership with other users. In the model presented here, this ownership authority is not necessarily all given to a data base administrator nor is it necessarily a permanent privilege of any individual in the system. Also, ownership is not always restricted to individuals. Since the model deals with sets of users, several users can share the authorization responsibility for a resource under group ownership. Representation of ownership is part of the more general notion of representation of access control information. The following sections will further explain some of these notions.

System Administrator

In the beginning, there are no users and no user data in the system. Yet there are still (system) resources. The system administrator (SYSADMIN) is usually declared to be the sole owner of these system resources. In most systems the SYSADMIN has certain administrative responsibilities requiring access privileges beyond those of the average user. However, often the SYSADMIN does not own everything in the data base. In fact, he may not even have access rights to some data, such as privately owned files. Such limitations may be important to protect individual users from possible misdoings by the SYSADMIN. The extent of the SYSADMIN's power is a question of policy. Here the mechanisms are provided to represent any choice in this
regard. (See the section on the validation process in chapter IV.)

Subownership

A hierarchy has been introduced [HSIAD69] in which the "system administrator" is the owner of all the files. Under the administrator are the owners who may grant co-ownership. Under these are the non-owning users. It is believed that a hierarchy is not always suitable, and it is proposed to generalize somewhat on these concepts. The generalization is based, to an extent, on this guideline:

Any right which is granted can also be taken away, but the only agent who may remove access rights is the original grantor of those rights.

The concept of co-ownership is not broad enough to allow for this guideline. For example, an owner may share his ownership causing the grantee to be a co-owner. The new co-owner can confer more co-ownerships and can retract them by removing ownership rights. But that new co-owner might remove some of the original owner's rights, especially his ownership rights. This should not be allowed; however, the original owner should be able to remove the new co-owner's rights. Therefore, they cannot both have exactly the same authority. Thus, it is proposed to introduce a new concept: subownership. An owner can create subowners. Subowners can affect other subowner rights, but not owner rights (all with respect to the same resource, of course). The original owner, then, always has master control and is the center of responsibility for his owned resources. Each subowner is on some lower authority level.

Conditional Ownership

Both ownership and subownership may be determined dynamically. Under the conditions which determine a user to be an owner of a resource, he may grant authority with respect to that resource. He is considered to be the owner in that situation. Under other conditions, he may not.
Such ownership is termed **conditional ownership**.

**Sets of the Model**

Several sets are involved in the conceptual model of access control for language constructs. The model is based on sets of n-tuples (n-ary relations) of these sets. These basic sets are introduced in this section.

**Users**

An important set of the model is the set of individual users or agents who will be making access requests. These users are users of the data base system and do not necessarily communicate directly with the access control system. For convenience, sets of users will be called **user groups**.

**Authorizers**

There is also a set of individuals who will be granting authorizations, the issuers or authorizers. The set of authorizers is generally a subset of the users. An authorizer is a user of the access control system. In some way he is the owner of resources and has the responsibility of making authorizations, determining who may share in the access of those resources and in what manner. In the literature the term "user" is almost universally used to denote both users of the data base system and users of the protection system. The distinction is emphasized here by using the term "authorizer" to denote a user of the access control system because of its importance in the authorization process.
Resources

The collection of data elements forms the set of resources or objects. A data element is the smallest entity which can be addressed or named on an individual basis. A data element may be a record, a field, a byte, or even a bit. Physical devices such as tape drives, channels, disks, and even CPU registers are viewed here as data elements, since they may be named. Generally, these physical resources are represented to the rest of the system by the values found in various associated status indicators, control blocks, directories, channel programs, and buffers. For convenience, sets of resources will be referred to as resource units.

Elemental Data Operations

The various operations a user may perform on the data elements also comprise a set. Examples of operations one might find in a given implementation are: OPEN, CREATE, READ, WRITE, ADD, UPDATE, DELETE, EXECUTE, COMPARE, DISPLAY, OWN, and SUBOWN.

Notation

Symbolically, there are the following sets:

\[ U \] - the set of users.

\[ A \] - the set of authorizers, including a distinguished member SYSADMIN. It is noted that \( A \subseteq U \).

\[ B \] - the set of resources, including all data elements in the data base.

\[ E \] - the set of elemental operations, usually with a small, fixed membership.
CONCEPTS AND TERMINOLOGY
SETS OF THE MODEL

It should be understood that wherever the sets are used in any mathematical manipulations, they are treated as sets of tokens, representing the names of users, data elements, and operations. Further, it is stipulated that the names of the elements of these sets are unique. Unless otherwise noted, there will be no effort to distinguish the elements from their names. For instance, u of U may be referred to either as a user or as a user name.

The following notation will be used throughout this report. Underscored upper case letters will denote sets while lower case letters, possibly subscripted, will denote set elements. Finally, subsets will be denoted by upper case letters, possible subscripted. For example, s, s[1], or s1 is an element of S, while S = {s[1],s[2]}={s1,s2} is a subset of S = {s[1],s[2],s[3]} = {s1,s2,s3}.

Structuring of the Sets

Various structures may be imposed within these sets, if desired. For example, experience may incline one to impose hierarchical structures on the users, the data elements, or the data operations. Those structures certainly can be accommodated, but they will not be assumed, and nothing in this work will depend on such assumptions.

Several major existing protection systems presented in the literature have built-in structuring that, in some way, limits their applicability to the general problem of protection. For example, access decisions in the ADEPT-50 system [NEISC69] are based on comparisons of fixed, ordered levels of clearance of users against fixed, ordered levels of classification of resources. Access is permitted when the clearance is greater than or equal to the classification.

Similarly, IBM’s Resource Security System [HOFFL73a] allows a user clearance to data whenever his security level is greater than or equal to that of the data. RSS is based on a rather inflexible scheme of eight hierarchical levels. The protection rings of MULTICS is another example of a hierarchical structure built into an access control system. The view is taken here that these structures represent policies, not fundamental mechanisms. As such, they should not be built into a basic model of protection nor, many times, immutably fixed in a given implementation. A basic
The ability to group users, data elements, and states into subsets of $U$, $R$, and $S$, respectively, is of such importance in real systems that methods for dealing with these groupings must be taken into account. Furthermore, these subsets must not be required to be disjoint. That is, user groups must be allowed to overlap other user groups. There must be similar flexibility for the resource units and the access conditions. Grouping offers protection at the data element level and at all subset levels. Many applications will have data bases with very large numbers of data elements, perhaps in the millions. Often, in conventional systems, the data elements cannot be grouped together to any extent because the groupings would have to be different for each type of protection. Subset protection allows the degree of fineness to be variable from the individual data element all the way up to the entire data base.

System States and Conditions

Resource Values

In dealing with the set of resources the content, or values, of the resources are of particular interest. The finite resources in $R$ may be enumerated as follows: $r_1, r_2, \ldots, r[n]$. Let $Y: R \rightarrow \Sigma^*$ be a set function, where $\Sigma$ is the alphabet used to describe data values (real numbers, integers, character strings, hexadecimal, etc.). Then $Y(r[i])$ denotes the set of all possible values for resource $r[i]$. To characterize all possible contents of the data base system (all possible combinations of resource values) a cartesian product of the values in $R$ is formed. More precisely,
CONCEPTS AND TERMINOLOGY
SYSTEM STATES

\[ S = \prod_{i=1}^{n} V(r[i]) \]  

(2.1)

and term the product \( S \), the set of resource value states. For brevity, \( S \) will be called the states of the data base system. There are now five basic sets in the model—namely, \( U, A, R, E, \) and \( S \).

A state of the data base system may be represented by an n-tuple as follows:

\[(v(r1), v(r2), \ldots, v(r[n])),\]

where \( v(r[i]) \in V(r[i]) \), for \( i=1, 2, \ldots, n \). Every data operation, in fact every step of every operation in the central processor or the data base, is a function \( f: S \rightarrow S \), a state changing function causing a new system state.

Restrictions on Resource Values

The use of all possible resource values in (2.1) corresponds to the set of all system states. Restricting resource values to certain ranges of values corresponds to subsets of states. Thus, state subsets are defined by placing restrictions on the values of some of the resources. When the restrictions made on the values of each resource are independent of the values of each of the other resources (see the example which follows), such a subset is obtained when specific subsets \( V(r[i]) \) are used in (2.1) giving

\[ S = \prod_{i=1}^{n} V(r[i]) \]

Here, at least one \( V(r[i]) \) is a proper subset of its corresponding \( V(r[i]) \) such that \( S \) is a proper subset of \( S \).

Consider the following simple example involving only two resources, \( r1 \) and \( r2 \). (The explanation of typography and notation at the beginning of the dissertation may be a useful reference for reading the examples in this section.) The sets of values that these resources can have are listed herewith:
CONCEPTS AND TERMINOLOGY

SYSTEM STATES

\[
V(r1) = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) \\
V(r2) = \{a, b, c\}
\]

The subset of states is described by the restriction \( v(r1) < 5 \) as follows

\[
VI(r1) = (v(r1) : v(r1) < 5)
\]

or is specified directly as:

\[
VI(r1) = \{1, 2, 3, 4\}
\]

Thus, the state subset is:

\[
S = \text{VI}(r1) \times V(r2) = \{(1,a), (1,b), (1,c), (2,a), (2,b), (2,c), (3,a), (3,b), (3,c), (4,a), (4,b), (4,c)\}
\]

However, when the restrictions defining the subsets of resource values have dependencies among resources, it is not possible to find a cartesian product of subsets which will yield the required subset. To illustrate, consider the resources of the following example:

\[
V(r1) = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) \\
V(r2) = \{1, 2, 3, 4, 5\}
\]

Suppose the restriction is: \( v(r1) + v(r2) < 5 \). Here, \( v(r1) \) and \( v(r2) \) are not independent. There can be no choice for \( V(r1) \) and \( V(r2) \) which will allow their cartesian product to be the required state subset:

\[
SI = \{(1,1), (1,2), (1,3), (2,1), (2,2), (3,1)\}
\]

Conditions

Although a representation for \( SI \) does exist in set notation (see appendix A), a more practical method for specifying restrictions is desirable. In other words, a very general approach to divide up the states in many different ways is needed. A predicate will partition the set of all states into two equivalence classes: those for which the predicate is true and those for which it is false. A uniform way to express one of these predicates is as
The contents of the square brackets in (2.2) is a comparison expression. The symbol \( a \) designates an arithmetic expression of resource values and \( p \) is a relational operator in \( \{=,\neq,\lt,\gt,\leq,\geq\} \). The interpretation of \( p \) and the operations within \( a \) may be defined by the user so that values other than purely numeric ones may be used (e.g., character strings, sets and relations, trees, lists). Herein is adopted the convention from programming languages that, in an expression, a resource name, \( r[i] \), actually refers to its value, \( v(r[i]) \). Some examples of comparison expressions are as follows:

\[
\begin{align*}
\text{ALPHA} & \leq 7.56 \\
X & = Y \\
R1 + 3752 & > PI \times (R2)^{**2}
\end{align*}
\]

The notation \( \land \) denotes a logical expression of comparison expressions connected by \& (AND), \+ (OR), and \( \neg \) (NOT). By context, the use of "+" for either the logical OR or the arithmetic addition operator will be distinguishable. Thus, an example of a predicate containing the above comparison expressions is:

\[
(\text{ALPHA} \leq 7.56) \land ((X = Y) + (R1 + 3752 > PI \times (R2)^{**2}))
\]

Because each predicate states a condition on which a grouping of the states is based, they will be referred to as conditions. A more precise definition of a condition, within the context of protection languages, is given in appendix C.

Thus, a condition is a predicate (or proposition) on states whose resource values are restricted. The evaluation of a condition at a given time is accomplished by substituting the current values, \( v(r[i]) \), of the resource \( r[i] \), for each corresponding resource name in the predicate and determining whether the predicate holds or not.

Conditions are used for testing set membership. Let \( C[i] \) be a condition associated with a set \( B \); it will be said that:

\[
\begin{align*}
b & \in B, \text{ if } C[i] \text{ holds} \\
b & \notin B, \text{ otherwise}
\end{align*}
\]
Obviously, the membership of $B$ is dependent on the condition, $C(i)$. In such a case, the condition will be denoted as $C(b,B)$. Thus,

$$B = \{ b : C(b,B) \text{ holds } \} \quad (2.3)$$

It will be said that $b$ is an unconditional (non-)member of $B$ if $C(b,B)$ always (never) holds. A condition which always (never) holds is simply a constant value of "true" ("false"). To determine if a candidate $b$ is an element of $B$, the expression for $C(b,B)$—an expression of the form $(2.2)$—is evaluated and then the above definition $(2.3)$ is applied. If the expression holds, then it will be said that the element $b$ satisfies the condition, $C(b,B)$. Furthermore, an elementary result of set theory states that, for sets $B_1$ and $B_2$,

$$C(b, B_1 \cup B_2) = C(b, B_1) + C(b, B_2)$$
$$C(b, B_1 \cap B_2) = C(b, B_1) \& C(b, B_2)$$
$$C(b, -B_1) = \neg C(b, B_1)$$

If the set in question is $U$, a group of users ($U \subseteq U$), then $C(u,U)$ determines whether or not an individual user $u$ has membership in user group $U$ at a specific instant in time and $C(u,U)$ is called a user group defining condition. The same applies to a resource $r$ and a resource unit $R$ ($R \subseteq R$), using condition $C(r,R)$, which is called a resource defining condition. The most important use of a condition is as an access condition, $C(s,S)$, which defines the set of states for which a given type of access is allowed. It will be seen that conditions are also used to control history keeping, as $C(s,Slh)$, and to control invocation of certain auxiliary procedures, as $C(s,Slf)$. This notation—using $C(u,U)$, $C(r,R)$, and $C(s,S)$—is not quite general mathematically. However, the more general notation, presented in appendix B, is less convenient to use throughout the rest of the dissertation. The main thrust of the more general approach is that a given $C(u,U)$ is an expression which, when evaluated, tells whether $u \in U$ or not. Since the expression is of the form of $(2.2)$, it is a logical expression of resource values. Therefore, $C(u,U)$ also describes a subset of states, namely those states in which $u \in U$. Thus, all sets $U$ (and $R$) have counterpart sets in $S$. That is, every condition defines both a set of states (since it is an expression of resource values) and some other set like $U$ or $R$. Therefore, the set of states of a given access condition must have as its counterpart another set in the system. Such a set exists and is discussed in
appendix B, but it is not useful for the development of this model.

The reader should see appendix E for examples of conditions; some of these examples will be referred to in the next chapter.

Introduction to Authorization and Enforcement

The statement giving access privileges, made by the owner, will be called an authorization specification. In its most general form it must say who is receiving access authority, for what parts of the data base, for what operations on that data, and when (or under what conditions) that access may take place. If a later access decision is independent of whether or not an access of the type being presently authorized has occurred, then such an occurrence need not be remembered by the system. Authorization specifications of this type are termed basic (memoryless) specifications. Specifications like these merely permit an authorizer to tie together previously established definitions of user group, resource units, and access conditions (discussed in the preceding sections).

On the other hand, if any later access is to be allowed or denied on the basis of whether or not an access of the type now being authorized has occurred, then such an occurrence must be remembered by the system. In this case, the system must be told that the specific access (for which the present authorization is being made) is to affect later access decisions. Systems which include this capability are emphasized in this report and are termed (access) history keeping systems.

In addition to history keeping, there is the need of a system capability to enter and invoke programs for many security related tasks (such as authentication and alarm and recovery). This capability is called auxiliary program invocation. There is also a need to control the use of (system) programs having greater access rights than those of the user. Such programs are termed extended resources in chapter III. The authorization specifications which can deal with history keeping, auxiliary program invocation, and use of extended resources are termed full authorization specifications.
For a given secure data base the net collection of all authorization specifications accepted by the system up until the present time represents a statement of security policy of the users and administrators as of the present time. Generally, as time advances and more authorization specifications are introduced the policy represented within the system changes.

The purpose of a data base system is to serve the user in response to an access request or query, a request by an individual user for a single data operation on a resource unit.

It seems clear that a secure data base system must have two processes which remain continuously operative: one to accept and understand authorization specifications and to add them to its collection, another to enforce these authorizations when an access request is received. The first is the authorization process and the latter is the enforcement process.

The authorization process first determines the validity of each specification based on whether the authorizer has the proper degree of ownership at the time of the authorization. After that, this process is responsible for adding and withdrawing access rights resulting from the stream of new authorization specifications (detailed in chapter IV). Furthermore, in response to full specifications, it must maintain history keeping conditions, enter and compile auxiliary programs to be invoked by users, and establish the necessary mechanisms for use of extended resources.

On the other hand, the enforcement process must use the output of the authorization process as a basis for access decisions, which must be made for every access request. In addition, if history keeping, auxiliary programs, and monitoring of extended resource usage have been specified, they must be invoked at access time by the enforcement process.

Thus, there are three conceptual streams of data flow in a secure system: the authorizations, the access requests, and the controlled flow of data in and out of the data base. The function of data security is to control the flow of data by making the proper allow/deny decisions for each access request at a given time, based on the set of authorization specifications received prior to that time.
CHAPTER III

THE SECURITY SPACE AS A 5-DIMENSIONAL DISCRETE SPACE

Space and time, and with them all phenomena, are not things by themselves, but representations, and cannot exist outside of the mind.
—Immanuel Kant, The Critique of Pure Reason

The use of history is to give value to the present hour and its duty.
—Emerson, Society and Solitude: Works and Days

Definition of the Space

If a single element were to be chosen from each of the five basic sets of the model and put together in an ordered 5-tuple:

$$(a,u,e,r,s)$$ (3.i)

where $a \in A$, $u \in U$, $e \in E$, $r \in R$ and $s \in S$ it would represent a single point in a discrete 5-space whose coordinate dimensions correspond to the five sets. The cartesian product of all the sets makes up this entire security space, which will be used to provide an abstract view of the semantics of protection language constructs.
Access Requests and Queries

An access request, or query, is a 4-tuple:

\[ q = (u, e, R, s) \]  \hspace{1cm} (3.2)

where \( u \in U \), \( e \in E \), \( R \in R \), and \( s \in S \). This represents a request by an individual user \( u \) for a single operation \( e \) to a resource unit \( R \) at a time when the system is in state \( s \). Of the four elements, the user actually requests only \( e \) and \( R \). Both \( u \) and \( s \) are supplied by the system in an "unforgeable" manner. An access is an access request which has become manifest by actually occurring in the system. An access also has the form of (3.2), the only difference being whether the operation is pending or has already occurred.

As will be seen in the section in chapter IV on modelling the enforcement process, queries for which the requested resource unit is a subset of just one resource unit previously defined by an authorization are called simple queries. If the requested unit overlaps more than one previously specified unit, the query is a compound query. Simple queries require less processing for enforcement. Let it be said that \( q \in Q \), where \( Q \) is the set of all possible access requests, or queries.

Clearly, a query is a subspace of a four dimensional projection of the security space. The enforcement problem requires a determination of whether the request is authorized, i.e., whether the access request subspace is completely contained within the subspace characterized by the authorizations. These are explained in the next section.

Authorization Specifications

DEFINITION: An instance of an authorization specification is a 5-tuple:

\[(a, U, E, R, S)\]

where \( a \in A \), \( U \subseteq U \), \( E \subseteq E \), \( R \subseteq R \), and \( S \subseteq S \). The authorizer \( a \) declares by this 5-tuple that each
and every individual user in group $U$ may do any or all of the operations in $E$ to any or all of the resources in $K$ only if the system is in one of the states in $S$.

(3.3)

The concept of ownership introduces a relation which determines the validity of the specification. The complete nature of this relation is given later in (4.1) and appendix D. In essence, the proper degree of ownership makes authorizers out of users; i.e., one must in some way own a resource in order to authorize its use. Thus, authorization languages must have the capability of explicitly denoting ownership.

The 5-tuple in (3.3) is of a slightly different kind than that in (3.1), because the last four items in (3.3) are not just elements of their corresponding sets, but are subsets. Thus, in general, an authorization corresponds to a security subspace greater than just a point. Typically, there might be overlap in several dimensions between subspaces corresponding to any two specifications; i.e., authorizations are not required to be disjoint along any axis. Examples of security subspaces are given in appendix F. The syntax of language statements expressing authorizations may vary widely, but each different form must translate unambiguously into these 5-tuples.

Generally, authorization specifications are used to give access rights to users and to withdraw existing rights back from users.

In considering the relation between the access conditions and the security space, it is noted that an access condition describes a subset of states or a portion of a state axis. More specifically, for fixed subsets $A$, $U$, $E$, and $R$, the condition describing subset $S$ corresponds to regions in the security space. A condition is a logical expression of terms and each term of the condition also corresponds to some region. The size of the corresponding subspace varies with the nature of the condition. When terms are conjoined in the Boolean expression, their corresponding regions are intersected in the security space resulting in a smaller region. When terms are disjoined, a union of their regions is formed resulting in a larger region.
Examples of Authorization Specifications

In the first example reference is made to the examples of conditions, in appendix E, for the definitions of U1, R2, and S3. These can be brought together in this specification:

\[ p1 = (a7, U1, \text{(WRITE)}, R2, S3) \]

Here the authorizer a7 says that anyone in U1 (Department 13 or in the Design Project) may write into R2 (records for which DEPARTMENT=7 and SALARY < 20000), providing it is a Friday.

In this authorization:

\[ p2 = (\text{SYSADMIN}, \text{GENERAL}, \text{(READ)}, \text{FILE1}, \text{ALL}) \]

the SYSADMIN is giving a basic right for every user (GENERAL) to read from File1 with no conditional restrictions.

The following is the protection requirement upon which the next authorization, according to authorizer a3, is based: Any user may add field A to any record in file F1 having the keyword COLOR=BLUE, providing there is no field B already in that record. This first requires a resource unit definition:

\[ C(r, R4) : \text{FILE=F1} \& \text{COLOR=BLUE} \& \text{FIELD = FIELD A} \]

The access condition must be given as:

\[ C(s, S6) : \text{FIELD B==NULL} \]

The authorization is:

\[ p3 = (a3, \text{GENERAL}, \text{(WRITE)}, R4, S6) \]

This next requirement is taken from [CONWR72]: A user may see and update only "financial" parts of each record in a file, say PERSONNEL, and only between 9 a.m. and 5 p.m. from a specific terminal in the payroll office. Let it also be required that the user be a member of the payroll department. The user group is defined by:
THE SECURITY SPACE
EXAMPLES OF AUTHORIZATION SPECIFICATIONS

\[ C(u,U6) : \text{DEPT}=\text{PAYROLL} \& \text{TERML}=\text{PAYOFFICE} \]

The resource unit is:

\[ C(r,R5) : \text{FILE}=\text{PERSONNEL} \& (\text{FIELD}=\text{SALARY} + \text{FIELD}=\text{RATE}) \]

assuming the salary and rate fields are the "financial" part of the records. The access condition is:

\[ C(s,S7) : \text{TIME} > 9:00 \& \text{TIME} < 17:00 \]

The authorization becomes:

\[ p4 = (a,U6,\{\text{READ,UPDATE}\},R5,S7) \]

In the next example, suppose it is discovered that a group of records recently entered into FILEA appear to have some anomaly which must be cleared up before anyone may safely access these records. Therefore, a temporary authorization is needed to block the use of these records. Each record has a non-keyword field containing an accession number. The group of records is known to be in a certain range of accession numbers. The user group GENERAL is specified, to include all users. Also the access condition must be zero to block all access. The resource unit definition contains non-keyword data dependency:

\[ C(r,R5) : \text{FILE}=\text{FILEA} \& \text{ACCNBR} > 100370 \& \text{ACCNBR} < 10100 \]

Suppose the required authorization is issued by SYSADMIN:

\[ p5 = (\text{SYSADMIN,GENERAL,ALL},R5,\text{NULL}) \]

Since this authorization is only to cause a temporary block until the data can be cleaned up, it might be helpful to inform users of this when they attempt an access. Such a case is used as an example in the section on auxiliary procedure invocation later in this chapter.

The last example has a requirement typical of a shared data base in an information system: The only user who can update a record in the part number file is the person who originally entered that same record. In this case it can be assumed that, at the time of record entry, the user's identification is automatically entered in the "source" field. The best way to express this requirement uses a resource unit of:
THE SECURITY SPACE
EXAMPLES OF AUTHORIZATION SPECIFICATIONS

\[ C(i, R6) \times FILE=PARTNBR \]

which refers to every field of every record in the named file. Next an access decision is needed:

\[ C(s, S8) \times SOURCEFIELD=USERID \]

where the value for \( SOURCEFIELD \) is obtained from the record for which an update is requested. \( USERID \) is probably taken from a user control block which was filled in at login time. This authorization:

\[ p6 = (SYSADMIN, GENERAL, (UPDATE), R6, S8) \]

will control the update requests. But another specification is needed to prevent general users from getting around the requirement by changing the resource field, defined by:

\[ C(r, R7) \times FILE=PARTNBR \& FIELD=SOURCEFIELD \]

This is the extra authorization needed:

\[ p7 = (SYSADMIN, GENERAL, (WRITE), R7, NULL) \]

This authorization prevents any user from writing in the source field regardless of any other authorizations in the system.

The reader is referred to appendix F for a discussion of the role of the basic authorization as depicted by projections of the security space.

\[ \text{The Role of Full Specifications} \]

Access History Keeping

In chapter II the history keeping function was mentioned. Since some access rules may depend on whether or not certain user operations have been executed under given conditions, past access information is necessary. History keeping means the storing of information in response to events within the system. Language specifications of history keeping requirements can be somewhat complex. For
example, an authorizer may want to keep track of only actual accesses (those that successfully occur) of a certain type. In other cases the authorizer, desiring to have a specification sensitive to an attempted security violation, may want to record the occurrence of an unsuccessful access request. Furthermore, each authorizer may issue language commands to reset his history keeping counters. The definition of "event" is more versatile if it includes a denied access as well as an allowed access. Thus, the term event will be associated with the access decision in the enforcement process rather than with the physical event. Therefore, an event is a logical occurrence. It can, of course, be assumed that, if the decision is to allow access, the physical occurrence will follow and, if it is to deny access, the physical occurrence will not take place. Furthermore, maximum ability to distinguish between events requires that those events be associated with the individual authorization specifications used in making the access decisions. In order to refer to these concepts more conveniently an access decision will now be defined more precisely.

**DEFN:** The access decision for request \( q \) is a function \( d(q) \) with the following meaning,

\[
d(q) = \begin{cases} 1, & \text{if the enforcement process allows request } q \\ 0, & \text{otherwise} \end{cases}
\]  

(3.4)

Here, the term "allows" means that the data base system will proceed to perform the requested operations on the requested resources on behalf of the requesting user. As is the case with conditions, the access decision can be used in expressions directly as a predicate having values true (1) and false (0).

**DEFN:** A p-event is the use of the authorization \( p \) in an access decision, \( d(q) \).

(3.5)

For convenience, the term event will be used to mean p-event. Since an access decision can involve several authorizations (shown in the section on modelling enforcement in chapter IV), every access decision can involve several events. Note that no access need occur to make an event, only an access decision (which may result in a denial of access). There are three types of events:
1) Independent event—regardless of the outcome of the access decision

2) Success event—the access decision is to allow, \( d(q) = 1 \)

3) Failure event—the access decision is to deny, \( d(q) = 0 \)

Models for including the history keeping function may vary, especially in implementation details. However, it is possible to identify approaches differing in the time when the decision is made. In the access decision keeping approach, the occurrence of an access event causes an access decision to be made at the time of that event and the decision to be stored for later use. In contrast, the access event keeping approach requires only the recording of the event, not any future effect. How this recorded information is used in making later access decisions is not affected by the way in which it is recorded.

Consider, for example, the following requirement appearing in the statement of a protection language: ul may write RL if he has not already read R2. In the event that ul does read R2 the access decision keeping approach would cause the following effect to be recorded: "If ul requests to write RL, block him because that is now an illegal access." The event keeping approach, on the other hand, would merely store the fact that ul did read R2 without any regard for its relation to a possible later request to write RL. Event keeping allows the access rules to be changed after the event is recorded. Decision keeping does not. Therefore, only with event keeping can the policy be determined later, say at a future access time.

There are three issues bearing on the general nature of the approaches discussed. The first issue is the matter of dependency on other resource values occurring at the time the event is observed, to be remembered with the event. In case of such dependence, it is necessary to remember parts of the "historical" context in terms of system conditions prevailing at the time that the event took place. In other words, the decision of whether or not to keep the history of a given access must be made to depend on an arbitrary condition which may (or may not) hold at the time of the access. Later, it will then be known, not only that the access has taken place, but that a certain condition (values of resources) prevailed at the time of the access. This history keeping condition provides a context to the access event. For example, a protection language may express this rather sophisticated requirement: "If the system is now in
the Top Security State, Ul can read Rl only if a member of U2 did not write R2 in a certain update mode." In this model, a full range of dependency is allowed at both event and future access times. In this example, the decision to record the event of U2 writing R2, whenever that event may occur, is dependent upon the condition that the system be in a certain update mode at the time of the event. The future access of Ul to read Rl then depends on this event's not having been recorded and the system being in the Top Security State.

The second issue may be stated as a general rule of thumb: a history keeping mechanism should be discriminating about what it remembers. It is not desirable to keep track of every operation that every user performs on every resource. The number of such operations is enormous even for a small system. Instead, only event information which will be needed later for making an access decision should be recorded. Thus, there needs to be a way, within protection languages of high sophistication levels, that authorizers can exactly specify their history taking needs.

The third issue is the problem of access history dominance [COHEE74]. Dominance occurs in either of these situations: once an access is denied, that access is always denied, or once an access is allowed, it is always allowed. Neither of these situations allows for dynamic adjustment of access decisions. In the first case a denial dominates; in the second, approval dominates. In other words, in a system with access history dominance a decision once made for an access request will be forever remembered (or kept) by the system. There is no way to make a new decision for the same access request at some later time. Dominance is an inherent problem in the access decision keeping approach, but does not affect the event keeping approach. As an example of a situation illustrating this problem, consider this protection language statement: a member of Ul (say ul) cannot write Rl after reading R2 unless he subsequently opens R3. After ul reads R2, the decision keeping process records "ul may not write Rl." There is no way that this decision can be reversed when Ul later opens R3. This is undesirable since by now ul is eligible to write Rl on the basis of the original protection requirement. The problem of dominance is illustrative of the reasoning behind the often quoted philosophy of rendering decisions at as late a point as possible.

The history keeping process of the present model is of the event keeping class. The basis of the process is a set K of system counters for access history keeping. It is
noted that $K$ is a subset of $R$, the resources, so that counter values are indeed resource values. Thus, access conditions can be defined in terms of counter values, allowing future accesses to be dependent upon corresponding events.

If history keeping requirements are connected with a particular authorization specification $p[i]$, a number of triples of the following form will be associated with the authorization:

$$(S[h],k,b[h])$$

(3.6)

where $S[h] \in S$, $k \in K$, and $b[h]$ is a single-value flag for history keeping. The condition for $S[h]$ is evaluated at the time that $p[i]$ is used for the access decision. If $C(s,S[h])$ does not hold, then no history keeping is done, regardless of any other condition. If $C(s,S[h])$ holds, the use of the flag is as follows, depending on the type of event:

<table>
<thead>
<tr>
<th>$b[h]$</th>
<th>Action Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Increment $k$ if a success event, $d(q)=1$</td>
</tr>
<tr>
<td>1</td>
<td>Increment $k$ if a failure event, $d(q)=0$</td>
</tr>
<tr>
<td>2</td>
<td>Increment $k$ independently of outcome of $d(q)$</td>
</tr>
<tr>
<td>3</td>
<td>Reset $k$ if a success event, $d(q)=1$</td>
</tr>
<tr>
<td>4</td>
<td>Reset $k$ if a failure event, $d(q)=0$</td>
</tr>
<tr>
<td>5</td>
<td>Reset $k$ independently of outcome of $d(q)$</td>
</tr>
<tr>
<td>6</td>
<td>Reset $k$ immediately, directly, and unconditionally</td>
</tr>
<tr>
<td>7</td>
<td>Increment $k$ immediately, directly, and unconditionally</td>
</tr>
</tbody>
</table>

Any implementation of this type of history keeping should also allow the authorizer the direct interactive capability to increment, reset, or display counter values. Also any protection language featuring history keeping must provide a naming scheme to make the correspondence between the event involving a given authorization specification and references to that event in a later access condition.

The role of this triple in the security space must be explained in the context of other subspaces, namely the authorized subspace and two or more access request subspaces. The triple enters the system in the company of an authorization—$p[i]$, for example—which specifies the access conditions for a certain type of access. The
authorization, $p_1$, defines a region of the authorized subspace, say, $x_1$. The triple defines a history keeping subspace associated with $x_1$ which corresponds to a single value, $k$, on the $R$ axis and an interval or set of intervals, $S[h]$, on the $S$ axis. The presence of the triple with $p_1$ means that occurrences of the accesses defined in $p_1$ are to be recorded in $k$ for those states in $S[h]$. Presumably the access condition of a later authorization, say, $p_2$, will depend upon this history. Suppose two requests, $q_1$ and $q_2$, have separate request subspaces, $y_1$ and $y_2$. Further, suppose the request $q_1$ falls in $x_1$ and $q_2$ falls in $x_2$, the subspace of $p_2$. The history keeping triple then ties the request subspaces, $y_1$ and $y_2$, together in a special relationship. This link, over time, becomes manifest as follows. Recall that a state is an $n$-tuple containing one value for each resource in $R$. One of these resources is $k_1$, the counter specified in the triple with $p_1$. The enforcement process records the occurrence of $q_1$ by incrementing $k_1$ (if $q_1$ arrives during a state of $S[h]$), and in so doing has changed the state of the system in which the second request, $q_2$, will be received. This change of state serves as the link between the two request subspaces, $y_1$ and $y_2$. From this discussion, it is learned that a language employing history keeping will need a definitional capability for assigning names to counters in order to establish the correspondence between the counter of $p_1$ and the same counter as specified in the access condition of $p_2$.

This approach to history keeping overcomes the problem of dominance by maintaining a separation between history recording and access decision making. Only information about occurrences of events can be passed from one request subspace to another by way of a counter. A separate access decision still must be made in response to a later request. The problem of storing more history information than is needed is overcome by storing no history information that is not explicitly requested by the authorizer. That is, the authorizer must decide ahead of time what events to remember and only those events are recorded. The flexibility of this approach is aimed at representing most history keeping requirements that might be expressed in a higher level protection language.

One can observe that the use of history dependent conditions may violate the confinement principle [LAMPB73]. The confinement principle requires that programs (including parts of the security software) be constrained from leaking data to an unauthorized user. The term is especially applied to "subtle paths by which data can escape." Access history dependence implies that an access decision can be
made to be a function of whether or not a certain type of access has previously occurred. If only the owner of a resource unit can establish history keeping for that resource unit, the range of dependency is very limited compared to that of a more general scheme. On the other hand, access history information with respect to any resource might be allowed in access conditions for resources that a given authorizer does own. However, this allows one authorizer to monitor the usage of other authorizers' resources. The confinement principle can be used to show that this can yield significant information about the supposedly protected resource. The choice of which approach to take in an implementation is a question of policy. The mechanisms of the model allow for the most general of the two cases.

In addition to creating access dependencies on access history, the history keeping mechanism allows access history information to be kept for any purpose. One important such purpose is to gather database usage statistics. (See the examples of this at the end of the next section.)

Examples of History Keeping

The history keeping counters can be used many ways. This first example illustrates how several logical operations can be performed via the counters.

1) Operation: AND
   Required history condition: No access to C if both events A and B have occurred.
   Implementation: Event A increments k and event B also increments k. Nothing else can change the value of k.
   Access condition: C(s,S) : k < 2
   Explanation: If neither A nor B have occurred, but not both, k=1 and access is allowed. If both have occurred, k=2 and access is denied. (Here it is assumed that neither A or B can occur more than once each.)

2) Operation: OR
   Required: No access to C if either A or B have occurred.
   Implementation: Same as for AND above.
   Access condition: C(s,S) : k=0
   Explanation: Either A or B will cause k+0
3) Operation: General n out of m  
Required: No access to B if any n out of A = (A1, A2, ..., Am) have occurred.  
Implementation: Each element of A increments k when it is performed.  
Access condition: C(s, S): k < n  
Explanation: Generalization of 1 and 2 above.

The next example illustrates a simple history dependency requirement: Any user in Ul may write in file F only if he has not read from file G. The authorizer has entered this specification:

\[ p_1 = (a, Ul, (READ), FILEG, S1) \quad (S[h] = S, k_1, b[h] = 0) \]

There may be other p's that apply to read operations from file G—for example, some members of Ul may also be elements of other groups with read privileges to file G. However, it is guaranteed that p1 will always participate in an access decision regarding Ul’s reading from file G. Therefore, since b=0, any successful read operation by Ul from file G will always be recorded by incrementing k1. Next, an authorization, say p3, must be made to create a dependency on this event:

\[ p_3 = (a, Ul, (WRITE), FILEF, S3) \]

where C(s, S3): k1=0.

The following example is mentioned in the previous section in the discussion of dominance. The requirement is: A member of Ul cannot write R1 after reading R2 unless he subsequently opens R3. Strictly speaking, this requirement has an ambiguity: What about the case when he reads R2 again after opening R3? There are two possible ways an authorizer could have meant to treat this.

1) The first interpretation is that the requirement is a one time application. Once the user opens R3 he can write R1 thereafter, regardless of further history. This requirement is declared by these specifications:

\[ p_1 = (a, Ul, (READ), R2, S1) \quad (S, k_1, 0) \]
\[ p_2 = (a, Ul, (OPEN), R3, S2) \quad (S, k_2, 0) \]
\[ p_3 = (a, Ul, (WRITE), R1, S3) \]

where C(s, S3): k1=0 + k2>0. Here a success event for p1 unconditionally causes k1 to be set to one
(incremented) and a success event for \( p_2 \) causes \( k_2 \) to be set. The access condition for writing to \( R_3 \) is either not ever having read from \( R_2 \) or having opened \( R_3 \).

2) The second interpretation is a continual application such that every read of \( R_2 \) requires a later opening of \( R_3 \) before reading \( R_1 \). This is an event for which the regular expression is:

\[
((\text{Open } R_3)^* + (\text{Read } R_2) + (\text{Open } R_3)^+) (\text{Write } R_1)^* )^*
\]

which can be shortened for better readability to:

\[
((03^* + R_2 + 03^+) W1^* )^*
\]

where \( A^* \) is the iterative closure of \( A \) and \( A^+ \) is the non-null iterative closure, \( A^+ = AA^* \). The following set of authorizations will provide the history keeping required:

\[
p_1 = (a, U_1, (READ), R_2, S_1) (S, k_1, 0)
p_2 = (a, U_1, (OPEN), R_3, S_2) (S, k_1, 3)
p_3 = (a, U_1, (WRITE), R_1, S_3)
\]

where \( C(s, S_3) : k_1=0 \). Whenever \( R_2 \) is read, \( k_1 \) is incremented but whenever \( R_3 \) is opened, \( k_1 \) is reset. Therefore, as long as at least one opening of \( R_3 \) follows any reading from \( R_1 \), the regular expression is satisfied.

The last two examples of this section are brief examples of the use of history keeping for gathering usage statistics, rather than for creating access dependencies. In the first of these examples, it is desired to find, over a given time period, the average number of legal \textit{READ} operations per attempt to read file \( A \) by a member of user group \( U_1 \). Two counters must be kept: one for the number of successful request to read \( A \), \( b[h]=0 \), and one for all attempts to read \( A \) regardless of outcome, \( b[h]=2 \). If the specification \( p_1 \) is as follows:

\[
p_1 = (a, U_1, (READ), A, S_1)
\]

the following triples must be added:

\[
(S[h]=S, k_1, b[h]=0) \ (S[h]=S, k_2, b[h]=2)
\]

At the end of the time period of interest, the value of \( k_1 \) may be divided by the value of \( k_2 \) to obtain the required average.
In this last example, as a guide to reorganizing a file structure, it is desired to find the $k$ most frequently accessed of $n$ records in the file $B$. This authorization is created:

$$p_2 = (a, \text{GENERAL}, \text{ALL}, B, S)$$

If there are existing access restrictions on any part of $B$, this authorization will not override them, since the AND operation of the enforcement process will require all access conditions to be met. Then these $n$ history keeping triples can be associated with $p_2$:

$$(S_1, k_1, \emptyset), (S_2, k_2, \emptyset), \ldots, (S_n, k_n, \emptyset)$$

where $C(s, S[i])$: record-accessed $= \text{record-}i$, for $i=1,2,\ldots,n$.

At the end of the designated time period the $k$ counters with the highest values will indicate the $k$ most active records. An alternate approach (but probably less desirable) is to have $n$ authorizations of the form:

$$p[i] = (a, \text{GENERAL}, \text{ALL}, R[i], S) \ (S[h]=S, k[i], \emptyset)$$

where $R[i]$ refers to the $i$-th record of file $B$, for $i=1,2,\ldots,n$.

In a later section in this chapter there is an example of history keeping used as a threat monitor to control the invocation of an auxiliary procedure.

**Auxiliary Program Invocation**

Whereas the history keeping process of the model is concerned with access decision making, the auxiliary program invocation process of the model is concerned with additional protection measures before, during, and after an access decision is made. For example, before the decision for an access request is made, a protection measure invoked by the process to authenticate user identification may render the access decision superfluous. In the course of making an access decision, another protection measure may be invoked by the process which records the decision making information and events for purposes of threat monitoring. Similarly, after an access decision is made, a protection measure may be invoked by the process for the purpose of making a journal for later auditing.
Auxiliary programs can provide many flexible protection capabilities found in procedurally oriented access control schemes [HOFFFL71]. There are also procedural features in other database specifications [CODAS71] which can be accommodated by auxiliary program invocations. The "trap extension" in MULTICS [SALTJ74] uses a one-bit flag associated with an access control list entry to invoke procedures at access time. As the MULTICS designers discovered, a fully general implementation of these procedures is beset with difficulties. The main problems are how to decide what access rights can be used by the procedure and how can the user be sure he is safe from damage to security or integrity by the procedures themselves? Since the procedures are specified by, and run on behalf of, the authorizers who created the associated authorization specifications, the procedures in this model will be run under the authorizer's rights. The effect should be no less safe than the situation in which the same authorizer and user are logged-in simultaneously, each using the system legally under his own rights.

Since access decisions are made on the basis of their corresponding authorization specifications, the procedural protection measures are associated with these authorization specifications. Each auxiliary program is invoked in conjunction with an event similar to the way that history keeping is controlled. More specifically, any given authorization may be associated with a number of concomitant triples of the form:

\[ v = (S[v], w, b[v]) \] (3.7)

corresponding to the auxiliary programs to be invoked whenever that authorization is used in an access decision. \( C(s, S[v]) \) is the invocation condition. If \( C(s, S[v]) \) does not hold, the corresponding program is not invoked, regardless of any other condition. If, during, the enforcement process for a request \( q \) the invocation condition, \( C(s, S[v]) \)—of any of the auxiliary procedures contained in the domain of auxiliary invocation (see chapter IV)—holds, processing is suspended and each of these procedures (with the "true" invocation conditions) is invoked at the appropriate time. The element \( w \) in each triple names the corresponding procedure. The invocation flag, \( b[v] \), is a parameter to indicate the activation sequences of these procedures. Its meaning is defined as follows:
The role of full specifications

Activation Sequence:

<table>
<thead>
<tr>
<th>b(v)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execute w before making the access decision</td>
</tr>
<tr>
<td>1</td>
<td>Execute w after making the access decision, regardless of the outcome of that decision</td>
</tr>
<tr>
<td>2</td>
<td>Execute w if d(q)=1</td>
</tr>
<tr>
<td>3</td>
<td>Execute w if d(q)=0</td>
</tr>
<tr>
<td>4</td>
<td>Execute w at login time</td>
</tr>
</tbody>
</table>

The case of b(v)=0 allows for a dependency of the access decision on the results of the auxiliary procedure. This dependency is communicated through the access condition, which, in turn, can be dependent on system counters (also used for history keeping) and other system variables (such as status indicators) previously established for that purpose. In these cases the authorizer will establish the counter and specify the appropriate check in his access condition. Each such procedure will then clear the corresponding counter, compute its results, and set (or not set) the counter based on those results. Thus, the access dependency on the outcome of the procedure is automatically taken into account and no special checks are needed in the enforcement process to keep track of the results of auxiliary procedures. There is an example of this use of auxiliary procedures at the end of the next section.

An authorizer, of course, must own every resource unit for which he specifies auxiliary program invocations. This means that, for example, even the SYSADMIN can be prevented from monitoring the use of private files. Since several authorizations are used in making an access decision for one request, potentially several groups of invocation triples may be associated with a given access decision. The corresponding programs of all these triples together may be activated in the course of handling this request by the enforcement process.

Following are some applications for which the sophisticated authorizer might use the protection measures.

1) **Authentication of User Identification**—To use the data base system at all a user must initially login using some form of personal identification. It is sometimes possible for an illegal user to forge this and enter the system under a false identification. As an additional safeguard for the security of extra sensitive data, the authorizer may specify a more complex interactive procedure to verify the user's identification [PURDU74, EVANA74] to be invoked for any particular access requests. This
furnishes the so called login program capability [HSIAD68] (which provides user verification at the file level) to be used at any level. Since an authentication procedure may affect access, it should be invoked before the evaluation of the effective access condition. In many cases user authentication is desirable at login time. Therefore, invocation triples will have the parameter b[v]=0 or b[v]=4 for authentication.

2) Journal and Audit Trail—A record of access history of a particular user, or of a particular resource can be maintained for later report generation. In the event of a suspected attempt to violate security the audit trail may be analyzed to reconstruct the series of events. An access journal could also be used as a database performance monitor, allowing a restructuring of the data base to match more closely certain access activities. Since these functions occur after an access decision, the activation sequence will employ parameters b[v]=1, 2, or 3 for journals or audit trails.

3) Alarm and Recovery—The question of what to do after an attempted access violation has been detected is an issue given little or no attention by many existing models of protection. In a real system it is absolutely necessary to provide the means to inform the proper authorities of such violations and to resolve these denied accesses in an appropriate manner. The system cannot always just suspend the operations of a violator, especially when the violation is due to an innocent error. It has been determined that the leading threats against security and integrity are errors and omissions by honest people [INTER74]. The IBM Resource Security System [HOFFL73a] has a provision for an actual alarm on the console. However, a more general ability for an owner-defined response to an illegal access request is the goal of this model. For many applications it will be necessary to recover without alarm and apprehension regardless of whether the attempt was malicious or innocent. For example, an unsensitive file may be read-only to protect its integrity. An attempt to write over this data should just be blocked, allowing each user to continue immediately with his other work. There
can be a number of standard system alarm and recovery actions. Resource owners may supply additional procedures. Alarm conditions represented by $S[v]$, the first element of the triple in (3.7), may be a function of history; for example an alarm may be caused by a certain number of consecutive unsuccessful attempts of a user experimenting to penetrate a given data subset. Since these functions are required only when access is denied, the activation sequence will use $b[v]=3$ for alarm and recovery.

4) Threat Monitoring (Surveillance)—For especially sensitive parts of the data base the technique of threat monitoring [PETEH67, HOFFL73b] can be used to detect potential penetration attempts, to perform surveillance, and to gather evidence of misdoings by users. Even an expert cannot penetrate a strange system without some experimentation to discover the protection rules and how they are enforced. The system can make use of information about his operations to uncover threats. The nature of the monitoring mechanism is very similar to the alarm and recovery described above, except that it is invoked when it is suspected that penetration attempts are occurring (when usage patterns vary dramatically from typical). As an example, a threat indication can be raised if more than a certain number of updates to a resource unit are made by a given user in the same day. Threat monitoring should be invoked following a decision to allow access, thus calling for a value of $b[v]=2$.

5) Input Data Validation— Many data base systems can benefit from input data validation to protect the integrity of the data. Input validation involves checking ranges of values, combination of values with certain other values occurring together in a record, and existence of required field values. Often this checking is dispersed inflexibly into the code of the access or processing routines. The inclusion of these checks as conditionally callable auxiliary procedures increases the flexibility and modularity considerably. It also serves to locate all the input checking rules in one place, which helps to give an accurate picture of their overall effect. Input data validation
is also invoked after a decision to allow access, again indicating a value of b[v]=2.

6) **Progressive Authorization**—Occasionally, it is desirable to evaluate security conditions in a specific order to produce a progressive authorization sequence. As an example, consider a data collection R1 in a system. A user enters a new program, G, which is supposed to perform a certain transformation on R1 (some kind of update operation). Assuming R1 is an important resource unit, it must be protected for destruction by a malfunctioning program. The system may, therefore, want to test the program on a special collection of test data, R2, for which the result of the transformation is known. The user first may be required to satisfy some other conditions to access either R1 or R2, then he must satisfactorily run G on R2 before accessing R1. Auxiliary procedures may be used to verify the results of G on R2. Since access may depend on progressive authorization, it should be invoked before the access decision, requiring b[v]=0.

7) **On-Conditions**—This category is a general catch-all for other applications of auxiliary procedures. In much the same manner as an ON-CONDITION of the PL/I programming language, the specified procedures will be invoked if the specified condition is satisfied during an attempt to use a resource involving the authorization which corresponds to the particular auxiliary procedures. A value of b[v]=1, 2, or 3 is suitable.

**Examples of Auxiliary Program Invocations**

Many examples are suggested in the preceding classes of applications for auxiliary programs. This section contains more examples to illustrate the use of these programs in specific situations.

Suppose an owner, for some administrative reason, must give the necessary rights to maintain a file to just one fairly large group, Ul. Since the file, say R1, should
mostly be used in a read mode, the owner wants additional control over updates. He wants to be notified with a message on the terminal in his office, stating the user identification, terminal number, date, time, and records affected, every time the file is written into or updated. He will use a specification like this:

\[ p_1 = (a, U_1, (\text{UPDATE}, \text{WRITE}), R_1, S_1) (\xi, \text{REPORT}, 2) \]

Here \( C(s, S[v]) \) is fixed as a constant "true" and \( b=2 \), making invocation of \( \text{REPORT} \), the message writing procedure, unconditional whenever a success event involving \( p_1 \) occurs.

The next example is a more sophisticated use of threat monitoring. Suppose an auxiliary program named \( \text{STATS} \) is used to keep track of the average usage rates, under various conditions, of a file, \( R_2 \). The same program compares each current user's statistics with the overall figures. When the usage rate exceeds these averages by some criterion amount (say, in the number of accesses per hour), the owner is informed and, perhaps, an explanation may be requested of the user as a part of the record of the transaction. The specification will be similar to that of the last example:

\[ p_2 = (a, U_2, \text{ALL}, R_2, S_2) (\xi, \text{STATS}, 2) \]

Next, referring back to an example in an early section of this chapter (on examples of authorizations), the following example has to do with the discovery of a problem regarding a group of records entered into \( \text{FILEA} \). Access to these records, known by their range of accession numbers, has been blocked until the problem can be corrected. Since the blocking of access in this case may affect supposedly authorized users, it would be helpful to offer rebuffed requesters an explanatory message to the effect that the problem is temporary. The best way to send such a message at the appropriate time is to use an auxiliary program (named \( \text{MESSAGE} \)) invoked for a failure event associated with the general use of the specified records in \( \text{FILEA} \). Here is the authorization used to block access, together with the added auxiliary invocation triple:

\[ p_5 = (\text{SYSADMIN}, \text{GENERAL}, \text{ALL}, R_5, \text{NULL}) (\xi, \text{MESSAGE}, 3) \]

The following example illustrates how history keeping and auxiliary program invocation can be used together to help an owner keep track of the use of his resources. Suppose the authorizer wants to be informed by some
procedure called ALARM after more than $x$ unsuccessful attempts to access a certain resource unit, $R7$. Suppose it is known that authorization $p2$ is instrumental in the control of access to $R7$. Any authorization naming $R7$ will suffice, since every such authorization will participate in the access decision, causing a corresponding event. Consider this authorization:

$$p2 = (a, U, E, R7, S) (S[h]=S, k, l) (S[v]=S, ALARM, 3)$$

where $C(s, S1) : k > x$. The counter $k$ will be incremented for each failure event. The alarm procedure will be invoked for a failure event when the counter has been incremented more than $x$ times. The authorizer will need to reset the counter himself when he desires to do so.

The last example in this section illustrates the use of an auxiliary procedure to influence the access decision. Suppose that authorizer $a$, who owns resource $R$, wants to execute an interactive identification verifying program called VERIFY whenever a member of $U4$ asks to write in $R5$. The program reaches its decision based on a certain dialogue with the user. The procedure has this structure:

```
PROC VERIFY:
  RESET k
  verify identification
  if identification correct, SET k
  HALT
END VERIFY
```

The following authorization will then suffice:

$$p6 = (a, U4, (WRITE), R5, S) (S[v]=S, VERIFY, b[v]=0)$$

where $C(s, S1) : k>0$ is used to establish the access dependency, and $b[v]=0$ is used to invoke the procedure before the access condition is evaluated.

**Extended Resources**

This section is devoted to the special problem of ensuring safe access to data when processing control is passed from a user to a program having, in some sense, greater access rights than the user. This problem is related to a process which has been called *amplification*.
[JONEA73]. Special consideration [LAMPB73] must be given to ensure that nothing illegal is passed to the user by way of the program. A mechanism for allowing this temporary enhancement of access rights will permit certain kinds of operations on data which cannot otherwise be directly accessed by the user. For example, a user may sort a file which he cannot read. Sorting does not require him to retrieve any values himself; it requires only that the sort program temporarily access the values during its manipulation of the data. Another example is the gathering of statistics to answer questions such as "How many records have SALARY > 15,000?" or "What is the average age of all the people in the personnel file?" Built-in data base statistical functions such as COUNT, AVG, and SUM [BOYCR75] are other examples of this type of operation.

To say, as just stated, that a user passes control to a program with higher level access rights tends to obscure the reality of the problem. From a security viewpoint, it is not as much a question of what that program may do as it is what information that user may get to or from that program. In other words, the program's rights, more properly interpreted, are actually the user's rights to use the program in a certain way. The programs are just packaged data operations that the user performs on various parts of the data base in various ways. This viewpoint is supported by a number of works in the literature. For example, MULTICS processes [POPEG74b] are identified by the user for whom they act as agents. In one model [BELLD73], it is stated that

Assuming an environment wherein processes are surrogates for users, we shall speak of processes making requests for access to objects, requests to create objects, or requests to delete objects. The reader should understand that such requests represent the intentions of the users of the system.

This relationship can have a direct bearing on the organization of the data base system in setting the interface between user responsibility and system responsibility [MANOF75]. The implication is, if a user needs only averages, compute the averages and give them to him instead of giving him all the values and letting him compute the averages for himself. The access operation then becomes a combination of record retrieval and computation. This subtle relationship among the user, the procedures, and the access rights cannot be recognized if users are included as resources [WEISC69] or procedures are included as users...
There are some obvious potential threats to security in the situations just described. To control these threats several measures for ensuring safe access to data via a procedure are required. First, the program must be certified [POPE74a, GOODD74], or at least trustworthy, that it will not return unauthorized information to the user. This means the program must be constructed in a way such that it checks parameters passed to and from it and that it is sensitive to certain situations within the data base, situations which must be treated carefully to prevent illegal deductions from being made about the data. For example, an average based on a small number of data items can be revealing [HOFFL73b]. Secondly, the program itself must be protected so that it cannot be modified. Furthermore, the protection mechanism must be capable of representing the full fact that a user is authorized to execute a procedure for the purpose of making specified types of accesses to specified resources. The first of these requirements (program proving) will not be considered in this work. The second requirement, protection from modification, can be accomplished with an access control mechanism. The requirement for the full representation of the relationships among the user, the procedure, and the resources is the subject of the rest of this section.

Consider an example in which it is desired to authorize U1 to execute a system program called W1 to read R1. First, an authorization is needed to allow the user group U1 to execute the procedure W1. This is given below:

\[ p1 = (a, U1, \{EXEC\}, W1, S) \]

Next an authorization for W1 is required:

\[ p2 = (a, W1, \{READ\}, R1, S) \]

The full meaning of the intended authorization, however, requires these two to be coupled into a single extended authorization, perhaps, by a link:

\[ p1 = (a, U1, \{EXEC\}, W1, S) \]
\[ p2 = (a, W1, \{READ\}, R1, S) \]
This link specifies the kind of use that \( U_1 \) can make of \( W_1 \). Lacking this subtle control of the \texttt{READ} operation, any other user with execute-access to \( W_1 \) would be able to read \( R_1 \) with it. The procedure \( W_1 \) is called the \textit{extended resource}. Since a request for use of \( W_1 \) would have this appearance:

\[ q = (u, W_1, R_1, s) \]

it might seem to be more appropriate to call \( W_1 \) a data operation. But, represented as an element of \( E \), \( W_1 \) cannot have different access rights for different users who call it. In fact, nothing can then be stated about which of the elemental access operations (the original elements of \( E \)) the procedure may use on the target resource. Therefore, two authorizations are used, connected together—one to represent the right to use the procedure and the other, the rights of the procedure to access the resource on behalf of that particular user. The second of these authorizations serves as an extension of the resource which is the procedure. In general, then, an extended resource will need different access rights for different callers, but sometimes the procedure will use the same rights for several callers. Thus, several user authorizations can be coupled with the same extended resource (e.g., procedure) authorization. For example, consider the following authorizations:

\[
(a, U_1, \texttt{EXEC}), \texttt{SORT}, S_1) (a, U_2, \texttt{EXEC}), \texttt{SORT}, S_2) (a, U_3, \texttt{EXEC}), \texttt{SORT}, S
\]

\[
(a, \texttt{SORT}, \texttt{READ}, \texttt{WRITE}), R_1, S_3
\]

Notice that another level of security is introduced. The conditions for both state subsets \( S_1 \) and \( S_3 \) must be satisfied in order for \( U_1 \) to be able to complete his sorting operation. A very significant use of \( S_3 \), the access condition for the use of \texttt{SORT}, is the checking of parameters passed to and from that procedure. Of course, the sort procedure may also be coupled with other procedures to carry out its operations. The important thing about these extended authorizations is that each coupled sequence acts as just one authorization. Its parts may not be used separately. Such a sequence may be represented by concatenating the authorizations as follows:

\[
(a, U_1, \texttt{EXEC}), \texttt{SORT}, S_1) \cdot (a, \texttt{SORT}, \texttt{READ}, \texttt{WRITE}), R_1, S_3
\]
However, a more general representation will be used, to facilitate sharing of extended resources. Each authorization specification may have associated with it a pair:

\[(b[x], L)\]  \hspace{1cm} (3.8)

where \(b[x]\) is a binary extended resources flag telling if this authorization has been linked to by another authorization (i.e., \(b[x]=1\)) or not (\(b[x]=0\)), and \(L\) is a (possibly null) link to another authorization. In the previous example of the SORT routine, assume that SORT must also call MERGE. The authorizations will appear as follows:

\[
\begin{align*}
    p_1 &= (a, U_1, (EXEC), SORT, S_1) \quad (0, p_4) \\
    p_2 &= (a, U_2, (EXEC), SORT, S_2) \quad (0, p_4) \\
    p_3 &= (a, U_3, (EXEC), SORT, S_3) \quad (0, p_4) \\
    p_4 &= (a, SORT, (EXEC), MERGE, S_3) \quad (1, p_5) \\
    p_5 &= (a, MERGE, (READ, WRITE), R_1, S_2) \quad (1, 0)
\end{align*}
\]

Interlocking for Protection During Shared Use of Resources

In an environment of shared resources, interlocking is required for certain operations whose untimely interruption might cause a loss in integrity of the data base. The locking and unlocking of data is considered by some to be an entirely separate function from access control [HOFFL71] and, often, the operating system or DBMS does handle this task separately from the access control system. However, it may be convenient in some cases to include the interlock capability in an access control system. In fact, since interlocking is most certainly a means of controlling access, no general access control system should lack this capability. Furthermore, the same capability can be used to temporarily withdraw access rights to or temporarily block all use of a particular resource [HISIAD68]. These uses are very convenient, especially for the SYSADMIN [POPEG74b].

In the section on modelling the authorization process in chapter IV, the set of all resource units defined to the system by all authorizers is introduced. Each of these resource units has the potential of being tagged with the
identifier of a user who wishes to lock that resource. If the resource being locked overlaps several resource units, each of them will be tagged. When the resources are to be unlocked, the tag is removed. The enforcement process, described in a section of chapter IV, will allow access to tagged resources only by the user identified in the tag. This small extension to the resource unit list adds considerable strength to the model. This is done by lending many of the capabilities of access lists without sacrificing any of the advantages of this capability oriented model of overlapping sets. Operational language statements, such as ALLOCATE/FREE or LOCK/UNLOCK, may be used to set and reset these bits directly. Also all update-type data operations will be forced to include the proper interlocks in their definitions. For example, an access type UPDATE may be defined by:

PROC UPDATE:
  LOCK R[i]
  LOOP: READ from R[i]
  Find place to update?
    If not, LOOP
    WRITE into R[i] (the actual update)
  UNLOCK R[i]
  End UPDATE

The problem of deadlocks, the constant companion of interlocks, is not within the scope of this dissertation, except to say that there do exist strategies for imposing constraints on user requests in order to avoid deadlocks.
CHAPTER IV

PROCESSES OF THE MODEL

The process itself is the actuality.
—Alfred North Whitehead

A clever student will soon learn how to model... almost without soiling his hands.
—Albert Toft, Modelling and Sculpture

**modelling the Authorization Process**

**In the Security Space**

The model presented here has two primary processes, the authorization and the enforcement. As stated previously, the mode of presentation will be mostly on an intuitive level.

The authorization process translates the protection language expressions of authorizers into internal representations of access control information. This internal representation can be viewed in several ways. The approach herein is based on the principle that access by a member of $U$ to a resource of $R$ is allowed only as the result of an authorization specification. No access can be made without the appropriate authorization. This approach, termed a "closed scheme" [DALE65], indicates a best answer to the question of whether to represent within the system what is authorized or to represent what is not authorized. Often, having one case more numerous than the other is
adduced as a basis for the best answer to this question. However, on the basis of protection, to represent what is authorized may be argued to be more secure in that any errors of omission (temporarily overlooked authorization specifications) will not allow a breach of security. In the model discussed here, because of the ability to group users and data in arbitrary subsets, this approach will also be frugal with storage of access control information. This is true even in cases where most of the data base is accessible to most of the users. Because of the ability to group data into subsets it takes no more storage to represent cases which allow access to large portions of the data than it does to represent cases allowing access to small portions. The storage requirement is more affected by the number of different portions specified—for protection in one case and for access in the other case.

Use of an access condition which has a constant "false" value gives a peremptory guarantee that access will be denied regardless of the access condition in any other authorizations. Therefore, it is also possible to represent explicitly what operations are not allowed, where this is deemed necessary for an absolute measure of security. Thus, the advantages of both approaches may be had simultaneously.

Any approach to modelling the authorization process must face the question of consistency, i.e., of how to resolve apparently conflicting authorizations received at different times. Within the approach taken here—by allowing authorization specifications to enter the system, by collecting these specifications without any checking for commonality and by using the cumulative effect of the specifications as an effective authorization—the consistency of the authorization process can be maintained. For example, two authorizations may exist for the same user group, operations, and resource unit. No attempt is made to combine these by the authorization process. The enforcement process will, however, require all specifications for access to a given resource to be checked before that access is considered. This allows for no internal inconsistencies. If one authorization says to permit an access and another says to deny the same access, it's a case of external inconsistency (the authorizers cannot agree) and a conservative solution (deny access) is automatically supplied by the AND of the two conditions in the enforcement process. (See the section on the enforcement process later in this chapter.) If two different conditions, however, are specified by two different authorizers for the same access, each access condition is treated as a further restriction on the other and the AND of them reflects the needs of both
Validation—The First Step

Validation is the first, and perhaps most important, step in the processing of authorizations. Specification validation ensures that each incoming specification is in the valid part of the security space. It reflects the set of rules governing the various degrees of ownership as shown in the following definition. There are many ways to formulate a set of rules of ownership, especially with respect to the role of the SYSADMIN. This, doubtlessly, is a case of policy at the highest level, the mechanisms to be fixed for a specific implementation by means of semantic parameter specifications (see section on semantic specification constructs in chapters V and VI). For the sake of illustrating the validation process within the rest of the model, an arrangement is arbitrarily chosen on the basis of what appears to be reasonable and general enough to suit the needs of a typical system. The modularity of the model allows the validation process to be tailored to a specific implementation without affecting other processes. The following, then, is a typical set of rules to validate an authorization p in state s:

1) SYSADMIN is the only authorizer who can assign OWN, and he may do so only if he also owns the resource unit specified.

2) An authorizer must own a resource unit to grant SUBOWN to it.

3) An authorizer must own or subown a resource unit to grant access to it.

(4.1)

The informal statement of these rules leaves a few considerations imprecise. The most important of those considerations is that which is referred to as ownership of a resource by an authorizer. The reader is reminded that the meaning of OWN may require the evaluation of conditions to determine ownership at any given instant in time. In that regard, even the authorization process requires reference to the enforcement process. To determine if an authorizer owns a certain resource unit, it must be seen if
there are any authorizations already in \( P \) which have assigned that ownership. If any such authorizations are found then their access conditions must be evaluated to determine if this ownership obtains in the present state. This is tantamount to the enforcement of an access request for the data operation of \( \text{OWN} \). Thus, the question of authorizer \( a \) owning \( R \) can be stated as a query:

\[
q = (a, \text{OWN}, R, s)
\]

and answered by the access decision predicate:

\[
d(a, \text{OWN}, R, s)
\]

At this point one more bit of notation is needed, which can also be used later. Some projection functions which operate on \( n \)-tuples are introduced. In general, where \( p = (x_1, x_2, \ldots, x_n) \), a set of projection functions \( i(p) = x_i \), is defined for \( i = 1, 2, \ldots, n \). Since the tuples of this model carry specific element names, mnemonic meaning is better served by the use of the names of the elements instead of a general scheme with subscripts. Therefore, for \( p = (a, U, E, R, S) \):

\[
\begin{align*}
a(p) &= a \\
U(p) &= U \\
E(p) &= E \\
R(p) &= R \\
S(p) &= S
\end{align*}
\]

Although the same letter is used as an element and as a function, ambiguity is avoided by this rule: The letter appearing alone always denotes the element and the letter appearing with an argument refers to the function. The validation rules of (4.1) can now be restated symbolically (the reader is referred to appendix D for this).

The portion of the points, entered into the security space via authorizations which pass the validity test, make up the \textit{valid subspace} of authorization. This model will deal with only those authorized subspaces which are valid. In the case of private files created by users, unconditional ownership is automatically declared by the system at the time of creation by a specification resembling this: \( p = (\ast, u, \text{OWN}, R, S) \). Since the authorizer is meaningless here, "\( \ast \)" is used to fill in its place in the 5-tuple. Unless explicitly authorized as such by the creating owner, such data sets are not owned by the SYSADMIN.
Granting Access Rights

Each valid authorization specification of the form (3.3) names a group $U$ of $\mathcal{U}$ and a unit $R$ of $\mathcal{R}$. A sequence of valid authorizations, which enters the system as an autonomous stream, then forms a collection of these groups and units. One of the steps in the authorization process is to maintain these collections by adding new elements from each incoming valid specification. The collections are defined as follows:

**DEFN:** $\mathcal{U} = \{U_1, U_2, \ldots, U[n]\}$ is the set of unique user groups having been defined to the system, including a distinguished member called GENERAL.

(4.2)

The GENERAL group is intended to contain all users and to give every user automatically some minimal rights of access. This is an answer to the problem of wasting storage to represent for example, the rights of everyone to use a public file [LAMPB71]. Recall, also that the grouping of users is a logical grouping based on common access rights, not necessarily based on organizational or administrative structure. Every user in a group is defined to have the rights of the group.

**DEFN:** $\mathcal{R} = \{R_1, R_2, \ldots, R[m]\}$ is the set of unique resource units, or data subsets, having been defined to the system.

(4.3)

**DEFN:** $\mathcal{P} = \{p_1, p_2, \ldots, p[k]\}$ is the net collection of valid authorization specifications up to a given point in time. Each element, $p$, is a 5-tuple of the form (3.3).

(4.4)

It may be helpful to elucidate. It is important, first of all, to emphasize that $\mathcal{U}$ and $\mathcal{R}$ are not, generally, the power sets of $\mathcal{U}$ and $\mathcal{R}$. Rather, $\mathcal{U}$ is the set of subsets of $\mathcal{U}$ which have up to some point in time been defined for the system by an external source and $\mathcal{R}$ is a similar set of resource subsets. No $U$ or $R$ can exist in an element of $\mathcal{P}$ without also being an element of $\mathcal{U}$ or $\mathcal{R}$, respectively.
As developed in the previous sections, the general form of a full authorization contains a basic authorization plus various triples for history keeping and program invocations, as well as a pair for extended resource usage:

\[(a, U, E, R, S), (S[h], k[n], b[h]), \ldots, (S[n], k[n], b[n]), (S[v], w[v], b[v]), \ldots, (S[n], w[n], b[n]), (b[x], L)]\]

(4.5)

The authorization process must therefore collect these triples and pairs and keep them associated with the proper authorizations. It must also allocate, clear, and assign a system counter for each authorization requiring history keeping. In addition, each auxiliary program to be invoked must be compiled, including careful diagnostics. In cases requiring use of extended resources, b[x] must be initially set to zero, indicating nothing is yet linked to this authorization. Also the link, L must be established. The corresponding b[x] of the linked authorization must be set to one.

Now that the process required to accept authorizations into the system has been seen, it is interesting to consider the effect on the shape of the authorized subspace. Although the change in shape of the authorized subspace as a result of each new authorization is discussed, the reader should realize that the authorization process produces no real authorized subspace. Rather, this subspace is merely an abstract means for visualizing the combined effect of the collection of authorization specifications.

If U, the specified user group is not an element of \(\mathcal{U}\), the specification merely adds to the authorized subspace, indicating that the specification is for a new user group U. If \(U \in \mathcal{U}\), then there are existing specifications concerning U. In this case, the specified operation set E of the new authorization is examined. If E does not match the operation set in the existing authorization for the user group of which U belongs, then E is a newly authorized operation set. The authorized subspace is enlarged by adding the operations of E. If both U and E of a new specification match the corresponding elements of some n previous authorizations, the resulting authorized subspace is determined by the following enforcement rules:

1) Access to resources contained in the resource unit of only one authorization are subject to just the one condition of the corresponding state subset.
2) Access to resources common to the resource units of more than one authorization are subject to the condition of the intersection of the corresponding state subsets.

For clarification, consider the following two authorizations:

\[ p_1 = (a, U, E, R_1, S_1) \]
\[ p_2 = (a, U, E, R_2, S_2) \]

In general, those resources contained in only \( R_1 \) are denoted by \( R_1 - R_2 \). According to rule (1) above, access to these resources is subject to the condition of \( S_1 \). In a two-dimensional projection of the authorized subspace for resources and states (with \( a, U, \) and \( E \) fixed) this results in these points: \( (R_1 - R_2) \times S_1 \). Similarly, access to the resources contained only in \( R_2 \) are subject to the same condition of \( S_2 \), causing these points to be in the same authorized subspace: \( (R_2 - R_1) \times S_2 \). Furthermore, resources common to \( R_1 \) and \( R_2 \) are indicated by \( (R_1 \cap R_2) \). Access to these is subject to the condition of \( (S_1 \cap S_2) \) by rule (2). Thus, \( (R_1 \cap R_2) \times (S_1 \cap S_2) \) is also in the resource and state projection of the authorized subspace. The full subspace resulting from these two authorizations, \( p_1 \) and \( p_2 \) is the union of these three cases:

\[ [(R_1 - R_2) \times S_1] \cup [(R_2 - R_1) \times S_2] \cup [(R_1 \cap R_2) \times (S_1 \cap S_2)] \]  
(4.6)

The expression (4.6) is a general expression for the resource and state projection of the authorized subspace of two authorization matching in \( a, U, \) and \( E \). This expression can be applied recursively to describe this projection in the case of \( n \) such specifications. This case is considered briefly in appendix D for completeness.

The example in the section on the relation between individual resources and resource units in appendix F illustrates the application of (4.6) for the case of two authorizations. In that case the authorized subspace is characterized by:

\[ ((r_1) \times (s_1, s_2)) \cup ((r_4) \times (s_2, s_3)) \cup ((r_2, r_3) \times (s_2)) \]

Example 4 of the last section in this chapter also illustrates graphically the use of (4.6). In other cases where \( R_1 = R_2 \), one can see that (4.6) reduces to:
PROCESSES OF THE MODEL

MODELLING THE AUTHORIZATION PROCESS

\[ R_1 \times (S_1 \cap S_2) \]

indicating that an additional condition has been specified for access to \( R_1 \).

Withdrawing Access Rights

Authorizations which remove access to a resource are treated somewhat differently. Here, the simplest approach appears to be to remove all authorizations which contain resource units exactly matching that resource unit specified for retraction. A more elaborate approach is to find all authorizations for resource units which partially overlap the resource unit in question and to modify those authorizations by deleting the parts of the resource units which overlap. A temporary withdrawal of rights to access a resource can be accomplished by the LOCK mechanism (see the section on this at the end of chapter III). The withdrawal of access rights is best described by illustrating it with examples. Initially, let there be only three basic authorizations in the system:

\[
(a, U, E_1, R_1, S_1) \\
(a, U, E_2, R_2, S_2) \\
(a, U, E_3, R_1, S_3)
\]

where \( E_1 = (e_1, e_2, e_3) \), \( E_2 = (e_2, e_3) \), \( E_3 = (e_1) \), and \( R_1 \subseteq R_2 \). These are followed by this authorization for unconditional withdrawal of access rights:

\[
(a, U, E_4, R_1)
\]

where \( E_4 = (e_2) \). The effect is:

\[
(a, U, E_5, R_1, S_1) \\
(a, U, E_2, R_2, S_2) \\
(a, U, E_3, R_1, S_3)
\]

where \( E_5 = E_1 - E_4 = (e_1, e_3) \). Although \( R_1 \) in the last of the three specifications matches the withdrawal authorization, \( E_3 \) does not have any elements in common with \( E_4 \). Thus, the last specification is unaffected.

Next, consider a new withdrawal specification:
Instead of the three original specifications, the system has by now the following two:

\[(a, U, E_6, R_1, S_1)\]
\[(a, U, E_2, R_2, S_2)\]

where \(E_6 = E_5 - E_3 = \{e_3\}\)

The model will not include more complex manipulations of authorizations, such as updates which change parts of existing authorizations. These manipulations might possibly be facilitated by a combination of authorizations for withdrawing and granting access rights. Refer to chapters V and VI and appendix C for protection language statements used to grant and withdraw rights.

Other Tasks

In addition to validation, granting, and removing of authorization specifications, there are other tasks of the authorization process. These tasks are directed toward processing user definitions which protection languages at various levels of sophistication will employ. These definitions allow each authorizer, for example, to specify membership in a user group which may be defined either by enumeration of the members or by means of a condition used as a characteristic function. The authorization process must also accept and process definitions of resource units and of conditions. And, of course, a journal of all authorizations will be a valuable service to the SYSADMIN and other owners.

In a protection system it is also important to have the ability to display authorizations so that direct feedback can be had for viewing the results of each authorization. Therefore, the authorization process will contain this function, too.

Examples of authorization and enforcement may be found in the last section of this chapter and in appendix F.
Modelling the Enforcement Process in the Security Space

The authorization process controls the changing shape of the protean subspace of authority as it grows and diminishes subject to the autonomous stream of authorization specifications. The enforcement process dynamically consults the authorized security subspace to make access decisions based on the state of the system at the time of the access request. In the section on history keeping in chapter III a p-event is defined as the use of the authorization, p, in an access decision. In this section the access decision, perhaps the most important concept in access enforcement, will be explained in more detail. There will usually be an access decision in response to each query. In this model the access decision is dependent on a set of authorizations, typically having several members. This set will be called the domain of authorization for that access decision.

A Dilemma in Enforcement

As with the authorization process, there are some decisions to be made regarding possible ways to model the enforcement process. For example, suppose the access request subspace only partially lies within the authorized subspace. Should the enforcement process allow access corresponding to just the authorized portion or should the entire request be denied? A straightforward approach is to deny the request and force the requester to modify the request if he wants any access. There are some pros and cons. If it can be assumed that the user needed everything he requested, it will often be the case that he cannot proceed without obtaining it all. Further, if he is given only part of the requested resources, he must be told. Otherwise, his results may be vitiated, since they are based on incomplete resources. On the other hand, if he is told that the granted resources are incomplete, he is given considerable information from which some of the forbidden information may possibly be deduced. Alternatively, a flat rejection is not useful in many cases. This question appears to be a basic matter of policy, at the security system semantics level (see the section on semantic specification constructs of protection languages in chapters V and VI). The approach to access control by query
modification [STONM74] is an example of irreversibly building this policy into a model. It must be concluded that a general model of access control should be capable of both approaches. In this chapter the enforcement process will first be discussed in terms of full rejection, and later, the modification necessary for partial rejection will be shown.

Upon closer inspection, the question of full vs. partial rejection is operational at many levels. In a slightly more general view, it is an issue of resolution of enforcement. That is, at how fine a grain can (or does) the enforcement process discriminate to determine which accesses should be allowed or denied? The finest resolution at which enforcement can be made is limited by the resolution at which resources may be defined (for example, fields and subfields within records in files). The coarsest enforcement resolution is at the level of the entire database. An example of a protection mechanism that operates at this coarsest level is a mechanism that has only user identification. Unidentified users may not access any data. Once identified, a user may access any and all of the data. Most protection systems operate at some level between the extremes. Below the operational level of enforcement resolution the mode is always full rejection, and above it the mode is partial rejection. For example, consider a file system which has the ability to describe fields and subfields. However, for enforcement it has been decided that no partial records will be accessed. That is, the enforcement process will allow access either to an entire record or else to no part of the record. The operational level of enforcement resolution is the record level in this case. Below that level is full rejection (all or nothing). Above that level is partial rejection; i.e., access to part of a group of records with certain common characteristics (such as attribute-value pairs) may be denied while access to the remaining part may be allowed.

Implications of Overlapping User Groups—an OR Operation

The grouping of users, data elements, and system states into subsets implies some basic capabilities which are required within the enforcement process. Each authorization specification of the form (3.3) names a group $U$ of $U$, a unit $R$ of $R$, and a subset $S$ of $S$. A sequence of authorization
specifications then names the collections of subsets, $\bar{U}$ and $\bar{R}$ of (4.2) and (4.3).

The implications of user grouping are considered first. Since each of the authorization specifications indicates that every member of its group is given certain access rights, an individual in two different groups is given the rights of one group by virtue of being a member of this group and also is given the rights of the other group by being a member of that group. In other words, a user in two groups may gain access via either the access conditions for one group or the access conditions of the other. Thus, the access condition for a single user in two different user groups, with respect to a given resource and data operation, is the logical OR of the access conditions taken for each group alone. The enforcement process must, therefore, have the capability to determine user group membership and to obtain the OR of access conditions over these user groups (for each resource involved). In summary, if a particular user is a member of several specified groups of $U$, he must then have the access rights of each of the groups.

Implications of Overlapping Resource Units—an AND Operation

The grouping of resources into units also has implications. The definition of authorization specification (3.3) imposes necessary conditions for access, but does not constitute sufficient conditions. A later specification naming data elements common to the resource units in a previous authorization will have the effect of requiring more stringent provisions for access to those common elements. The first specification requires that certain conditions be met and the later specification requires that other conditions must be met for access to the same data elements. Since all conditions with respect to a given resource unit must be met before access to any of its data elements is allowed, the new access condition is the logical AND of all the contributing access conditions. As previously observed, this conjunction is represented by a smaller region in the security space than is any of the original conditions. Thus, the enforcement process must also have the capability to determine which authorized resources overlap the requested resource and to compute the new access conditions for these resource units (for each
given, fixed $U$ and $e$ involved).

Some care is required for cases in which some elements of the requested resource units are not included in any of the overlapping authorized resource units. It is possible, for example, for the authorized resources to intersect part of a requested resource, but not all of it. The part not intersected with is not, of course, authorized for access. Therefore, the enforcement process must also check for covering of the request; i.e., it must ensure that every data element requested is included by the resource unit of some authorization specification used in the access decision.

Interpretations of the Enforcement Process

At this point, one may question whether the choice of the AND operation of access conditions over resource units and the OR operation of access conditions over user groups affects the generality of the model. The choice of these operations is a matter of highest level policy. The AND and OR operations were not arbitrarily chosen; they are necessary to satisfy the definition of authorization specification in (3.3). However, since that definition is, in effect, an interpretation and, therefore, represents one view of reality, it might be worthwhile to allow for other views. The answer is that having these operations as a part of the enforcement process does not create a fixed interpretation; there are still enough degrees of freedom to allow different interpretations. One approach to accommodate a different interpretation is at the authorization specification level. In other words, the occasional authorizer who wants a different definition can use the same enforcement process by constructing specifications in such a way as to have the desired effect. For example, consider a special case in which the user $u$ is in groups $U_1$ and $U_2$ and an AND operation (instead of the OR operations which is a part of the model) is required for user grouping. The solution is to enter a separate specification for $u$ with an access condition which is the AND of the access conditions for $U_1$ and $U_2$. The problem is thereby easily solved without modifying the basic operations of the enforcement process.

In general, there are two ways to vary the interpretation:
1) Use an AND operation of access conditions for user groups

2) Use an OR operation of access conditions for resource units.

The first case, as used in the above example, has these implications: membership in one group will reduce rights with respect to another group, and loss of membership in a group generally increases access rights with respect to another group. Although these effects appear to be counter-intuitive, they may be meaningful in some contexts. However, for most situations the real answer is: if a user should not have the rights of a particular group, do not put him in that group.

The second case, 2) above, has this implication: membership of a data element in more and more units will cause the effective access condition of that element to be progressively more permissive. This has the undesirable property that only isolated data elements can enjoy adequate protection.

On the other hand there are some compelling reasons for the proposed arrangement:

1) The OR operation over user groups allows the facility of assigning an overall minimum level to all users with a single authorization to the GENERAL user group.

2) The AND operation over resources gives a non-subvertible authority to an owner of a resource with shared ownership. Each other owner can only make access conditions more restrictive, and can never defeat the intent of an owner by making the conditions less restrictive.

3) In cases where extra precautions are necessary, complete assurance that access to a given resource is blocked is achieved by a constant "false" access condition in conjunction with the AND operation over the resources.

4) A convenient way to refer to exceptions is made possible by the AND operation. Suppose R2 ⊂ R1 and access is to be granted to all of R1 except R2. This can be done without specifying the elements of R1-R2. Access is authorized in p1 for R1. Then a specification p2, containing a constant "false" access condition, is issued for R2. All requests for R1-R2 will involve only p1 in the access decision. But requests within R2 will
be denied on the basis of the AND of conditions for R1 and the "false" condition for R2.

The question of these logical operations in the enforcement process is probably the most borderline case of policy or mechanism in the entire model. All systems and models of security must face these basic decisions, but few of them specifically draw attention to the decisions or discuss the alternatives. For example, the protection mechanism of MULTICS follows the convention of using the first applicable access list entry for enforcement [SALTJ74]. By sorting the access lists, specific users appear before general groups of users, allowing exceptions for individuals to be made with respect to group privileges. This process is a built-in logical operation in disguise.

In cases where a different interpretation of the enforcement process is consistently required, one choice of logical operations can be made as well as any other choice. The model is inherently capable of accommodating either policy. In fact, the semantic specification statements, yet to be introduced (in chapter V), explicitly allow for this choice. The important point is that in no specific case can there be a requirement for more than one choice of logical operations. This would be tantamount to having more than one definition of enforcement—a situation of internal inconsistency.

In this presentation, the AND operation will be used for grouping data elements and OR operations will be used for grouping users in an enforcement process.

The Enforcement Process as a Series of Steps

The term "franchise" is used in the rest of this chapter and in some language statements of chapter VI. A franchise is a set of authorization specifications with some common characteristic; i.e., a franchise is a set of 5-tuples grouped together for a particular purpose. For example, the franchise of a user, u, is:

\[ F(u) = \{ p \in P : u \in U(p) \} \]

That is, the franchise of a user is the set of authorization 5-tuples which are relevant to him in that they are the only authorizations which refer to the user groups of which he is
a member. In other words, no other authorizations will ever be used to control his requested accesses. In effect, his franchise is a set of authorization 5-tuples which, when taken as a whole, represent his net access privileges (his franchise) in the system. Any grouping of authorizations by a common characteristic of one of the elements of the 5-tuple will be called a franchise.

A second term which may benefit from clarification is "domain." The access decision, $d(q)$, has been defined as a function which maps access requests into (allow, deny). The basis for those decisions is the access control information in $\mathcal{P}$. Therefore, it can be said that the access decision maps:

$$q \times \mathcal{P} \rightarrow \{\text{allow, deny}\}$$

But, since the enforcement process uses only part of $\mathcal{P}$ for each request, the domain of $d(q)$ can be stated more restrictively. Namely, since each decision is in response to a given request, $q$, only the authorizations of $\mathcal{P}$ which are relevant to $q$ need to be in the domain of $d(q)$. Those authorizations germane to $q$ are the set of authorizations which are, at the same time, pertinent to the requesting user, $u(q)$; to the requested operation, $e(q)$; and to the requested resource, $R(q)$. Therefore, it will be seen in the steps that follow that the domain of $d(q)$ can be limited to the authorizations common to the franchise of the user, $F(u)$; the franchise of the operation, $F(e)$; and the franchise of the resource, $F(R)$. (These franchises are defined precisely in appendix D.) This intersection of franchises will be called the domain of authorization, since it is the domain of the function, $d(q)$, and it is a set of authorizations. By the definition of event in (3.5), the number of p-events in an access decision equals the cardinality of the domain of authorization.

For the sake of clarity, the enforcement process will be given first for the basic memoryless case without auxiliary programs or extended resources. Then the case for full authorizations will be given in the same form with the extra steps inserted. A third, more formalized, version is given in appendix D to define the set operations in each step more precisely. Each case is in the context of a request $q$, where $q = (u,e,R,s)$, a set $U$ of unique user groups, a set $R$ of unique resource units and the net collection $\mathcal{P}$ of valid authorizations specifications, and the outcome of the access decision, $d(q)$, is computed.
Basic Memoryless Case

For the basic memoryless case, the enforcement process takes the following steps for a given request \( q \):

1) Determine from the set of known user groups, \( \bar{U} \), those user groups to which the requesting user, \( u(q) \), belongs. Then collect from the set of all authorizations received to date, \( P \), those authorizations which have specified the thus determined user groups as elements. This set of authorizations is called \( F(u) \), the franchise of the user, \( u(q) \).

2) Determine from \( P \) the set of authorizations which have specified \( e(q) \) as an elemental operation. This set is called the franchise of the operation, \( F(e) \).

3) Determine from \( P \) the set \( F(R) \), the franchise of the resource unit, \( R \), the authorizations which name resource units from \( R \) having elements in common with the requested unit, \( R(q) \).

4) Compute \( D(q) \), the domain of authorization for request \( q \), those authorizations common to the three franchises found in steps 1), 2), and 3).

5) Determine the set of all resource units named in elements of \( D(q) \). As these are the resources referenced by the query, this set will be called \( R* \), the resource reference. If the cardinality of the resource reference is one, the request is a simple query; if the cardinality is greater than one, it is a compound query.

6) If any of the resource units in the resource reference are locked, deny the request (perhaps with an explanatory message).

7) Determine if the requested resource, \( R(q) \), is covered by \( R* \); i.e., every element of \( R(q) \) must be contained in some resource unit in some member of \( D(q) \). This first level check means that \( R(q) \) must be a subset of the union of the elements of the resource reference. Failure to have covering here implies that there cannot be the required access rights and access is immediately denied without further checking. (See example 3 at the end of this chapter.) If the request is covered, continue checking. It can be said that the covering check separates the no’s from the maybe’s.
8) Partition the set $D(q)$ into equivalence classes based on the relation such that two authorizations are in the same class if and only if they specify the same resource unit. Compute a single representative authorization for each such class by performing a logical OR over the access conditions of its elements. Each of these classes is for the requesting user $u(q)$. Thus, the new set of authorizations—one per unique resource unit named in $D(q)$—is $F(u,q)$, the effective franchise of the user $u(q)$ with respect to the request $q$.

9) Find the EAC, the effective access condition corresponding to the request $q$, by performing a logical AND on the access conditions of the members of $F(u,q)$. The effect is that this AND is taken over all resource units which overlap the requested resource unit.

10) This step represents the exact moment of the access decision: Evaluate the effective access condition and render an access decision,

$$d(q) = \begin{cases} 1, & \text{to permit access to } r, \text{ if EAC is "true"} \\ 0, & \text{to deny access, if otherwise.} \end{cases}$$

This step itself might require physical accesses to obtain data contents used in access conditions.

11) If $d(q)=1$, complete the logical access operation by passing the data to or from the user.

See appendix D for a formal statement of the basic memoryless enforcement process and appendix G for inclusion of variable semantics in the enforcement process.

**Full Specification Case**

For the more general case of full specifications, the enforcement process is given in the following steps. (The previously discussed steps are somewhat abbreviated.)

1) Determine the franchise of the user, as before. If extended resources are requested, follow all links to find authorizations for the extended resources and include these in $F(u)$, so that the additional access
rights are used for this particular access decision.

2) Determine the franchise of the operation, F(o).

3) Determine the franchise of the resource, F(R).

4) Compute D(q), the domain of authorization, the intersection of the franchises found in 1), 2), and 3).

5) Determine the resource reference, R*, as before.

6) If any elements of the resource reference are locked, deny the request and halt.

7) Determine covering of the request by R*; deny access and halt if the request is not covered.

8) Compute the domain of auxiliary invocation, the set of all auxiliary invocation triples associated with elements of D(q). Invoke all programs for which b[v]=0 and C(s,S[v]) holds. Recall that the results of these procedures can be passed to access conditions via system counters and other system variables.

9) Partition D(q) into equivalence classes based on common resource units and take the OR of access conditions within each class to produce a single effective authorization representing the class. The set of representative authorizations is the effective user franchise, F(u,q).

10) Compute and evaluate the effective access condition by performing a logical AND of the access conditions in F(u,q) and render the corresponding access decision, d(q).

11) If history keeping is requested, increment the specified counters, k, per the flags, b[h].

12) If auxiliary invocations are required, invoke all programs for which b[v]=1 and C(s,S[v]) holds. If the access decision from step 10) above is d(q)=1, execute all auxiliary programs for which b[v]=2 and for which C(s,S[v]) holds. If the decision is d(q)=0, execute all programs having b[v]=3 and for which C(s,S[v]) holds.

13) If d(q)=1, complete the logical access operation by allowing the data base system to perform e(q) on R(q) for u(q).
These steps, when considered separately, imply an enforcement process done on one axis at a time within the security space. Perhaps there is an implementation which allows the enforcement to take place on all axes simultaneously, the process being done in many dimensions at once. For example, steps 1), 2), and 3) can be done at the same time, or at least can be accomplished progressively, i.e., using the results of each step as the starting point (domain) for the next step. Of course, implementations which lack the feature of non-disjoint groupings do not need these functions anyway. It is also emphasized that this is only a conceptual model of enforcement. Various implementations can combine, or rearrange the order of, other steps. For example, it is expected that most implementations will execute steps 1) and 9) at login time, preparing in advance of any requests an effective franchise of a user and taking the OR of access conditions over groups containing the requester.

Partial Rejection by the Enforcement Process

At the beginning of this section the question was raised regarding the case in which the requested resource units are only partly authorized for access. The enforcement process thus far has been based on the approach that, if the requested resource units are not completely contained within the authorized subspace, the entire request is denied. If it is desired to take an alternative approach and allow access to the authorized part without rejecting the user's request, the enforcement process must be modified in the step determining the covering by D(q). At this point a new request:

\[ q = (u, e, R', s) \]  \hspace{1cm} (4.9)

is formulated for the user, where \( R' = R \cap R(q) \) and is the set of those requested data elements which do lie in the authorized subspace. (See example 1 in the next section.)

The next section contains examples which illustrate the authorization and enforcement processes.
Examples of Authorization and Enforcement

In the Security Space

Six examples will be presented in this section. The first three examples are brief sketches illustrating, respectively, full or partial rejection, partitioning for the use of the OR and AND operations, and a request that is not covered. These sketches use Venn diagrams to refer to the sets without discussing individual elements. The last three examples are more detailed and illustrate the steps of the enforcement process.

Example 1. Full vs. Partial Rejection

Suppose that $P = (P_1, P_2)$ where $P_1 = (a, U_1, e, R_1, S_1)$ and $P_2 = (a, U_2, e, R_2, S_2)$. Next, consider a request, $q = (u, e, R, s)$ such that $u(q) \in U_1$, $u(q) \in U_2$, and the relationship among $R_1, R_2, R(q)$ is illustrated in figure IV.1. Recall that, in the previous section, a full rejection policy corresponds to a complete denial response from the enforcement process in case any part of the request cannot be allowed. On the other hand a partial rejection policy will allow authorized parts of a request while rejecting unauthorized parts. In case 1, figure IV.1, the enforcement process requires an effective access condition of:

$$EAC = C(s, S_1) \land C(s, S_2)$$

because the requested resource is contained in the intersection of $R_1$ and $R_2$. Using either policy of rejection, the success of the entire request depends on the EAC. However, in case 2, figure IV.2, while the EAC of the request is the same, the two rejection policies will give different results in some cases. For example, if the evaluation of $C(s, S_1)$ is "true," but the evaluation of $C(s, S_2)$ is not, the full rejection policy will deny the entire request because the EAC="false." But the partial rejection policy will allow access to $(R_1 - R_2) \cap R(q)$, shown as the shaded area, since $C(s, S_1)$ holds.
Figure IV.1 - Full vs. Partial Rejection, Case 1: Request Contained in Intersection

Figure IV.2 - Full vs. Partial Rejection, Case 2: Request Overlapping
Example 2. Partitioning for OR and AND Operations

Consider $P = \{p_1, p_2, p_3\}$, where $p_1 = (a, U_1, e, R_1, S_1)$, $p_2 = (a, U_2, e, R_1, S_2)$, and $p_3 = (a, U_3, e, R_2, S_3)$. Suppose a request $q$ is received such that $u(q) \subseteq U_1$, $u(q) \subseteq U_2$, and $u(q) \subseteq U_3$. Suppose also that the relationship among $R_1$, $R_2$, and $R(q)$ is illustrated in figure IV.3. In this example $D(q) = \{p_1, p_2, p_3\}$ and it is partitioned into two equivalence classes, corresponding to $R_1$ and $R_2$, respectively: $(p_1, p_2)$ and $(p_3)$. The effective franchise of the user, $F(u, q)$ becomes:

$$\{(a, u(q), e(q), R_1, S_1 \cup S_2), (a, u(q), e(q), R_2, S_3)\}.$$ 

The EAC is then:

$$(C(s, S_1) + C(s, S_2)) \& C(s, S_3).$$

Example 3. A Request Not Covered

Let $P = \{p_1, p_2\}$ where $p_1 = (a, U_1, e, R_1, S_1)$ and $p_2 = (a, U_2, e, R_2, S_2)$. $R_1$, $R_2$, and the requested resource, $R(q)$, are shown in figure IV.4. In this example $D(q) = \{p_1, p_2\}$ and the EAC can be calculated to be $C(s, S_1) \& C(s, S_2)$. However, the request must be denied because $R(q)$ is not covered, since $R(q) \notin (R_1 \cup R_2)$.


Consider a system with only 10 elemental resources $r_1, r_2, \ldots, r_{10}$, one authorizer $a$, and one user $u$, for one operation $e$. Suppose $a$ owns all of the resources. Furthermore, resource values are restricted to $0$ and $1$.

Presently, the authorization specification $p_1$ is the only specification in the system:

$$p_1 = (a, u, e, R_1, S_1)$$

where $R_1 = \{r_1, r_2, r_3, r_4\}$ and the condition describing $S_1$ is:
Figure IV.3 - Partitioning for OR and AND Operations

Figure IV.4 - A Request Not Covered
It may be helpful to enumerate the states in $S_1$ on the basis of the condition. Since each resource can take either the resource value 0 or the resource value 1, the number of different states is $2^{16}$. For convenience, only the 16 pertinent resource value patterns will be listed and they will be named states $s_0$ through $s_{15}$ as shown in figure IV.5.

It is obvious that the states which satisfy the condition:

$((v(r_7)=1 & (v(r_8)=0))$

are the state subset:

$S_1 = \{s_8, s_9, s_{10}, s_{11}\}$

The next authorization specification entered is:

$p_2 = (a, u, e, R_2, S_2)$

where $R_2 = \{r_3, r_4, r_5, r_6\}$ and the condition describing $S_2$ is:

$(v(r_7)=0) + ((v(r_9)=1) & ((v(r_8)=0) + (v(r_{10})=1)))$

By referring to figure IV.5, it may be noted that the new condition determines for the set $S_2$ the following states:

$S_2 = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_{10}, s_{11}, s_{15}\}$

The security subspace determined by the first authorization specification $p_1$ is marked with X's and the security subspace determined by the second authorization specification $p_2$, with +'s. Then the two-dimensional projections on the resource and state axes can be depicted in figure IV.6.

This example illustrates the application of expression (4.6). In referring to figure IV.6, those X's corresponding to $(R_1-R_2)$, i.e., corresponding to $(r_1, r_2)$, are affected exclusively by $S_1$ as a result of the first authorization. Similarly, those +'s in figure IV.6 corresponding to $(R_2-R_1)$ or $(r_3, r_6)$ are affected only by the second authorization with $S_2$ as its state subset. On the other hand those X's
### Figure IV.5 - States in Terms of Resource Values

<table>
<thead>
<tr>
<th>Resource Values</th>
<th>( s_0 )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( s_4 )</th>
<th>( s_5 )</th>
<th>( s_6 )</th>
<th>( s_7 )</th>
<th>( s_8 )</th>
<th>( s_9 )</th>
<th>( s_{10} )</th>
<th>( s_{11} )</th>
<th>( s_{12} )</th>
<th>( s_{13} )</th>
<th>( s_{14} )</th>
<th>( s_{15} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{10} )</td>
<td>0 0 1</td>
<td>1 0 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( r_{9} )</td>
<td>0 0 1</td>
<td>1 0 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( r_{8} )</td>
<td>0 0 0</td>
<td>0 1 1</td>
<td>1 1 0</td>
<td>0 0 1</td>
<td>1 1 0</td>
<td>0 0 1</td>
<td>1 1 0</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( r_{7} )</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 1 1</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( r_{6} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( r_{5} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( r_{1} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure IV.6 - The Resource and State Projection of the Security Space

Figure IV.7 - The Final Authorized Subspace
overwritten with +'s, corresponding to \((R_1 \cap R_2)\) or \((r_3,r_4)\) are affected by both conditions \(S_1\) and \(S_2\). Thus, \(S_1 \cap S_2\) is the state subset for this region. (i.e., a smaller subset of states will be acceptable to contain a current state; the access conditions are now more strict.)

The above mentioned three regions constitute the final authorized subspace for the specifications as depicted in figure IV.7.

Next suppose the following access request is made:

\[ q = (u,e,R_3,s_{10}) \]

where \(R_3 = (r_3,r_4)\). The enforcement process is applied. Since \(u\) and \(e\) are fixed in this example, \(F(u)\) and \(F(e)\) are both \((p_1,p_2)\). Since \(R_3\) overlaps both \(R_1\) and \(R_2\), \(F(R_3) = (p_1,p_2)\) and, therefore, the domain of authorization, \(D(q)\), is also \((p_1,p_2)\). Also, \(R\) is covered by \(D(q)\), as \(r_3\) and \(r_4\) are contained in \(R_1\) (and also happen to be in \(R_2\)). The partitioning of \(D(q)\) yields:

\[ F(u,q) = ((a,u,e,R_1,S_1), (a,u,e,R_2,S_2)) \]

No OR was taken here because only a single user is being considered. The effective access condition becomes:

\[ EAC = C(s,S_1) \land C(s,S_2) \]

Recalling, the first condition is:

\[ (v(r_7) = 1) \land (v(r_8) = 0) \]

and the second condition is:

\[ (v(r_7) = 0) + ((v(r_9) = 1) \land ((v(r_8) = 0) + (v(r_{10}) = 1))) \]

The conjunction of the above conditions yields the condition:

\[ EAC = (v(r_7) = 1) \land (v(r_8) = 0) \land (v(r_9) = 1) \]

This newly derived condition can be expressed in terms of a state subset:

\[ EAC = C(s,S_1 \cap S_2) = C(s,(s_{10},s_{11})) \]

The valid subspace authorized for the request is depicted in figure IV.8. Since the request occurs during state \(s_{10}\), the enforcement process will allow access. From the
Example 5. Controlling Access to Multiple Resource Units in a Multi-User Environment

In this example a different projection is studied. In figure IV.9 a security space is illustrated for one authorizer a, a fixed state of the system s1, and a given operation e.

The subspace determined by the following specification is marked with X's:

\[ p_1 = (a, U_1, e, R_1, s_1) \]

where \( U_1 = \{u_1, u_2, u_3, u_4\} \) and \( R_1 = \{r_1, r_2, r_3, r_4\} \). The next subspace is determined by the following authorization and is marked with +'s:

\[ p_2 = (a, U_2, e, R_2, s_1) \]

where \( U_2 = \{u_3, u_4, u_5, u_6\} \) and \( R_2 = \{r_3, r_4, r_5, r_6\} \). The resulting authorized subspace is also depicted in figure IV.9.

As discussed before, the additional authorization will expand the authorized subspace along the user axis. The overlapping of the two authorizations indicates that two different user groups are involved in the access authorization for common resources \( r_3 \) and \( r_4 \).

Next, suppose the following access request is made:

\[ q = (u_3, e, R_3, s_1) \]

where \( R_3 = \{r_1, r_2, r_3, r_6\} \). Pursuant to the enforcement process, it can be seen that since \( u_3 \in U_1 \) and \( u_3 \in U_2 \), \( F(u) \) is \( (p_1, p_2) \) and \( F(e) \) trivially is \( (p_1, p_2) \). Also, since \( R_3 \cap R_1 = \Phi \) and \( R_3 \cap R_2 = \Phi \), \( F(R_3) = (p_1, p_2) \) and the domain of authorization also is:

\[ D(q) = (p_1, p_2) \]

and \( R_3 \) is covered. The partitioning produces:
PROCESSES OF THE MODEL
EXAMPLES OF AUTHORIZATION AND ENFORCEMENT

Figure IV.8 - The Subspace for a User Request

Figure IV.9 - Projection of Security Space for Fixed State, Multi-User Case
PROCESSES OF THE MODEL
EXAMPLES OF AUTHORIZATION AND ENFORCEMENT

\[ F(u,q) = \{ (a,u,e,R_1,s_1), (a,u,e,R_2,s_1) \} \]

The EAC = C(s,s_1), and since the state s_1 is fixed in this example, the request is allowed. The entire requested security subspace is contained in the authorized subspace as depicted in figure IV.9.

Example 6. Conditional Access Involving Multiple Authorizations

This example illustrates the enforcement process on multiple authorization specifications. These specifications are listed in \( \mathcal{P} \). It is posited that, as a result of previous validation, invalid specifications have been removed. With this assumption, there is no need to represent various authorizers. For the purpose of discussion, a single authorizer \( a \) will be employed. Further, discussion is restricted to a single operation \( e \).

\[ \mathcal{P} = \{ (a,U_1,e,R_2,S_1), (a,U_3,e,R_1,S_7), (a,U_1,e,R_3,S_2), (a,U_3,e,R_3,S_8), (a,U_1,e,R_5,S_3), (a,U_3,e,R_5,S_9), (a,U_2,e,R_2,S_4), (a,U_4,e,R_1,S_9), (a,U_2,e,R_3,S_5), (a,U_4,e,R_2,S_5), (a,U_2,e,R_4,S_6), (a,U_4,e,R_3,S_10) \} \]

It may be noted that the state subsets in two of the above specifications are given as \( S \), indicating access to resources is unconditional. The access conditions are tabulated as entries in a matrix of user groups and resource units in figure IV.10.

Suppose that a request, \( q = (u,e,R,s) \), is received such that \( u \in U_1 \), and \( u \in U_4 \). Also assume that it is determined that \( R \) has elements in common with \( R_1 \) and \( R_3 \) and that \( R \) is covered by \( R_1 \) and \( R_3 \). Since the requested operation \( e \) is the same as the authorized operation, it will not be discussed. The authorized conditions under which the user \( u \) can make accesses are represented by the first and fourth rows in figure IV.10.

More specifically, the collection of the access conditions for user, \( u \), is characterized by the following
Fixed Parameters: \( a, c \)

Resource Subsets

\[
\begin{array}{cccccc}
R_1 & R_2 & R_3 & R_4 & R_5 \\
0 & c_1 & c_2 & 0 & c_3 \\
0 & c_4 & c_5 & c_6 & 0 \\
c_7 & 0 & c_8 & 0 & 1 \\
c_9 & 1 & c_{10} & 0 & c_{11} \\
\end{array}
\]

\( F(R) \)

\( F(u) \)

Figure IV.10 - Access Conditions
set of specifications:

\[ F(u) = \{(a, U_1, e, R_2, S_1), (a, U_4, e, R_1, S_9), (a, U_1, e, R_3, S_2), (a, U_4, e, R_2, S), (a, U_1, e, R_5, S_3), (a, U_4, e, R_3, S_{10}), (a, U_4, e, R_5, S_{11})\} \]

In terms of the specific request, the user restricts his access to resources \( R_1 \) and \( R_3 \). The collection of conditions imposed on the resources \( R_1 \) and \( R_3 \) are the first and third columns of the matrix in figure IV.10. The set of specifications which characterize these columns is as follows:

\[ F(r) = \{(a, U_3, e, R_1, S_7), (a, U_2, e, R_3, S_5), (a, U_4, e, R_1, S_9), (a, U_3, e, R_3, S_8), (a, U_1, e, R_3, S_2), (a, U_4, e, R_3, S_{10})\} \]

In terms of the terminology introduced in the section on the modelling of the enforcement process in this chapter, the franchise of the operation \( e \) is \( F(e) = \mathbb{P} \), and the domain of the authorization is:

\[
F_D(q) = F(u) \cap F(e) \cap F(R)
= \{(a, U_1, e, R_3, S_2), (a, U_4, e, R_1, S_9), (a, U_4, e, R_3, S_{10})\}
\]

The effective franchise of the user in this example is:

\[ F(u, q) = \{(a, u, e, R_1, S_9), (a, u, e, R_3, S_{10})\} \]

The effective access condition is:

\[
EAC = C(s, S_9 \cap (S_2 \cup S_{10}))
= C(s, S_9) \land (C(s, S_2) \lor C(s, S_{10}))
\]

This is the expression to be evaluated by the enforcement process to yield an access decision.
CHAPTER V

THE NATURE OF PROTECTION LANGUAGES

The language ranks highest which goes furthest in the art of accomplishing much with little means, or in other words, which is able to express the greatest amount of meaning with the simplest mechanism.

---Otto Jespersen

Introduction

Protection languages are the key to the expression of policies and rules of access by authorizers (see figure II.1, chapter II). The question of how to transmit the necessary information to access control mechanisms has been a subject almost totally neglected. To overcome this neglect is the justification of this dissertation. The present chapter deals explicitly with the heart of that neglected subject: languages for protection and authorization in access control.

In most early systems it was assumed that one individual will hold all authority and all ownership [HOFFL73a]. In such systems only the security officer or data base administrator can make authorizations allowing others to share the use of the data. The system was built around the central power of the SYSADMIN. Typically, the SYSADMIN had but crude methods to enter access control information. Often it was a case of maintaining tables of users, passwords, and a minimal representation of access rights. Usually such tables were updated only in a batch, off-line mode by rigid and awkward procedures. Recently,
however, some researchers are realizing that, due to vastly increasing numbers of users, terminals, programs, and files, a single central SYSADMIN loses the ability to make proper authorizations [GLADH74]. Some designers are aware that the SYSADMIN can be a serious bottleneck [SALTJ74]. Thus, in many systems it is no longer practical, or maybe not even safe, to concentrate in one place all of the necessary authority, responsibility, and knowledge for authorization decisions. Thus, a move toward decentralization of authority is in evidence. This results in different levels of authorizers, operating at many levels of sophistication. The task of authorizing the use of system resources is now shared. Accompanying this move is a pattern of increased complexity in protection requirements. These go beyond the very primitive protection requirements accepted by most current systems. Therefore, a need is evident for better ways to enter the protection requirements of these authorizers. The answer proposed here is a family of protection languages, aimed at the various levels of authorizer sophistication which have evolved. These languages will allow authorizers to precisely define aggregates of users and resources. They also will allow refinement of the conditions relating to access and specification of further protective measures, such as audit trails and authentication procedures. Even in cases where authority is to remain centralized, the increasingly complex security requirements demand a wide scope of protection languages and corresponding access control mechanisms.

In this research a model was developed to characterize the capabilities of protection languages. In this chapter a set of protection language constructs is presented, and the capability of the language constructs to realize the model is analyzed.

The Nature of Protection Languages

According to the terminology of programming languages [SAMMJ69], protection languages fall into the classification of specialized languages and non-procedural languages, translated in an interpretive mode. A special purpose language is a language designed to satisfy one specific objective. Such specialized languages require, on the part of the user, a knowledge of a particular technical discipline. Surely, this is true of protection languages.
The users of these languages, the authorizers, must understand the effects of their authorizations in terms of how their rules and policies will be enforced by the processes. Specialized languages also usually contain terminology related to the corresponding discipline. Protection language statements will often contain terminology such as CONFER, DIVEST, and KEEP HISTORY, which hint at their corresponding authorization functions.

The non-procedural nature of protection languages derives mostly from the fact that protection languages are for specifications and declarations and not for programming. Therefore, one does not see results that look like programs, or at least not very complicated programs. Rather, one mostly sees sequences of non-procedural statements. These are mainly for defining sets of objects and relations among them, with little or no constructs for control of flow. The only structural requirement is that certain definitions appear before their use in operational statements.

Many programming languages have some non-procedural characteristics, such as the matrix operations of OMNITAB [HILSJ66], the algebraic manipulations of MATHLAB [ENGEC65], and the mathematical functions of FORTRAN and PL/I. There are many other examples of non-procedural languages, most of which are also special-purpose, for surveying, numerical machine control, design automation, job control, etc. Thus, there are many kinds of languages associated with computers that are not procedural programming languages, per se. "Programs" in most of these languages are largely sets of definitions of the entities used in the associated disciplines and a few operations which connect them together in some way. Protection languages are also of this nature. Another reason why one will not see large blocks of code in protection languages can be seen from the point of view of security. It is safer to execute statements one at a time in an interactive, or at least interpretive, mode in order that the authorizer can observe the effects of each individual statement. Bugs or errors introduced in the use of protection languages can be far more damaging than those introduced in the use of programming languages.
Translation of Protection Languages

The translation of protection languages can be divided into a lexical part and a semantic part.

Syntax and Lexical Translation

The central point of this dissertation is a family of protection languages built on different levels of user sophistication. This means different levels of syntax as well as of semantics. The importance of this chapter is in the semantic aspects of protection languages. However, it is difficult to discuss languages, and impossible to write down expressions of languages, without some attention to syntax. The syntax is, of course, also important to the user; it is his instrument of access to the semantics of protection constructs.

There is a spectrum upon which the general types of syntax of protection (or any kind of) languages may be placed. This spectrum exhibits a tradeoff between succinctness and naturalness. The characteristic of naturalness is the key to ordering syntaxes for levels of sophistication. To explain, let the spectrum be arbitrarily divided into three broad, overlapping categories. At the extreme of naturalness and ease of use is found the interactive conversational type of input. This is often called a "menu-driven" input because it consists of presenting the user, at each point in his development of the input, a choice from a "menu" of constructs and sub-constructs. He builds his input from sequences of these choices, and by typing in names wherever identifiers are needed. Virtually no knowledge of the syntax is required, the user is prompted at every step; the syntax is built into the interactive prompting scheme. Because of the way definitions and other statements are built up, such interactive syntax is highly suitable for protection languages, especially for uses of lower levels of sophistication.

The next level of syntax sophistication is in a style like that of the COBOL programming language. Here the syntax is very wordy, somewhat English-like, and almost as natural as the interactive syntax. Many extra words may be
used in each command to help explain its meaning. The primary difference is that it is user directed, instead of machine directed as in the previous category. The user is now responsible for generating the syntactic expressions.

A third possible level of syntax is a very terse command language of the type often associated with the SYSADMIN. His language is often succinct but not natural. The short statements, perhaps similar to those found in a typical job control language, express some powerful constructs, but are difficult to understand by the casual authorizer. Much like the skillful use of a job control language, the use of such a high level language may be esoteric to a select few, highly sophisticated authorizers.

Since the various forms of syntax are just different ways to express the same constructs, these forms can be transformed by lexical transformations into a single standard form—the tuples and other entities of the model. The diagram in figure V.1 depicts input specifications in a number of different protection languages (PL-1, PL-2, etc.) having various forms of syntax. If each language has its own lexical translator (LT-1, LT-2, etc.), each language statement can be transformed into a common form. After that point the process is independent of syntax, for all languages in the family.

Thus, it can be seen that very different forms of syntax can be used to represent the same constructs, even for the same level of user. Any one construct can vary greatly in its embodiment. For convenience, only one representative syntax will be selected to illustrate language constructs in the rest of this chapter. For purposes of illustration the choice will be a syntax typified by the middle category—wordy and somewhat natural. The interactive type of syntax, intended for presentation on a terminal in a real time situation, is not as suitable for use in this text. Also, the very terse syntax is not as easy to understand as the more wordy syntax. Examples of these other forms of syntax are found in [WONGD75].

Semantic Translation

An access control system regulates the use of resources by various users under differing conditions. The details on how this regulation is carried out are communicated to the
systems by the semantics of protection language statements. A thorough understanding of the semantics is prerequisite for the accurate composition of protection language statements and effective protection policies.

The semantics of a protection language statement is the meaning behind its construct. This meaning is often best described in terms of the total effect of the statement. In a recent work [ELLID74] in which he studied the semantics of data structures in programming languages, Ellis used the base language model of Dennis [DENN72] as a semantic foundation. Similarly, the set oriented model developed in chapters II through IV provides the semantic basis for work on protection languages here.

What is the semantic translation shown in figure V.1? Is it similar to a compiler of a high level programming language? Figure V.2 illustrates the use of a compiler. In this arrangement the bulk of the translation is carried out by the compiler. The composite effect of the compiler, the execution monitor, and the assembly language or machine language instruction set serves to interpret the semantics carried in the program and data. Such a viewpoint can be called an operational model of semantics in that the changing states of the system are modelled. The effects of protection language statements can be seen by observing the access control system at various moments during its operation. This view (as an overall effect) of semantics, however, is not generally taken regarding compilers. As pointed out in [ELLID74], a model of the changing states, as a language statement is processed, is essentially a model of an interpreter for those language statements. Therefore, this kind of model is termed an interpretive semantic model. The Vienna Definition Language [WEGN72], an interpreter-writing metalanguage, is a well known interpretive semantic model. An analogy with interpreters of programming languages may, then, be more appropriate than one with compilers.

An interpretive system is shown in figure V.3. In this scheme the syntactic transformer plays a less dominant role than the compiler of figure V.2. A simpler transformation is used to arrive at the internal form. Then, possibly at a later time, the interpreter directs the execution, producing the results.

Next, consider protection language translation, as shown in figure V.4. This entire diagram is contained in the "semantic translation" box of figure V.1. That is, the input protection language here is assumed to be in the
THE NATURE OF PROTECTION LANGUAGES
TRANSLATION OF PROTECTION LANGUAGES

Figure V.1 - Lexical Translation

Figure V.2 - Programming Language Compiler
standard form of the model (tuples, etc.). The authorization process, after performing validation of ownership, enters the authorizations into the data structures for the access control information (see also (5), (6), and (7) of figure II.1, in chapter II).

There are close similarities between the interpretation of programming languages in figure V.3 and the translation or protection languages in figure V.4. In both cases a relatively minor transformation is performed to produce an internal representation and the purpose of the interpreter in one case, and the enforcement process in the other case, is to manipulate and control data based on what is contained in the internal representation.

It is this interpretive model which illustrates that the semantics of protection languages are manifested in the combined effect of the authorization process and the enforcement process, but mostly in the execution of the enforcement process. The security space, developed in chapters III and IV and illustrated in appendix F, is an abstract way to conceptualize this joint semantic effect. The security space is not tied directly to either the authorization or the enforcement process. The intimate relationship of the semantics with these processes is the reason why the model for protection languages must be so elaborately considered; the complete translation of protection languages is embraced by the combination of the transformations to standard form and the processes of the model.

Perhaps the concepts of this section can be summarized by a comparison with the similar concepts regarding programming languages, as found in [DENNJ72]. Figure V.5 is Dennis' view of these concepts. Note the very close similarity with figure V.1, in which the same concepts are illustrated for protection languages. The "translators" for the programming languages correspond to the lexical translators for protection languages. The concrete programs have a counterpart in in the protection language statements in their original syntactic form. The abstract programs in the base language are reflected in the standard form (the entities of the model). In fact, the tuples and other entities of the model are a base language for protection languages! The authorization process and the enforcement process interpret the semantics of that base protection language.
Figure V.3 - Programming Language Interpreter

Figure V.4 - Protection Language Translation
Figure V.5 - Programming Language Definitions in Terms of A Common Base Language (from [DENNJ72])
Programming languages provide the means for a programmer to encode expressions for constructs which he chooses to use. Examples of programming language constructs include assignment, control of flow, data structures, and input/output. The study of programming languages has sometimes been hindered by the fact that each language construct can take on quite different appearances when expressed in the forms of syntax found in different programming languages. There is also some variation in the semantics of the constructs as one goes from language to language. Because of these problems Ledgard [LEDGH71] devised ten "mini-programming languages," among which the common programming language constructs were divided. Each mini-language, while perhaps not suitable for programming by itself, has the advantage that its description is brief and its salient features are isolated for study. Later this same approach was used in [ELLID74] to provide (via a series of mini-languages) a semantic basis for data structures.

Similarly, in protection languages it will be very useful to discuss the constructs before a family of protection languages can be developed. The mini-language approach is found to be well suited for this task. Sets of related protection language constructs, corresponding to existing or proposed features of access control, can be compartmentalized into protection language components (PLC's). They are so named to emphasize that they are not protection languages, but the building blocks from which protection languages will be fabricated. This allows the critical issues to be underscored without the clutter of detail often found in a complete language description. Several PLC's can be formed by making several of these groupings of related features. An attempt will be made to order these protection language components by their protection constructs (in terms of levels of sophistication). This ordering is done with regard to semantics, independently of the syntactic structure. It will be shown how some of these features can be selected for use in protection languages.
Categories of Constructs

As in programming languages, certain categories of constructs or general types of statements can be found in protection languages. Basic authorization specifications address the questions of who can use the system, what resources they may access during their use of the system, what data operations are permitted, and under what conditions may these operations be performed. These basic questions are answered by the category called definitional statements. In this category the answers lie, respectively, in individual user definitions and user group definitions, resource unit definitions, elemental operation definitions, and condition definitions. As these are definitions of sets, they may be expressed explicitly by enumeration of set membership or implicitly as conditions on set membership.

Once these entities are defined by definitional statements they can be used in an active way to determine an operation in the authorization process. Thus, the operational statements tie definitions together in a meaningful way by telling how they are used in the authorization process. This class of statements includes authorizations which confer, divest, or alter user access privileges. It also contains commands calling for display of access control information and commands for temporarily locking and unlocking resource units.

There are a number of additional, more complex, functions ancillary to the main task of the operational statements, but very important to expressing protection measures to complement the operational statements. Two important uses of these advanced protection services statements are access history keeping and the use of extended resources. Also included in this classification are commands to invoke auxiliary programs before, during, and after each access decision is made, to accomplish functions like alarm and recovery, user authentication, and threat monitoring.

The last category of authorization commands is used to determine certain things about how the model is applied in a given implementation. Because of the philosophy of separation of policy and mechanism in this dissertation, many dilemmas and questions discussed at length in current literature remain unresolved within the mechanisms of the model. These issues are debated in the literature often under the subject of "What's the best design philosophy?"
Inevitably, a choice is made and justified and then built, deeply and irrevocably, into the innards of the system (or the model) under discussion. The approach taken here is to spotlight some of these issues, but to attempt solutions within policy, rather than within mechanisms of the model. Thus, the last class of protection language constructs discussed here is for expressing policy at the highest level. Such policies are usually fixed for a given system implementation by specifying parameters in the class of **semantic specification statements**. Since the execution of the enforcement process interprets the semantics of the authorizations, any change to the structure of the enforcement process will have an effect on the semantics. The semantic specification statements determine the structure of the enforcement process, with regard to these highest policy issues. Therefore, such statements represent a kind of **variable semantics**, allowing for some basic choices about how the semantics of all the other protection language statements are to be interpreted in a particular system. (See appendix G for a description of how the semantic parameters can affect the enforcement process.)

The person(s) responsible for specifying those parameters must possess the ultimate in authorizer sophistication. Such a person is a sort of "super-SYSADMIN."
The chief virtue that language can have is clearness, ...
--Galen, On the Natural Faculties

Language is called the garment of thought.
--Carlyle, Sartor Resartus

This chapter is an intuitive and informal discussion of protection language constructs and how each can be translated into entities of the model. Appendix C is a more formal description of a protection language. The syntax in this chapter and in appendix C is adapted from a related master's degree thesis by Vong [VONG75] to which the reader is referred for a more complete language description along with many examples and a discussion of translator implementation. It might be helpful for the reader to glance at appendix C before proceeding with this chapter.

A Sample Protection Language

The syntax of the protection language in appendix C is given in Backus-Naur Form (BNF). This form is very suitable as a reference for forming statements. However, the BNF does not give a clear picture of the overall structure and appearance of the language. Therefore, this chapter will begin with a language specification which will be easier to
read. It is a sentential form produced by performing many of the derivations of the grammar to get most of the important forms of protection statements in this language. The statements are numbered, for reference throughout the chapter.

In this chapter, and in appendix C, the following convention has been adopted. Terms are underlined if they are necessary to distinguish the statement type. These are the required terms. The fixed part of each statement is in upper case. This fixed part is the same for all instances of a given statement, except that non-underlined upper case terms are optional (included by beginners for naturalness, omitted by more experienced users for succinctness). Variable parts are given in lower case. Those parts which are fixed for any one system, but which may vary from system to system are lower case and underlined. This class includes certain reserved terms such as: always, never, general, true, and false. Those variable parts supplied by the authorizer are lower case and not underlined. This last class consists mostly of authorizer-created identifiers and values.

Many of the language statements assign names to entities being defined. It shall be assumed that some means (such as the automatic appending of the authorizer’s identification number as an extension to each of his names) will be used to allow each authorizer to use his own local names without conflicting with those used by other authorizers.

The informal specification of the language follows:

Definitional Language Constructs

(1) **DEFINE USER** BY ASSIGNING TO USER <user-number>, name = <user-name>, project = <project>, dept = <dept>, class = <class>, password = <password>
(2) DEFINE USERS.
BEGIN ASSIGNING
TO USER <user-number-1>, name = <user-name-1>, project = <project-1>, dept = <dept-1>, class = <class-1>, password = <password-1> ;

TO USER <user-number-2>, name = <user-name-2>, project = <project-2>, dept = <dept-2>, class = <class-2>, password = <password-2> ;

.
.
.

TO USER <user-number-n>, name = <user-name-n>, project = <project-n>, dept = <dept-n>, class = <class-n>, password = <password-n> ;
END

(3) DEFINE USERS FROM THE FILE NAMED <filename>

(4) DEFINE THE USER GROUP NAMED <user-group-identifier> TO HAVE THESE MEMBERS: <user-number-1>, <user-number-2>, ..., <user-number-n>

(5) DEFINE THE USER GROUP NAMED <user-group-identifier> BY THE CONDITION NAMED <cond-identifier>

(6) DEFINE THE RESOURCE UNIT NAMED <resource-unit-identifier> BY ASSIGNING file = <filename>, template = <user-tmp-identifier>, keywords = (<keyword-1>, <keyword-2>, ..., <keyword-n>)

(7) DEFINE THE RESOURCE UNIT GROUP NAMED <resource-group-identifier> TO HAVE THESE MEMBERS: <resource-unit-identifier-1>, <resource-unit-identifier-2>, ..., <resource-unit-identifier-n>

(8) DEFINE THE CONDITION NAMED <condition-identifier> FOR A USER GROUP BY <Boolean-expression>

(9) DEFINE THE CONDITION NAMED <condition-identifier> FOR ACCESS BY <Boolean-expression>
(13) **DEFINE THE CONDITION NAMED** <condition-identifier> **FOR**
    **ACCESS BY** true

(11) **DEFINE THE CONDITION NAMED** <cond-identifier> **FOR**
    **ACCESS BY** false

(12) **DEFINE THE CONDITION NAMED** <cond-identifier> **FOR**
    **HISTORY KEEPING BY** <Boolean-expression>

(13) **DEFINE THE CONDITION NAMED** <cond-identifier> **FOR**
    **AUXILIARY PROGRAM INVOCATION** **BY**
    <Boolean-expression>

(14) **DELETE USERS** <user-number-1>, <user-number-2>, ..., <user-number-n>

(15) **DELETE USER GROUPS NAMED** <user-group-identifier-1>,
    <user-group-identifier-2>, ..., <user-group-identifier-n>

(16) **DELETE RESOURCE UNITS** NAMED
    <resource-unit-identifier-1>,
    <resource-unit-identifier-2>, ...
    <resource-unit-identifier-n>

(17) **DELETE RESOURCE UNIT GROUPS** NAMED
    <resource-group-identifier-1>,
    <resource-group-identifier-2>, ...
    <resource-group-identifier-n>

(18) **DELETE USER GROUP DEFINING CONDITIONS** NAMED
    <cond-identifier-1>, <cond-identifier-2>, ..., <cond-identifier-n>

(19) **DELETE ACCESS CONDITIONS** NAMED <cond-identifier-1>,
    <cond-identifier-2>, ..., <cond-identifier-n>

(20) **DELETE HISTORY KEEPING CONDITIONS** NAMED
    <cond-identifier-1>, <cond-identifier-2>, ..., <cond-identifier-n>

(21) **DELETE AUXILIARY PROGRAM INVOCATION CONDITIONS** NAMED
    <cond-identifier-1>, <cond-identifier-2>, ..., <cond-identifier-n>

(22) **TEST USER GROUP DEFINING CONDITION** NAMED
    <cond-identifier>

(23) **TEST ACCESS CONDITION** NAMED <cond-identifier>
(24) TEST HISTORY KEEPING CONDITION NAMED <cond-identifier>

(25) TEST AUXILIARY PROGRAM INVOCATION CONDITION NAMED <cond-identifier>

Operational Constructs

(26) CONFER USER <user-number> WITH THE RIGHT TO <op-list>
THE RESOURCE UNIT NAMED <resource-unit-identifier>
ON THE CONDITION NAMED <cond-identifier>

(27) CONFER USER <user-number> WITH THE RIGHT TO <op-list>
THE RESOURCE UNIT NAMED <resource-unit-identifier>
always

(28) CONFER USER <user-number> WITH THE RIGHT TO <op-list>
THE RESOURCE UNIT NAMED <resource-unit-identifier>
never

(29) CONFER USER <user-number> WITH THE RIGHT TO <op-list>
THE RESOURCE UNIT GROUP NAMED <resource-group-identifier> always

(31) CONFER THE SYSTEM PROCEDURE NAMED <sys-proc> WITH THE
RIGHT TO <op-list> THE RESOURCE UNIT NAMED <resource-unit-identifier> always

(32) CONFER THE USER GROUP NAMED general WITH THE RIGHT TO
<op-list> THE RESOURCE UNIT NAMED <resource-unit-identifier> always

(33) CONFER THE USER GROUP NAMED <user-group-identifier> WITH THE RIGHT TO
<op-list> THE RESOURCE UNIT NAMED <resource-unit-identifier> always AND CALL
THIS FRANCHISE <franchise-identifier>

(34) DIVEST THE USER GROUP NAMED <user-group-identifier> OF
THE RIGHT TO <op-list> THE RESOURCE UNIT NAMED <resource-unit-identifier>
(35) **DIVEST THE FRANCHISE** NAMED <franchise-identifier>

(36) **ASSIGN THE NAME** <franchise-identifier> **TO THE FRANCHISE** OF THE **USER GROUP** NAMED <user-group-identifier> TO <op-list> **THE RESOURCE UNIT** NAMED <resource-unit-identifier>

(The above statement can also have all the forms corresponding to those shown for CONFER.)

(37) **ALTER USER** BY ASSIGNING TO USER <user-number> **name =** <user-name>, **project =** <project>, **dept =** <dept>, **class =** <class>, **password =** <password>

(38) **ALTER USERS,**

BEGIN ASSIGNING:

TO USER <user-number-1>, <user-name-1>, <project-1>, <dept-1>, <class-1>, <password-1>;

TO USER <user-number-2>, <user-name-2>, <project-2>, <dept-2>, <class-2>, <password-2>;

TO USER <user-number-n>, <user-name-n>, <project-n>, <dept-n>, <class-n>, <password-n>;

END

(39) **ALTER USERS FROM THE** FILE NAMED <filename>

(40) **ALTER THE** USER GROUP NAMED <user-group-identifier> **BY ADDING THESE MEMBERS:** <user-number-1>, <user-number-2>, . . . , <user-number-n>
(41) **ALTER THE USER GROUP NAMED** `<user-group-identifier>` **BY** **REMOVING** **THOSE** **MEMBERS:** `<user-number-1>`, `<user-number-2>`, ..., `<user-number-n>`

(42) **ALTER THE RESOURCE UNIT NAMED** `<resource-unit-identifier>` **BY** **ASSIGNING** `file = <filename>`, `template = <user-tmp-identifier>`, `keywords = <keyword-1>, <keyword-2>, ..., <keyword-n>`

(43) **ALTER THE RESOURCE UNIT GROUP NAMED** `<resource-group-identifier>` **BY** **ADDING** **THOSE** **MEMBERS:** `<resource-unit-identifier-1>`, `<resource-unit-identifier-2>`, ..., `<resource-unit-identifier-n>`

(44) **ALTER THE RESOURCE UNIT GROUP NAMED** `<resource-group-identifier>` **BY** **REMOVING** **THOSE** **MEMBERS:** `<resource-unit-identifier-1>`, `<resource-unit-identifier-2>`, ..., `<resource-unit-identifier-n>`

(45) **ALTER THE FRANCHISE NAMED** `<franchise-identifier>` **BY** **REPLACING** **EXISTING** **OPERATIONS** **WITH** `<op-list>`

(46) **ALTER THE FRANCHISE NAMED** `<franchise-identifier>` **BY** **REPLACING** **EXISTING** **ACCESS CONDITIONS** **NAMED** `<cond-identifier-1>`, `<cond-identifier-2>`, ..., `<cond-identifier-n>` **WITH** `<cond-identifier-n+1>`, `<cond-identifier-n+2>`, ..., `<cond-identifier-n+m>`

(47) **ALTER THE FRANCHISE NAMED** `<franchise-identifier>` **BY** **REPLACING** **EXISTING** **OPERATIONS** **WITH** `<op-list>` **AND** **ACCESS CONDITIONS** **NAMED** `<cond-identifier-1>`, `<cond-identifier-2>`, ..., `<cond-identifier-n>` **WITH** `<cond-identifier-n+1>`, `<cond-identifier-n+2>`, ..., `<cond-identifier-n+m>`

(48) **ALTER THE FRANCHISE OF THE USER GROUP NAMED** `<user-group-identifier>` **FOR THE RESOURCE UNIT NAMED** `<resource-unit-identifier>` **BY** **REPLACING** **EXISTING** **OPERATIONS** **WITH** `<op-list>`

(The above statement also can refer to individual users, system procedures, the general class of users, and resource unit groups. It can also alter access conditions and a combination of operations and access conditions.)
DISPLAY DEFINITION OF USER <user-number>

DISPLAY DEFINITION OF THE USER GROUP NAMED <user-group-identifier>

DISPLAY DEFINITION OF THE RESOURCE UNIT NAMED <resource-unit-identifier>

DISPLAY DEFINITION OF THE RESOURCE UNIT GROUP NAMED <resource-group-identifier>

DISPLAY DEFINITION OF THE USER GROUP DEFINING CONDITION NAMED <cond-identifier>

DISPLAY DEFINITION OF THE ACCESS CONDITION NAMED <cond-identifier>

DISPLAY DEFINITION OF THE HISTORY KEEPING CONDITION NAMED <cond-identifier>

DISPLAY DEFINITION OF THE AUXILIARY PROGRAM INVOCATION CONDITION NAMED <cond-identifier>

DISPLAY THE FRANCHISE OF USER <user-number>

DISPLAY THE FRANCHISE OF THE USER GROUP NAMED <user-group-identifier>

DISPLAY THE FRANCHISE OF THE SYSTEM PROCEDURE NAMED <sys-proc>

DISPLAY THE FRANCHISE OF THE USER GROUP NAMED general

DISPLAY THE FRANCHISE NAMED <franchise-identifier>

DISPLAY THE FRANCHISES REFERRING TO THE RESOURCE UNIT NAMED <resource-unit-identifier>

DISPLAY THE FRANCHISE REFERRING TO THE RESOURCE UNIT GROUP NAMED <resource-group-identifier>

DISPLAY THE FRANCHISE OF USER <user-number> TO THE RESOURCE UNIT NAMED <resource-unit-identifier>

DISPLAY THE FRANCHISE OF THE USER GROUP NAMED <user-group-identifier> TO THE RESOURCE UNIT NAMED <resource-unit-identifier>

DISPLAY THE FRANCHISE OF THE SYSTEM PROCEDURE NAMED <sys-proc> TO THE RESOURCE UNIT NAMED
PROTECTION LANGUAGES
A SAMPLE PROTECTION LANGUAGE

<resource-unit-identifier>

(67) DISPLAY THE FRANCHISE OF THE USER GROUP NAMED general TO THE RESOURCE UNIT NAMED <resource-unit-identifier>

(68) DISPLAY THE EVENT NAMED <event-identifier>

(69) DISPLAY NAMES OF USER GROUPS

(70) DISPLAY NAMES OF RESOURCES UNITS

(71) DISPLAY NAMES OF RESOURCE UNIT GROUPS

(72) DISPLAY NAMES OF FRANCHISES

(73) DISPLAY NAMES OF EVENTS

Advanced Protection Services

(74) ON THE CONDITION NAMED <cond-identifier> KEEP THE HISTORY OF SUCCESSFUL REQUESTS INVOLVING THE FRANCHISE NAMED <franchise-identifier> AND CALL THIS EVENT <event-identifier>

(75) always KEEP THE HISTORY OF UNSUCCESSFUL ATTEMPTS INVOLVING THE FRANCHISE NAMED <franchise-identifier> AND CALL THIS EVENT <event-identifier>

(76) always KEEP THE HISTORY OF REQUESTS REGARDLESS OF OUTCOME INVOLVING THE FRANCHISE NAMED <franchise-identifier> AND CALL THIS EVENT <event-identifier>

(77) ON THE CONDITION NAMED <cond-identifier> RESET THE HISTORY OF SUCCESSFUL REQUESTS INVOLVING THE FRANCHISE NAMED <franchise-identifier> AND CALL THIS EVENT <event-identifier>

(The above statement also can refer to unsuccessful attempts or events regardless of outcome.)

(78) RESET THE HISTORY OF ACCESSES FOR THE EVENT NAMED <event-identifier>
(79) **INCREMENT** the history of accesses for the event named `<event-identifier>`

(80) **DISCONNECT** the history of accesses involving the franchise named `<franchise-identifier>`

(81) **DISPLAY** the system counter for the event named `<event-identifier>`

(82) **ENTER** the auxiliary program named `<pgm-identifier>` from `<file-identifier>`

(83) **ON THE CONDITION** named `<cond-identifier>` **INVOKE** the auxiliary program named `<pgm-identifier>` **BEFORE** evaluating the access condition of the franchise named `<franchise-identifier>`

(84) **always** **INVOKE** the auxiliary program named `<pgm-identifier>` **REGARDLESS** of the access condition of the franchise named `<franchise-identifier>`

(85) **always** **INVOKE** the auxiliary program named `<pgm-identifier>` **WHEN SATISFYING** the access condition of the franchise named `<franchise-identifier>`

(86) **always** **INVOKE** the auxiliary program named `<pgm-identifier>` **WHEN NOT SATISFYING** the access condition of the franchise named `<franchise-identifier>`

(87) **ON THE CONDITION** named `<cond-identifier>` **INVOKE** the auxiliary program named `<pgm-identifier>` **AT LOGIN**

(88) **DISCONNECT** the auxiliary program named `<pgm-identifier>` from the franchise named `<franchise-identifier>`

(89) **DELETE** the auxiliary program named `<pgm-identifier>`

(90) **TEST** the auxiliary program named `<pgm-identifier>`

(91) **EXTEND** the resource unit of the franchise named `<franchise-identifier>` to the franchise named `<franchise-identifier>`

(92) **REMOVE** the extension of the resource unit in the franchise named `<franchise-identifier>` to the franchise named `<franchise-identifier>`
(93) **DISPLAY THE EXTENSIONS OF THE RESOURCE UNIT IN THE FRANCHISE** NAMED <franchise-identifier>

Semantic Specification Statements

(94) **SELECT SEMANTIC PARAMETERS TO BE:**
   - EP USERS OPERATION = or
   - EP RESOURCES OPERATION = and
   - REJECTION = partial
   - SYSADMIN ON(HS = system files
   - LOCK-UNLOCK = yes
   - EXTENDED RESOURCES = yes
   - EXTENDABLE RESOURCES = <sys-proc-1>, <sys-proc-2>,
     ..., <sys-proc-n>

(The parameters above are examples; other parameters are possible.)

(95) *THIS IS A COMMENT*

The Protection Language Components and Their Translation into Entities of the Model

This section shows how the language statements of each category are translated into entities of the model. It should be noted here that the tuples and sets are still part of an abstract model and the semantics are being interpreted in terms of their effects within this model. This does not imply that an implementation should be designed exactly this way, although any implementation of this model would, of course, be based on the same principles. In most cases, the translation is very straightforward and often almost trivial.
Definitional Constructs

**PLC-1: Simple User Group and Resource Unit Definitions**

This protection language component contains constructs to express simple user group and resource unit definitions. These constructs are often adequate for users requiring little sophistication. Simple user group definitions (4) are used to form a user group by listing the identification numbers of the members. (The numbers in parentheses will be used throughout this chapter to make reference to the language specification in the preceding section.)

\[
\text{DEFINE USER GROUP NAMED group1 TO HAVE THESE MEMBERS:} 1234, 2345, 3456, 6904.
\]

The effect of this enumerative user group definition is:

\[
\text{group1} \in \bar{U} = \{1234, 2345, 3456, 6904\}
\]

where \( \bar{U} \) has been defined as the set of unique user groups having been specified to the system.

Resource unit definitions, as it is pointed out in appendix E, are dependent on the structures and naming schemes of the data base. Attribute-based structures (see appendix E) will be used in examples presented here, using form (6):

\[
\begin{align*}
\text{DEFINE RESOURCE UNIT NAMED profile BY ASSIGNING} \\
\text{file=\textit{filea}, template=ut-3,} \\
\text{keywords=(name=smith, age=40, salary=20000)}
\end{align*}
\]

\[
\begin{align*}
\text{DEFINE RESOURCE UNIT NAMED records BY ASSIGNING} \\
\text{file=\textit{filea}}
\end{align*}
\]

These have the following effects:

\[
\begin{align*}
\text{profile} \in \bar{R} \\
C(r, \text{profile}) : \text{file=\textit{filea}} \& \text{template=ut-3} \\
\& \text{keywords=(name=smith, age=40, salary=20000)}
\end{align*}
\]

\[
\begin{align*}
\text{records} \in \bar{R} \\
C(r, \text{records}) : \text{file=\textit{filea}}
\end{align*}
\]
where \( \bar{R} \) is the set of defined resource units and \( C(r, R) \) is the resource defining condition. Since the descriptions of resource units rarely enumerate each data element in the unit, most such descriptions translate into a resource unit defining condition with a somewhat fixed form (i.e., usually only certain variables are allowed in these conditions). The two resource units just defined overlap each other; in fact, "profile" is a proper subset of "records". The unit named records refers to the entire file without any restrictions on templates or keys.

Any of these definitions can be deleted from the system as follows, using forms (15) and (16):

\[
\text{DELETE USER GROUP NAMED group1}
\]

\[
\text{DELETE RESOURCE UNIT NAMED profile}
\]

The effect of a deletion is to remove the specified element from the appropriate set:

\[
\bar{U} \leftarrow \bar{U} \setminus \text{group1}
\]

\[
\bar{R} \leftarrow \bar{R} \setminus \text{profile}
\]

PLC-1 is also a convenient place to introduce comment statements (95). As in any language, comments are used for documentation.

*This is a comment*

**PLC-2: Definition and Testing of Conditions**

The more general conditions, which play such an important part in defining sets of the model, are created in this component. A condition is a non-procedural representation which becomes translated and stored as a procedural entity with a highly controlled structure. In many implementations a condition will be translated into a short function-type subroutine which, when called, evaluates the expression of the condition and returns a "true" or "false". The conditions are divided into classes, because most translator implementations will benefit from knowing the kind of variables to which each condition will be limited. It is slightly analogous to the problem of static type checking in programming languages. These are conditions used to define user groups (8):
Examples from previous chapters and the appendices are used to illustrate the PLC's. For the condition definitions, these examples are found in the examples of conditions in appendix E. These statements defining conditions do not become manifest in the semantic model at this point. That is due to the fact that the sets, whose memberships are to be determined by these conditions, have not yet been introduced. The authorization processor should do as much as possible to detect errors in the syntax of each statement. Certain other errors normally considered as semantic errors can also be detected at this point. As an example, a variable used in a given type of condition may not be in the class of permissible variable types. The sample syntax in appendix C states that only these variables, giving information about individual users, may be used in user group defining conditions:

\[ \text{id, project, dept, class, name, password} \]

Therefore, if this statement is entered,

```
DEFINE CONDITION NAMED usergrpdef FOR A USER GROUP
BY file=filea & time>12:00
```

the authorization process should recognize that neither variable used is valid for user group definitions. Of course, limitations like this are not inherent in the model, they are typically imposed for convenience and economy of implementation. The conditions for user groups, resource definitions, and access conditions all have an effect on access enforcement. Thus, a limitation on one type of condition may not reduce the number of possible protection requirements which can be represented (as long as the same limitation is not simultaneously imposed on all condition types). Restrictions do, however, restrict the number of different ways any one requirement can be represented.

Other kinds of errors cannot be detected at the time a condition is defined. Therefore, the conditions must be tested for execution errors, such as the use of variables which have not been given values (the "undefined variable"
problem of programming languages). The test statement is for the purpose of testing the execution of condition evaluations. The statement, using any of the forms (22) through (25):

**TEST USER GROUP DEFINING CONDITION NAMED cond-1**

is not a translatable statement. Instead it is a directly executed command to the authorization process. This statement is executed by accessing the appropriate values for substitution in, and evaluation of, the specified condition. If the condition is well defined, the authorizer will be informed whether the condition holds or not, reflecting the current value of the condition. If the condition is ill-defined resulting in execution errors, the authorization process should respond with the appropriate diagnostic message.

Conditions for purposes other than user group definitions are established by very similar statements, forms (9) through (13):

**DEFINE CONDITION NAMED accound FOR ACCESS BY time > 7:00 a.m. & salary < 20000**

**DEFINE CONDITION NAMED checkacc FOR HISTORY KEEPING BY mode=update-cycle**

**DEFINE CONDITION NAMED monitor FOR AUXILIARY PROGRAM INVOCATION BY event13 > 10**

The conditions for history keeping and auxiliary invocation, shown in the last two examples above in the form of (12) and (13), correspond to S[h] and S[v] in (3.6) and (3.7), respectively, of the model. As with other definitions, conditions can be deleted; see (18) through (21):

**DELETE USER GROUP DEFINING CONDITION NAMED cond1**

**PLC-3: Generalized User Group and Resource Unit Group Definitions and Single User Descriptions**

PLC-3 contains definitional constructs which are used by authorizers of higher levels of sophistication. Of these constructs, the use of conditions for generalized user group
definitions (5) is perhaps the most useful.

**DEFINE USER GROUP NAMED designers BY THE CONDITION NAMED cond**

The effect is:

\[
\text{designers} \in \overline{U} \\
C(u, \text{designers}) = \text{cond}
\]

Often it is desirable to define a resource unit by grouping together previously defined resource units, rather than having to restate the definitions of the constituent units. At this level authorizers have this ability to group resource units together to form compound resource units, (7):

**DEFINE RESOURCE UNIT GROUP NAMED bunch TO HAVE THESE MEMBERS: profile, records**

This results in:

\[
bunch \in R \\
bunch = \text{profile} \cup \text{records}
\]

Resource unit groups are deleted from the system using form (17).

Since the description of which users are authorized to even login to the system is an important first line of security, the function of entering these descriptions is often reserved for authorizers having a very high degree of sophistication, such as SYSADMIN. Single individuals are described in this manner, form (1):

**DEFINE USER BY ASSIGNING TO USER 1234**

name=hartson ali k.,
project=mousatrans, dept=5a,
class=a, password=X123akh

Upon processing this statement the authorization process adds the defined user to the set of all users:

\[
1234 \in U
\]

Often, in larger systems, the number of users is great and a blocked statement (2) can be used to describe several individual users at once:
DEFINE USERS,
BEGIN ASSIGNING:

TO USER 1234, name=hartson ali k.,
project=mousetraps, Dept=5a, class=a,
password=X123akh;

TO USER 2345, name=doe jane j.,
project=doorknobs, Dept=5b, class=C,
password= XYZ;

TO USER 3456, name=pike chester q.,
project=admin, Dept=3d, class=a,
password=abcabc;

END

In some systems the list of users may be kept in a file (separate from the access control system) where the properly authorized people can maintain it on-line. To enter user descriptions from this file in a batch, one may use this statement (3):

\[ \text{DEFINE USERS, FROM FILE NAMED user-file} \]

User entries are deleted from the system using form (14).

Operational Constructs

PLC-4: Basic Operational Commands and The Use of Access Conditions

This protection language component is devoted to the operational functions. The first of these, (26) through (33), is the statement used to grant access privileges:

\[ \text{CONFER THE USER GROUP NAMED designers WITH THE RIGHT TO WRITE THE RESOURCE UNIT NAMED profile ON CONDITION accord AND CALL THIS FRANCHISE desrights} \]
The first of these examples is termed a "named CONFER," since it assigns a name to the authorization; see form (33). This name may be useful later to make direct reference to this authorization to, for example, delete it (35) or to associate history keeping (74) or auxiliary programs (83) with it. If an unnamed authorization is later determined to need a name, the name can be added by the ASSIGN statement of form (36). The CONFER statement is also used to assign rights to individual users (26), to system procedures (31), and to the "general" group (32), the group which includes all users. Form (27) demonstrates the unconditional granting of access rights, while form (28) shows unconditional denial of rights. These statements are translated by adding two authorization specifications to the system:

\[ p_1 \in P \]
\[ p_2 \in P \]

where

\[ p_1 = (a, \text{designers}, \text{(write)}, \text{profile}, S_1) \]
\[ p_2 = (a, \text{general}, \text{(read)}, \text{records}, S_2) \]
\[ C(s, S_1) = \text{cond} \]

and where \( p_1 \) is known as "desrights." Similarly, these privileges are removed by (34):

DIVEST THE USER GROUP NAMED designers OF THE RIGHT
TO write THE RESOURCE UNIT NAMED profile

DIVEST THE USER GROUP NAMED general OF THE RIGHT
TO read THE RESOURCE UNIT NAMED records

The obvious effect is:

\[ P \leftarrow P - p_1 \]
\[ P \leftarrow P - p_2 \]

where \( p_1, p_2 \) are defined in the previous example.

The DEFINE, DELETE, CONFER, and DIVEST statements are technically adequate to set the internal representation of access control information to reflect any combination of privileges for users, resources, data operations, and conditions. However, certain changes to existing
definitions and franchises can be conveniently accomplished with an ALTER statement, (37) through (48):

ALTER THE USER GROUP NAMED group1 BY ADDING THESE MEMBERS: 2073, 2943, 8706

ALTER THE FRANCHISE OF THE USER GROUP NAMED designers FOR THE RESOURCE UNIT NAMED profile BY REPLACING EXISTING OPERATIONS WITH read, update

The effect of the first statement is:

group1 = group1 U (2073, 2943, 8706)

The effect of the other statement is:

E(p1) = (read, update)

The ALTER statement can also be used to change definitions of users (37) through (39), resource units (42), and resource unit groups (43) and (44). It may also be used to change named or unnamed franchises by altering the data operation and/or the access conditions, (45) through (48).

The DISPLAY command is another non-executable directive to the authorization process. It provides the indispensable capability to each authorizer of being able to query the internal representation of the access control information. This is done to keep track of his definitions and the franchises he has conferred. The access control information is part of the set of resources and DISPLAY is a data operation. Therefore, an authorizer must have the explicitly conferred right to display any access control information before he can do so. The assignment of display rights is a matter of policy; often a user may display just those items which he himself has entered.

DISPLAY DEFINITION OF USER GROUP NAMED group1
DISPLAY DEFINITION OF RESOURCE UNIT NAMED profile
DISPLAY DEFINITION OF USER GROUP DEFINING CONDITION NAMED cond1

DISPLAY FRANCHISE OF THE USER GROUP NAMED designers FOR THE RESOURCE UNIT NAMED records

Forms (49) through (73) illustrate that there is a very large variety of information which can be displayed,
including all possible definitions, (49) through (56); all franchises of users and user groups, (57) through (61); franchises referring to particular resources or particular user-resource combinations, (62) through (67); history keeping counters or events, (68); and names of various entities, (69) through (73).

Advanced Protection Services

**PLC-5: Access History Keeping and Auxiliary**

**Program Invocation**

This language component contains the statements for controlling access history keeping, (74) through (81), such as illustrated here by form (74):

```
ON THE CONDITION NAMED checkacc KEEP THE HISTORY OF SUCCESSFUL REQUESTS INVOLVING THE FRANCHISE NAMED desrights AND CALL THIS EVENT happening!
```

*desrights is pl, the franchise for designers writing in profile*

The result of this statement is the association of the following triple with the authorization pl:

```
(G(s,S[hl]),k,b[hl])
```

where \(G(s,S[hl])\) is the previously defined history keeping condition named checkacc, and the next available unused system counter k will be assigned to the event named happening!. A value of zero is given to \(b[hl]\) to initiate a response to a success event for which pl is an element of the domain of authorization of the corresponding access decision. (Forms (75) and (76) correspond to \(b[hl]=1\) and \(b[hl]=2\); form (77) covers \(b[hl]=3, 4, \text{ and } 5\); and forms (78) and (79) are for \(b[hl]=6\) and \(b[hl]=7\).) If another user's access rights are then to be made dependant on this event, the name "happening!" must appear in one of that user's access conditions. Let a user group be established based on the defined condition ugc=3:
DEFINE the user group NAMED programmers BY THE
CONDITION NAMED ugc-3.

and a new resource unit called specifications:

DEFINE THE RESOURCE UNIT NAMED specifications BY
ASSIGNING file=file-b, template=ut-7,
keywords=(plans=1a, details=yes, cost=highrange)

Now let the policy be adopted that members of the group
called programmers can write in the resource unit called
specifications—if and only if no one in the designers group
has written in the resource unit called profile. The
franchise desrights will always be involved in an access
decision for designers writing in profile. Policy here
stipulates that the designers must be successful at writing
into profile before recording the event; therefore, the
history of a success event is required. Thus, the history
keeping statement just defined, naming the event as
happening!, is suitable here. The required access condition
for the programmers can now be defined:

DEFINE CONDITION NAMED desevent FOR ACCESS BY
happening! = 0

If the franchise desrights is ever used in a success event,
the counter named happening! will be incremented due to the
history keeping command. The access condition desevent will
then not hold and will thereafter block access through this
operational command:

CONFIR THE USER GROUP NAMED programmers WITH THE
RIGHT TO write THE RESOURCE UNIT NAMED
specifications ON THE CONDITION NAMED
desevent

and the policy is represented. This is translated into
entities of the model as follows:

programmers ∈ U
C(u, programmers) : ugc=3: term1=42

specifications ∈ R
C(r1, specifications) : file=fileb & template=ut-7
& keywords=(plans=1a, details=yes, cost=highrange)
C(s, desevent) : happening! = 0
p = (a, programmers, (write), specifications, desevent)

Suppose now, for example, that the contents of profile
and specifications are later changed in a major update. The
The authorizer may wish to reestablish the history keeping in a way such that the programmers may again write into records, but only until the franchise desrights is again successfully used. This requirement is met merely by resetting the counter, form (78):

```
RESET HISTORY OF ACCESSES FOR THE EVENT NAMED happening
```

The authorizer may at any time see the value of the system counter he is using here by form (81):

```
DISPLAY THE SYSTEM COUNTER FOR THE EVENT NAMED happening
```

When history keeping is no longer required, the counter can be freed by form (80):

```
DISCONNECT THE HISTORY OF ACCESSES INVOLVING THE FRANCHISE NAMED desrights
```

In chapter III the access history keeping section cites several examples. In those examples the translations are shown. The corresponding syntax will be shown in this section. The translations will be repeated to remove the need to continually refer back to chapter III.

The first example illustrated the use of the history keeping counters to perform logical operations on events. The AND function will be shown here. In the case of two existing authorizations called A and B some corresponding events can be defined:

```
always KEEP THE HISTORY OF SUCCESSFUL REQUESTS INVOLVING THE FRANCHISE NAMED a AND CALL THIS EVENT count

always KEEP THE HISTORY OF SUCCESSFUL REQUESTS INVOLVING THE FRANCHISE NAMED b AND CALL THIS EVENT count
```

Notice the use of always to establish unconditional history keeping; see forms (75) and (76). Next, the access condition for authorization C will hold, if both events A and B have occurred:

```
DEFINE CONDITION NAMED notboth FOR ACCESS BY count<2
```
The translations are:

\[
A = (a, U, E, R, S) (S[h]=S, COUNT=0) \\
B = (a, U, E, R, S) (S[h]=S, COUNT=0) \\
C = (a, U, E, R, S) \\
\]

where \(C(s, S1) : NOTBOTH \cdot COUNT < 2\)

In the second example a simple history dependent requirement is met. Any user in \(U1\) may write in file \(F\) only if he has not read from file \(G\). First there must be an authorization for the reading operation:

CONFER THE USER GROUP NAMED \(U1\) WITH THE RIGHT TO 
read THE RESOURCE UNIT NAMED \(G\) always AND 
call THIS FRANCHISE readfileg

The occurrence of this operation must be recorded, however, to enforce the rest of the requirement:

always KEEP THE HISTORY OF SUCCESSFUL REQUESTS 
involving THE FRANCHISE NAMED readfileg AND 
call THIS EVENT didread

The access condition for writing file \(F\) will be dependent on this history information:

DEFINE THE CONDITION NAMED writecontrol FOR ACCESS 
BY didread = 0

CONFER THE USER GROUP NAMED \(U1\) WITH THE RIGHT TO 
write THE RESOURCE UNIT NAMED \(F\) ON THE 
CONDITION NAMED writecontrol AND CALL THIS 
FRANCHISE writefilef

The translations are as follows:

\[
READFILEG = (a, U1, (READ), FILEG, S) (S[h]=S, DIDREAD=0) \\
WRITEFILEF = (a, U1, (WRITE), FILEF, S1) \\
\]

where \(C(s, S1) : WRITECONTROL \cdot DIDREAD=0\)

In this last example the requirement was subject to two interpretations and the second interpretation, as discussed in chapter III, will be illustrated here. Under this interpretation the requirement is that a member of \(U1\) cannot write \(R1\) after reading \(R2\) unless he subsequently opens \(R3\) after each reading of \(R2\). The history keeping counter is used somewhat like a "flip-flop" in this case. Let \(pl\) be
the authorization giving U1 the right to read R2 and p2 be the authorization giving U1 the right to open R3.

Always keep the history of successful requests involving the franchise named p1 and call this event stop.

Always reset the history of successful requests involving the franchise named p2 and call this event stop.

In the second statement above, form (77) is used to reset the counter in response to a data base event. Next the ability of U1 to write R1 depends on this "flip flop".

Define the condition named writeacc for access by stop=0.

Confer the user group named u1 with the right to write the resource unit named rl on the condition named writeacc.

The corresponding translations are:

\[ p1 = (a, U1, (READ), R2, S1) \quad (S[h]=S, STOP, 0) \]
\[ p2 = (a, U1, (OPEN), R3, S2) \quad (S[h]=S, STOP, 3) \]
\[ p3 = (a, U1, (WRITE), R1, S3) \]

where \( C(s, S3) : \) WRITEACC : STOP=0

Additional statement types may also be used in PLC-5 for entering and invoking auxiliary programs for invocation before, during, and after an access decision is made. Separate statements are used to enter, (82), the program into the system and to initialize the conditions for its invocation, (82) through (87).

Enter the auxiliary program named monitor from auxfile

On the condition named noncond invoke the auxiliary program named monitor before evaluating the access condition of the franchise named descriptor.

The various times of invocation before, during, and after the access decision—given by the values 0, 1, 2, and 3 of \( b[v] \) in the model—are illustrated, respectively, in forms (83) through (86). Form (87) allows for a special
invocation (corresponding to h(v)=4) at login time, usually for user verification. Since each auxiliary program must usually be compiled and debugged prior to use in the access control system, it is assumed to be residing in a file, ready to be loaded and executed. Programs also may be removed from the system, using forms (88) and (89):

\[
\text{DISCONNECT THE AUXILIARY PROGRAM NAMED monitor FROM THE FRANCHISE NAMED desrights}
\]

\[
\text{DELETE THE AUXILIARY PROGRAM NAMED monitor}
\]

As with conditions, auxiliary programs can be directly invoked by an authorizer for test purposes, form (90):

\[
\text{TEST THE AUXILIARY PROGRAM NAMED monitor}
\]

Chapter III also has several examples of auxiliary program invocations. The syntax will be presented here in the same way as it was for the history keeping examples.

In the first example the owner of a file, R1, wishes to be notified on his terminal wherever a user in user group Ul does an update. The required statements are:

\[
\text{CONFER THE USER GROUP NAMED ul WITH THE RIGHT TO update,write THE RESOURCE UNIT NAMED r1 ON THE CONDITION NAMED cs1 AND CALL THIS FRANCHISE ulupdate}
\]

\[
\text{ENTER THE AUXILIARY PROGRAM NAMED report FROM sysfile1}
\]

\[
\text{always INVOKE THE AUXILIARY PROGRAM NAMED report WHEN SATISFYING THE ACCESS CONDITION OF THE FRANCHISE NAMED ulupdate}
\]

The corresponding translation is:

\[
\text{ULUPDATE} = (a,ul,(update,write),r1,s1) (s,REPORT,2)
\]

The second example uses an auxiliary program for threat monitoring. A program called STATS will keep track of average usage rates and call attention to unusual deviations from those averages. Here the program must be called every time the resource is accessed:
always **INVOKE** THE AUXILIARY PROGRAM NAMED stats
**WHEN SATISFYING THE ACCESS CONDITION OF THE**
FRANCHISE NAMED p2

The effect on p2 is shown here:

\[ p2 = (a,U2,ALL,R2,S2) (S,STATS,2) \]

In the next example data in a certain range of records
was of questionable integrity due to some accident during
input. Until the records were cleaned up, access was to be
temporarily blocked and an interim message of explanation
was to be sent to all who request access to this subset of
data. This message can be sent by a short auxiliary program
invoked upon a failure event associated with that set of
records. The SYSADMIN will enter:

**CONFER THE USER GROUP NAMED general WITH THE RIGHT**
**TO DO all OPERATIONS ON THE RESOURCE UNIT**
**NAMED r5 never AND CALL THIS FRANCHISE r5off**

always **INVOKE** THE AUXILIARY PROGRAM NAMED message
**WHEN NOT SATISFYING THE ACCESS CONDITION OF**
**THE FRANCHISE NAMED r5off**

The use of "always" in the above example is for
unconditional invocation in response to (in this case) an
unsuccessful request involving a given franchise; see forms
(84) through (86). The translation is:

\[ r5off = (SYSADMIN,GENERAL,ALL,R5,0) (S,MESSAGE,3) \]

In this last example history keeping and auxiliary
program invocation are used together to send an alarm
message to the owner of R7 after more than \( x \) unsuccessful
attempts by any user to gain access. Assuming the proper
access rights to R7 are conferred with a franchise called
ACCTOR7, the history keeping is next to be established:

always **KEEP THE HISTORY OF UNSUCCESSFUL ATTEMPTS**
**INVOLVING THE FRANCHISE NAMED acctor7 AND**
**CALL THIS EVENT attempts**

Next the auxiliary invocation conditions are established.

**DEFINE THE CONDITION NAMED too-many FOR AUXILIARY**
**PROGRAM INVOCATION BY attempts > x**
ON THE CONDITION NAMED too-many INVOKE THE AUXILIARY PROGRAM NAMED alarm WHEN NOT SATISFYING THE ACCESS CONDITION OF THE FRANCHISE NAMED acc7

The translations of these statements are:

\[ \text{ACCTOR7} = (a, U, R7, S)(S[h]=S, k, 1) (S[v]=S1, \text{ALARM}, 3) \]

where

\[ C(s, S1) : k > x \]

In order to be sure the history counter starts off at zero, the authorizer may then enter:

RESET THE HISTORY OF ACCESSES FOR THE EVENT NAMED attempts

which is simply translated to a history keeping statement with \( b[h]=0 \), resulting in:

\[ k = 0 \]

Although the protection languages are non-procedural, the auxiliary procedures which are invoked by the languages are procedural. These procedures are written in separate languages containing a full range of programming language constructs suitable for systems programming. These languages will be translated by the regular compilers or assemblers. Therefore, no syntax or semantics are given here for constructs internal to auxiliary programs.

**PLC-6: Use of Extended Resources**

Protection language statements are available for extending certain types of resources, by specifying exactly how they may be used in relation to other resources. In particular, this applies to the use of procedures, such as system and utility programs, for processing data resources. As an illustration consider an access routine ACCPROC for formatting fields of records, provided by the system in order that each user needn't write his own access programs. Some users will be able to use ACCPROC only to read files, others can use it to read and write. Also suppose that ACCPROC may be used on some files, but not others, as it might cause errors due to incompatible formatting. If
ACCPROC is considered to be just a data operation, there is no way to enforce these protection requirements. Here are some examples of how ACCPROC may be authorized for a particular purpose. Let it be assumed that ACCPROC has been declared as an extendable resource (this can be done with a semantic specification parameter of the next section).

**CONFER THE USER GROUP NAMED ul WITH THE RIGHT TO execute THE RESOURCE UNIT NAMED accproc ON THE CONDITION NAMED cs1 AND CALL THIS FRANCHISE ulexap**

This will establish Ul's right to execute ACCPROC, but ACCPROC has no access rights of its own yet.

**CONFER THE SYSTEM PROCEDURE NAMED accproc WITH THE RIGHT TO read,write THE RESOURCE UNIT NAMED rl ON CONDITION NAMED cs2 AND CALL THIS FRANCHISE accrl**

This gives some access rights to ACCPROC, but since ACCPROC is not a user or user group, it cannot make any accesses on its own. Therefore, a user group requesting to execute ACCPROC on Rl, for example, will need the resource ACCPROC to be extended by form (91):

**EXTEND THE RESOURCE UNIT OF THE FRANCHISE NAMED ulexap TO THE FRANCHISE NAMED accrl**

The translation of these statements is as follows. The two CONFER commands created separate authorizations:

\[
\text{UIEXAP} = (a, ul, \{\text{EXEC}\}, \text{ACCPROC}, \text{SI}) \\
\text{ACCR1} = (a, \text{ACCPROC}, \{\text{READ, WRITE}\}, \text{RI, S2})
\]

The EXTEND command connected them together with the \((b[x], L)\) pairs of the model.

\[
\text{UIEXAP} = (a, ul, \{\text{EXEC}\}, \text{ACCPROC}, \text{SI}) \ (a, \text{ACCR1}) \\
\text{ACCR1} = (a, \text{ACCPROC}, \{\text{READ, WRITE}\}, \text{RI, S2}) \ (1, 0)
\]

Next, it may be desired to let user group U2 execute ACCPROC, but only to read Rl.

**CONFER THE USER GROUP NAMED u2 WITH THE RIGHT TO execute THE RESOURCE UNIT NAMED accproc ON THE CONDITION NAMED cs3 AND CALL THIS FRANCHISE fu2**
Again, U2 still cannot do anything with ACCPROC until it is extended. But it cannot be extended to ACCRI, because this would allow U2 to write R1 with ACCPROC, as well as read. The following statements will suffice:

CONFER THE SYSTEM PROCEDURE NAMED accproc WITH THE RIGHT TO read THE RESOURCE UNIT NAMED r1 ON THE CONDITION NAMED cs4 AND CALL THIS FRANCHISE facc

EXTEND THE RESOURCE UNIT IN THE FRANCHISE NAMED fu2 TO THE FRANCHISE NAMED facc

Now, U2 can use ACCPROC to read, but not write, R1.

As with most other kinds of statements there are corresponding statements to remove (92) and display (93) resource extensions:

REMOVE THE EXTENSION OF THE RESOURCE UNIT OF THE FRANCHISE NAMED ulexap TO THE FRANCHISE NAMED acc1

DISPLAY EXTENSIONS OF THE RESOURCE UNIT OF THE FRANCHISE NAMED ulexap

Semantic Specification Constructs

PLC-7: Semantic Parameters

The security system designer (SYSDES) represents the highest level of sophistication, even higher than that of the SYSADMIN. The basic configuration of each implementation of an access control system must be specified. SYSDES, the authority with the highest level of sophistication, must be the one to specify certain details about which mechanisms are to be used and how. These highest level policy specifications allow the SYSDES to resolve some of the "dilemmas" of authorization and enforcement (previously discussed) in a way that best suits a given system application. Most of these parameters can be used directly by the authorization and enforcement processes to decide their execution sequences. Thus, these parameters allow for variable semantics without different system
The semantic specification statements can resolve the question of individual control of ownership versus a strong central authority. They can also express whether or not extended resources will be used and, if so, which resources are extendable. At this level, the SYSDES also may resolve the question of partial or full rejection in cases where a requested operation is only partly authorized. Most existing models of access control have fixed this choice. The ability here to specify this parameter allows a custom-made configuration, so that such policies need not be built into the basic model. This level of protection language construct may also be used to specify whether or not to include a LOCK and UNLOCK capability, or to vary the basic OR/AND operations of the enforcement process to an AND/OR, an OR/OR, or even an AND/AND operation. (Appendix G describes how the semantic parameters might affect the enforcement process.)

New access types (data operations) may also be defined at this semantic specification level. The section on the sets of the model in chapter II implies that E, the set of elemental operations, is a fixed set for any given implementation. More flexible policies can be enjoyed if new access types can be defined as needed. Requests for the basic operations of the data base system, such as READ and WRITE, would still be recognized internally by name or by an assigned code. New access types must necessarily be defined as macros or subroutines, combining the basic operations. These definitions might be presented to the system as procedures in the same way that auxiliary invocations are presented. Obviously, they must be carefully checked out, and, preferably, verified. This approach has been used to define all of the data operations in one model [BELLD73]. Another obvious requirement of systems allowing access type definitions is that the enforcement process must be able to correctly and unambiguously recognize requests for each of these operations. This recognition can be achieved by the naming scheme for the operations and their corresponding procedures. Examples of access types which the SYSDES, the SYSADM, or selected high level users might define are UPDATE, PRINT, SEARCH, SORT, CONFIRM, RELEASE, OPEN, CLOSE.

The following examples indicate how semantic specifications might appear for one system, form (94):

```
SELECT SEMANTIC PARAMETERS TO BE:
EP USERS OPERATION = or
EP RESOURCES OPERATION = and
```
The approach in this work has been to discuss protection language constructs through the pedagogical device of protection language components and using one particular choice of syntax. The design of a protection language for a working system entails two important steps. First, a set of protection constructs must be chosen from
the full set presented in the PLC's. This set of constructs determines the semantic power of the protection language and must be matched to the structures of the system and the needs of the authorizers who will be using the language. Secondly, a syntax must be selected in which to "clothe" the semantics. The choice of syntax determines the appearance and ease of use of the language.

A full set of protection language constructs has been presented. Almost any subset of features selected from these PLC's can be used to form a protection language. The set of all such protection languages is the family of protection languages. The choice of features for any particular language is to be guided by the following considerations:

1) Sophistication of the authorizer who will use the language
2) System performance
3) Security requirements for the language itself
4) Nature of the data base and its environment

The authorizer's level of sophistication is a direct function of his skill and experience. An example of how different levels of sophistication result in different constructs for the languages is seen in the following situation. At one level of sophistication a user may be content with a "passive defense" using basic authorizations and access conditions. These features are designed to keep unauthorized users outside of logical barriers around the data. At a higher level of sophistication an authorizer may want some "active" security features such as a combination of history keeping and auxiliary procedures. These features can be used to actively gain intelligence about security offenders and potential violators by threat monitoring, surveillance, and intruder identification. The level of sophistication is also largely dictated by the authorizer's need for complexity of protection constructs as a result of his job responsibilities. For example, a relatively naive data base user may need only simple authorization capabilities to control the use by others of the files he creates. To serve this need he may require only simple (enumerative) user group definitions, or he may even be able to use just the user groups which already exist in the system (as defined by other, higher level, authorizers). He also may need basic resource unit definitions. It is doubtful that access conditions will be used at the lowest level of sophistication, so these are not included in the corresponding language. Also, of course, the advanced
constructs such as history keeping are omitted at this level. At the other extreme of sophistication, the SYSADMIN will most certainly require most of these constructs. The low level user and the SYSADMIN, then, get very different subsets of features, resulting in protection languages with very different semantic power.

Performance considerations may affect the choice of features at any level. Any feature which is known to be costly in storage or processing (e.g., data dependent access conditions at the record or field level) may be restricted to languages used by only a few authorizers who are aware that anything but sparing use of these features is unacceptable. The cost of any feature should be weighed against the sensitivity of, and the value of, the data that feature is being used to protect.

The third consideration points out that even the use of a protection language must be subject to security controls. For example, the definition of valid systems users controls who may legally log into the system. This first line of security can be compromised if just any authorizer can make these definitions. Therefore, the use of languages containing constructs for making user definitions should be limited to authorizers having the proper degree of authority. As another example, auxiliary programs require extra care in their use. Therefore, only qualified authorizers should have access to protection languages featuring the associated constructs. In other words a protection language itself is also a resource of the system and, correspondingly, should have its use controlled.

The nature of the data base and its computing environment is a further consideration in the selection of constructs for protection languages. The various computing environments include the following examples:

--open type computer centers (e.g., at a university) where the set of users is constantly changing

--closed type computer centers with a fixed set of cooperative users (e.g., within one corporation)

--closed type computer centers with a relatively stable set of competing users (e.g., a commercial service bureau)

--classified military data base system
--a public commercial information system with many diverse customers
--a private information system (e.g., corporate MIS)
--a public time sharing network
--a private transaction system (e.g., for corporate billing and accounting)

Those various computing environments have an important effect on usage patterns and, therefore, on the issue of which protection constructs are likely to be most useful. For example, in certain of these environments there may be no access rules that would depend on access history. For this particular data base system no history keeping constructs are required for protection. Also, other protection constructs might be very difficult to implement in a specific type of system and, therefore, not likely to appear in the corresponding protection languages.

The other major part of protection language design is in the choice of a syntactic form. As previously discussed, there are a variety of approaches to syntax. A choice of syntax for a language may overlap more than one of the three categories which were described. The syntax for a given semantic construct can vary widely, just as it can in programming languages. As in the case of semantic features, the syntax of a protection language is selected on the basis of authorizer sophistication. The less sophisticated authorizers require a more natural, less terse syntax as an aid to his understanding of the use of the language. A more sophisticated user requires a more succinct language for efficiency in its use. Such a user soon will tire of a wordy syntax. The level of sophistication can be viewed in terms of complexity of the semantic features that one needs to complete his tasks as an authorizer. This is not always the same as the level of sophistication required in the syntax. For example, an authorizer experienced in the use of protection languages may still only need a minimal set of constructs, because his position may not necessitate him to express many different kinds of protection requirements. This authorizer needs a less powerful language, but a terse syntax. Alternatively, it is possible that an authorizer’s responsibilities require a rather full complement of semantic constructs, but his lack of experience as an authorizer obliges him to start with an easy-to-use syntax. For this situation the wordy syntax (as used in chapters V and VI) may be most suitable. An advantage of this syntax
is that, as the authorizer becomes more proficient (as often happens rather rapidly), the expressions may be abbreviated by omitting the optional filler words. Thus, this type of syntax actually represents a range of syntax forms, depending on how much of the optional part is used. It is expected that, most commonly, users will proceed rather quickly to the more terse expressions.

This adjustability of wordiness can also be built into the interactive type of syntax. If response time is affected by long prompting messages, the experienced authorizer may elect to go to a very abbreviated form for messages, or to eliminate prompting altogether. This variability of wordiness, along with the natural ease in using the interactive dialogue—at any level of sophistication—makes this form of syntax strongly attractive for many applications.

Some examples of languages, selected on the basis of levels of authorizer sophistication and having very different forms of syntax, are developed in [WONGD75].

A Note on Implementation

There are basically two reasons for implementing the model of access control presented in the preceding chapters. The first reason is to demonstrate the functional value of the concept of protection languages for authorization and to demonstrate the features of the model itself. A second reason for implementation is to observe the effect of the various protection constructs on system performance. The implementation is being carried out in two parts, the authorization process and the enforcement process. Inter-process communication will be achieved through the internal representation of the access control information (see again (6), (7), and (13) of figure II.1). The authorization process creates and maintains the internal representation in response to incoming protection language specifications, because of its ability to be understood by many levels of users, an interactive dialogue-driven syntax was selected as the syntax for language implementation. A middle-to-high level level of sophistication was chosen as a basis for selecting semantic constructs, in order to demonstrate most of the constructs. The enforcement process, simply put, uses the internal access control
information to make access decisions.

The system environment of the implementation is the hardware and software of a DEC PDP-10 computer and an attribute-based data base management system, also currently being implemented. A structured FORTRAN (called PREST4), designed within the project on Data and Data Secure Systems, is being used for almost all of the coding. It's major limitation is a restriction to fixed record I/O to secondary storage.

While the needs of real protection systems were kept in mind throughout the modelling process, no restrictions were made to the model on the basis of any particular implementation. At implementation time, then, the questions arose as to how much of the model to include and how much policy to build into the mechanisms. Since the main reason for implementation is to demonstrate the basic concepts and since the manpower was severely limited, there are a number of deliberate limitations. For example, multiple ownership of resources will not be supported. Neither will more than one history triple or auxiliary invocation triple per franchise. One of each per franchise is sufficient to demonstrate the concepts, while more than one causes extra complexity in the data structures. No inter-file resource unit definitions will be allowed, but grouping into resource unit groups will fully compensate for this lack. It is a minor drawback that no garbage collection is used to handle deletion operations in the data structures. A certain amount of policy has been built into the display operations; namely that only the creating authorizer and the SYSADMIN can display access control information. The creating authorizer's identification number is carried with each definition because of this policy. The identification number is also used with each name (of user groups, conditions, franchises, etc.) assigned by an authorizer to guarantee the uniqueness of local names.

A very natural departure from the abstract model is seen in the data structures of the franchises. Instead of having numerous 5-tuples floating around, the implementation calls for one franchise per user group, with all authorizations for that user group to be merged into that one franchise. No protection capability is lost, but this form is much easier to use for reference in computing access decisions. At login time all of the franchises (for which the logged-in user is a member of the corresponding user group) are further merged into one effective franchise, retained until that user logs off.
Many of the problems encountered are a function of the implementation environment—programming language, operating system, and data base system. There are the usual problems due to design tradeoffs, such as the tradeoff between ease of construction and ease of reference in some of the data structures. Since conditions are being implemented as a special type of short subroutine and the computer system does not support a dynamic compiling and linking capability, conditions cannot be dynamically defined. Therefore, conditions must be translated and linked along with the rest of the protection software. This is an inconvenience that could be eliminated by writing a special translator which could access various symbol tables within the system in order to dynamically resolve main memory addresses. Such a translator would be several orders of magnitude simpler than a programming language translator in that it would apply only to the limited forms found in condition definitions.

A few other parts of the implementation have required careful consideration, such as the running of auxiliary procedures under the rights of the authorizer who specified the procedure and not the user's (whose request caused the invocation). Special handling of the resource unit definitions also is needed because of their being in several parts due to the attribute-based data organization.

Part of the reason for implementation is to observe performance. Certain instrumentation software is planned as a part of the implementation. Almost anyone can predict that more complex protection constructs will result in more storage and processing overhead. There is a definite lack of existing data relating this extra cost to the protection constructs. Thus, it will be interesting and useful to determine roughly what percent extra it will cost to add, for example, a history keeping ability to an existing protection system—even just for the specific implementation of this specific system. It will also be useful, for example, to know if dynamic evaluation of access conditions increases the average protection processing of a request by a factor of two or a factor of ten. Only with this kind of information can the true value of each protection construct be determined.

In any implementation of a protection system the question of program verification comes to mind. Although a study in the project on Data Security and Data Secure Systems [107] has obtained some program proving results for some general and complex data structures, program verification is not included in the scope and goals of this implementation. Much progress is being made in other
efforts [POPEG73, GOODD74, BELLD74], however, in this direction. (The reader may see almost any issue of IEEE Transactions on Software Engineering for current work in the design, proving, and testing of programs.) It must be pointed out, though, that even these approaches are far from guaranteeing a completely fail-safe system. For example, there might be an unknown error in a proof. Who will prove that the proofs are all correct? Many such proofs are long and complex. Also, the proving of programs is not at all sufficient to prove the correctness of systems. It is very difficult, indeed, to be completely circumspect at the system level, against all untoward events. The kernel approach [LIPNS74, POPEG74a, WULF74], which gathers the most sensitive parts into one small package, is, in one way, a good approach to the problem on a system level. In another way, it begs the question. How can it be known for sure that an incomplete case analysis hasn't allowed some potentially sensitive section of code to escape the kernel? The reason for collecting the sensitive code into a small package is the feasibility of proving something about a smaller amount of code. It can be proven that all the code in a reasonably small kernel is correct. But it must also be proven that none of the code outside the kernel can cause a compromise of security. Therefore, the problem of proving something about a large volume of code suffers a relapse. Furthermore, the proofs must be translated into reality by showing that the hypotheses, upon which the proofs are based, are actually valid in the real system. This must be proved on some meta-system basis.

At the very least, however, the program proving work is forcing a close look at the structures of programs and systems, a worthwhile effect in itself.
CHAPTER VII

SUMMARY

When all is said and done, more is said than done.
—Unknown

Currently, languages for authorization in data bases do not exist. This dissertation develops a set-oriented model of access control based on the principle of separation between policies and mechanisms [WULFW74]. The model is used to provide a semantic base for constructs of protection languages at many levels of sophistication. Since the model is tied neither to any particular system structure or data organization nor to any protection policies, it is applicable to operating system security as well as data base security, accommodating a wide range of protection policies.

The role of the authorizer, as distinct from the data base user, is emphasized. The concept of ownership is used for validation of authorizations in the model. Various ownership policies, such as group ownership, subownership, and conditional ownership can be represented. The basic sets of the model are presented and the set of system states is derived from the set of all values of the resources. Resources may be defined in many degrees of granularity, from single data elements to the entire data base in one resource unit. Subsets of states are defined by restrictions on resource values, described by Boolean expressions of arithmetic relations on the resource values. These expressions are known as conditions. The concept of a five-dimensional security space is used to visualize how authorization specifications and access requests are manipulated by the authorization and enforcement processes. Several examples are presented to illustrate the relationships among the various parts of the security space.

140
Within the model, users and resources can be grouped in many possibly overlapping subsets. This grouping ability, backed up by a corresponding enforcement process based on an OR-AND operation, provides a powerful means for expressing relations among users and resource elements. A general class of users is defined to establish an overall minimum level of access rights without an explicit representation for each and every user. Conditions are used to characterize sets. In this role, they can serve as dynamically evaluated definitions of user groups and resource units. The most important use of these expressions is as access conditions, which allow each access decision to depend dynamically on information related to the user, to resources, to data content, to access history, and to the general state of the system.

An important contribution is the group of constructs called the advanced protection services. These services include history keeping, auxiliary program invocation, and extended resources. History keeping is used to allow a sensitivity to the context of access history so that access decisions can be based on the occurrence or non-occurrence of previous operations. It is accomplished in a way which avoids dominance and also avoids keeping unnecessary history data—two problems which are common to many history keeping mechanisms. The recording of history itself can also be conditional, allowing for the recording of full historical context. That is, a given access history will be kept only if a history-keeping condition is satisfied. Therefore, information about the state of the system at the time of the access is kept along with the history. Since the history-keeping condition can refer to any resource value in the database or the system itself, any part of the historical context of a database event can be recorded.

Provision is made for the specification of auxiliary program invocations to achieve additional protection measures before, during, and after the access decision-making process. These programs may be used for journaling, alarm and recovery, threat monitoring, user authentication, and data entry validation. The concept of extended resources allows a user to invoke procedures having greater access rights than does the user himself.

The authorization and enforcement processes are modeled in the security space. Validation of authorization by rules of ownership is emphasized in the authorization process. The most promising enforcement policy is developed extensively: When the user is in more than one group, enforcement is shown to be based on a logical OR of the
access conditions for each group. When the requested resource unit overlaps more than one authorized unit, enforcement is based on the logical AND of the access conditions to those units. Furthermore, the requested resource unit must be covered by the authorized units.

The model is followed by some further examples to illustrate the application of the authorization and enforcement processes in the security space.

The development of protection languages, the key to expression of policies and rules of access by authorizers, is the principle aim of this dissertation. A family of protection languages, based on many levels of authorizer sophistication, is aimed to replace the typically crude present methods for entering access control information, allowing the use of a much broader spectrum of authorization policies. Protection languages are characterized as mainly non-procedural languages for specifications and declarations and not for programming. Translation of protection languages consists of lexical translation and semantic translation. Each protection language has its own syntax. The forms of syntax are grouped into three broad categories: an interactive, prompted, "menu-driven" syntax for naive users; a wordy, somewhat natural syntax in which the authorizer of medium sophistication writes his own specifications; and a succinct, terse syntax for efficient use by experienced and sophisticated authorizers with extensive authorization responsibility. Each language has a different lexical translator to transform all syntax into a single standard form, namely the tuples and other entities of the model. The model, an interpretive model of protection semantics, provides a common base language [DENNJ72] for protection languages. The semantics are interpreted partly by the authorization process, but mostly by the execution of the enforcement process.

The approach to presentation of the protection language constructs is a "mini-language" approach [LEDGH71]. The protection language constructs are divided into several groups in order to allow brief descriptions of their salient features. These groups, called protection language components (PLC's) are summarized as follows:

PLC-1 Simple user group and resource unit definitions, comments.

PLC-2 Definition and testing of conditions.
PLC-3 Generalized user group definitions using conditions, resource unit groups, single user descriptions.

PLC-4 Basic operational commands and use of access conditions.

PLC-5 Access history keeping and auxiliary program invocation.

PLC-6 Use of extended resources.

PLC-7 Semantic Parameters.

The constructs are also broken into four basic categories: definitional statements, operational statements, advanced protection services, and semantic parameters. The definitional category provides the means to establish definitions of users, user groups, resource units, and conditions. The operational category contains statements that group the definitions together in ways that specify the basic information about who can do what operations, on what resources, and under what conditions. Operational statements also include the means for testing and displaying of the entities defined in the definitional category. The advanced protection services contain protection constructs to be used by more sophisticated authorizers. These constructs correspond to the full specifications of the model. Access history keeping is in this category. Language statements are used to establish history keeping counters and to associate them with the required access operations (data base events). These statements allow that association to be for either successful or unsuccessful access requests or to be independent of the outcome of the access decision. Auxiliary procedure invocations and extended resources are also part of the advanced protection services. Protection language statements control the entry and invocation of these procedures and, as in the case of history keeping, provide associations with specific events.

The final category, the category for semantic parameter specification, provides the means to define protection policies at the highest levels of sophistication. Such high level policy decisions are irrevocably built into most protection systems. These protection language constructs allow some of these decisions to be specified as parameters. Since the parameters affect the function of the enforcement process, and the execution of the enforcement process determines the semantic interpretation of protection
language statements, these parameters represent a kind of variable semantics.

The complete specification of a sample protection language is used to discuss the translation of protection constructs into the entities of the model.

The process of protection language design requires a choice of syntax and a choice of protection constructs. Both choices are governed by the level of sophistication toward which the language is aimed. Some discussion is given to the problem of implementation.
APPENDIX A

A GENERAL SET NOTATION FOR CONDITIONS

This appendix is a brief discussion of the use of set notation in representing general condition expressions, as referred to in the section on conditions in chapter II. Recall the resources of the example in chapter II:

\[ V(rl) = \{1,2,3,4,5,6,7,8,9,10\} \]
\[ V(r2) = \{1,2,3,4,5\} \]

The restriction is: \( v(rl) + v(r2) < 5 \). Here, \( v(rl) \) and \( v(r2) \) are not independent. There can be no choice for \( V(rl) \) and \( V(r2) \) which will allow their product to be the required state subset:

\[ S1 = \{(1,1),(1,2),(1,3),(2,1),(2,2),(3,1)\} \]

A union of products is a general form in set notation, analogous to a sum of products in Boolean algebra. Let a state subset be described by an equation of the following form:

\[ S1 = \bigcup_{j=1}^{n} \left( X \ V[j](r[i]) \right) \]

In the application of this equation, the proper subsets \( V[j](r[i]) \) must be chosen to make the final product yield the required state subset. In considering the restriction for \( S1 \): \( v(rl) + v(r2) < 5 \)

let \( V1(rl) = \{1\} \) and \( V1(r2) = \{1,2,3\} \); \( V2(rl) = \{1,2\} \) and \( V2(r2) = \{1,2\} \); \( V3(rl) = \{1,2,3\} \) and \( V3(r2) = \{1\} \);

\[ S1 = \bigcup_{j=1}^{3} \left( X \ V[j](r[1]) \right) \]

\[ j=1 \quad i=1 \]

145
Appendix A

A General Set Notation for Conditions

\[ V_1(r_1) \times V_1(r_2) \cup V_2(r_1) \times V_2(r_2) \cup V_3(r_1) \times V_3(r_2) \]

\[ = \{(1,1),(1,2),(1,3)\} \cup \{(1,1),(1,2),(2,1),(2,2)\} \cup \{(1,2),(2,1),(3,1)\} \]

\[ S = \{(1,1),(1,2),(1,3),(2,1),(2,2),(3,1)\} \]
APPENDIX B

SETS DEFINED BY CONDITIONS

The reader should be familiar with the material in the sections on conditions in chapter II and on the security space in chapter III before reading this appendix. The section on conditions made reference to some relationships among sets defined by conditions. \( C(u,U) \) was given as a user group defining condition and \( C(r,R) \) as a resource unit defining condition. The complication arises with the access condition. The section on authorization specifications in chapter II might have begun by defining an authorization specification to be a 5-tuple of this form:

\[ p = (a,U,E,R,X) \]

where \( a \in A \), \( U \subseteq U \), \( E \subseteq E \), and \( R \subseteq R \). The condition which defines \( X \) (the access condition) is easier to visualize than the set \( X \) itself. This condition will be used roughly in the following way in an access decision for an access request, \( q \):

1) Does \( p \) apply to \( q \)? If not, no decision is made using this \( p \).
2) If so, evaluate the characteristic function of \( X \), \( C(x,X) \). If it has a value of 1, allow \( q \). If not, deny \( q \).

Therefore, \( X \) is the set of all access requests which \( p \) allows and \( X \) is a subset of the access requests to which \( p \) applies.

To put this more precisely, the set of requests of the form:

\[ q = (u,e,R,s) \]

to which \( p \) applies is:

147
SETS DEFINED BY CONDITIONS

\[ Q[p] = \{ q : u(q) \in U(p) \text{ and } e(q) \in E(p) \text{ and } R(q) \cap R(p) = \emptyset \} \]

Consider all \( q[p] \in Q[p] \) and let

\[ Q[p]^+ = \{ q[p] \mid p \text{ allows } q \} \]

Clearly, \( Q[p]^+ \) is that subset of \( Q[p] \) allowed by \( p \), or \( Q[p]^+ \) is the set \( X \), and its characteristic function,

\[ C(q[p],Q[p]^+) = \begin{cases} 1, & \text{if } q[p] \in Q[p]^+ \\ 0, & \text{otherwise} \end{cases} \]

will be called the access condition of \( p \). Since these are the only subsets of access requests associated with conditions, the notation is simplified to \( C(q,Q) \).

Clearly, conditional set membership is just an extension of the special case of unconditional membership, for which each characteristic function is just a constant function. In the model, then, set membership is often a function of time, or more precisely, a function of system state. Therefore, each condition can actually be the characteristic function of two different sets from two different domains. Recall that each of the types of conditions—\( C(u,U) \), \( C(r,R) \), \( C(q,Q) \)—are characteristic functions of their corresponding types of sets: \( U \), \( R \), and \( Q \) (derived in this appendix), respectively. However, each condition defines its set by defining the set of states (in terms of a range of resource values) for which a given element is a member of the set in question. Therefore, each condition also defines a set of states. That is, for every \( C(u,U) \) defining a set \( U \), there is a corresponding set \( S \) also defined by \( C(s,S) = C(u,U) \). The same is also true for each \( C(r,R) \) and \( C(q,Q) \). \( S \) is the domain of \( S \) and \( U \) is the domain of \( U \). Thus, as illustrated in figure B.1, the characteristic function is an isomorphism between the two domains, \( U \) and \( S \). A similar relationship exists between \( R \) and \( S \), and \( Q \) and \( S \).
Figure B.1 - The Relationship Between U and S
This appendix presents a Backus-Naur Form (BNF) syntax of a sample protection language, adapted from [WONGD75]. The first part of the appendix is an index to the syntax rules. To the left of the semicolon, each index entry indicates the number of the rule in which the index term is defined. To the right of the semicolon is a list of the rules in which the index term is used.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>access-cond</td>
<td>41; 39</td>
</tr>
<tr>
<td>access-cond-description</td>
<td>48; 41</td>
</tr>
<tr>
<td>access-cond-evaluation</td>
<td>110; 109</td>
</tr>
<tr>
<td>adjust-action</td>
<td>100; 99</td>
</tr>
<tr>
<td>alter</td>
<td>76; 62</td>
</tr>
<tr>
<td>alter-action</td>
<td>79; 78, 80</td>
</tr>
<tr>
<td>alter-entity</td>
<td>77; 76</td>
</tr>
<tr>
<td>answer</td>
<td>124; 120</td>
</tr>
<tr>
<td>attribute</td>
<td>32; 31, 52</td>
</tr>
<tr>
<td>aux-cond</td>
<td>43; 39</td>
</tr>
<tr>
<td>aux-cond-description</td>
<td>54; 43</td>
</tr>
<tr>
<td>aux-stmt</td>
<td>104; 93</td>
</tr>
<tr>
<td>c-type</td>
<td>57; 56, 61, 92</td>
</tr>
<tr>
<td>change-list</td>
<td>83; 81</td>
</tr>
<tr>
<td>comment-stmt</td>
<td>127; 1</td>
</tr>
<tr>
<td>cond</td>
<td>72; 64, 95, 108</td>
</tr>
<tr>
<td>cond-change</td>
<td>85; 83</td>
</tr>
<tr>
<td>cond-identifier</td>
<td>38; 22, 37, 59, 61, 72, 92</td>
</tr>
<tr>
<td>cond-identifier-list</td>
<td>59; 56, 85</td>
</tr>
<tr>
<td>cond-type</td>
<td>39; 37</td>
</tr>
</tbody>
</table>
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

condition 37; 4
confer 63; 62
confer-name 66; 65
content-comparison-exp 52; 48
def-display 92; 89
def-entity 4; 3
define-stmt 3; 2
definition 89; 88
definitional-stmt 2; 1
delete-entity 56; 55
delete-stmt 55; 2
depth 15; 11
display 87; 62
display-entity 88; 87
divest 73; 62
ever-identifer 105; 104
ever-name 98; 95, 101
extend 101; 89, 99, 103
extend-display 115; 114
extend-remove 116; 114
extend-stmt 114; 93
filename 17; 6, 28, 105
file-identifier 18; 17
franchise-alter 81; 77
franchise-description 82; 81
franchise-display 132; 89
franchise-identifier 71; 66, 74, 86
franchise-name 86; 73, 82, 89, 95, 102,
109, 111, 115, 116, 117
131; 89
franchise-ref 101; 94
franchise-resource 42; 39
hist-adjust 53; 42
hist-condition 102; 94
hist-condition-description 103; 94
hist-condition-disconnect 95; 94
hist-display 94; 93
hist-record 129; 12, 13, 14, 15, 18, 21,
27, 29, 32, 35, 38, 50, 71,
98, 107
hist-stmt 6; 5, 77
individual-users 108; 104
invoke 139; 138
invoke-time 31; 30
keyword 30; 28
keyword-list 121; 120
logical-op 74; 62
name 65; 63
named-confer
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

name-list
name-type
object-name
op
op-change
operational-stmt
op-list
outcome
password
pgm-delete
pgm-identifier
pgm-name
pgm-remove
pgm-test
proc-list
project
protection-stmt
protection-service-stmt
record-action
rejection
relational-op
resource
resource-group-alter
resource-group-identifier
resource-group-identifier-list
resource-group-name
resource-identifier-identifier-list
resource-unit
resource-unit-group
resource-unit-identifier
resource-unit-name
resource-var-assign
semantic-stmt
single-user-description
single-user-list
subject-name
sysowns
sys-par-assign
sys-par-assign-list
sys-proc
sys-proc-name
sys-var
sys-var-comparison-exp
test-stmt
text
unnamed-confer
user
user-class
user-cond
user-cond-description
**APPENDIX C**

**SAMPLE PROTECTION LANGUAGE SYNTAX**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>user-group</td>
<td>19; 5</td>
</tr>
<tr>
<td>user-group-alter</td>
<td>78; 77</td>
</tr>
<tr>
<td>user-group-description</td>
<td>22; 19</td>
</tr>
<tr>
<td>user-group-identifier</td>
<td>21; 20, 53</td>
</tr>
<tr>
<td>user-group-identifier-list</td>
<td>58; 56, 58</td>
</tr>
<tr>
<td>user-group-name</td>
<td>20; 19, 67, 78, 92</td>
</tr>
<tr>
<td>user-id</td>
<td>8; 7, 67, 92</td>
</tr>
<tr>
<td>user-name</td>
<td>13; 11</td>
</tr>
<tr>
<td>user-number</td>
<td>9; 8, 23</td>
</tr>
<tr>
<td>user-number-list</td>
<td>23; 22, 23, 56, 78</td>
</tr>
<tr>
<td>user-tmp-identifier</td>
<td>29; 28</td>
</tr>
<tr>
<td>user-var</td>
<td>46; 45</td>
</tr>
<tr>
<td>user-var-assign</td>
<td>11; 7, 11</td>
</tr>
<tr>
<td>user-var-comparison-exp</td>
<td>45; 44</td>
</tr>
<tr>
<td>value</td>
<td>51; 45, 49, 52</td>
</tr>
</tbody>
</table>

The syntax rules begin on the next page.
Syntax Rules for a Sample Protection Language

(1) \(<protection-stmt>: = \<definitional-stmt>/\n   \<operational-stmt>/\n   \<protection-service-stmt>/\n   \<semantic-stmt>/\n   \<comment-stmt>\n
(2) \(<definitional-stmt>: = \<define-stmt>/\<delete-stmt>/\<test-stmt>\n
(3) \(<define-stmt>: = \text{DEFINE} \<def-entity>\n
(4) \(<def-entity>: = \<user>/\<resource>/\<condition>\n
(5) \(<user>: = \<individual-users>/\<user-group>\n
(6) \(<individual-users>: = \text{USER BY ASSIGNING} \n   \<single-user-description>/\n   \text{USERS, BEGIN ASSIGNING:} \<single-user-list>\n   \text{END/} \n   \text{USERS, FROM THE FILE NAMED} \<filename>\n
(7) \(<single-user-description>: = \text{TO} \<user-id>, \n   \<user-var-assign>\n
(8) \(<user-id>: = \text{USER} \<user-number>\n
(9) \(<user-number>: = \text{a 4-digit decimal number.}\n
(10) \(<single-user-list>: = \<single-user-description>/\n       \<single-user-description>;\n       \<single-user-list>\n
(11) \(<user-var-assign>: = \<user-var-assign>, \n    \text{name=} \<user-name>/\n    \text{project=} \<project>/\n    \text{dept=} \<dept>/\n    \text{class=} \<user-class>/\n    \text{password=} \<password>\n
(12) \(<password>: = \<identifier>\n
(13) \(<user-name>: = \<identifier>\,', \text{ASCII string of up to 32} \n    \text{characters formed by concatenating the} \n    \text{user's last name, first name, and middle} \n    \text{initial.} \n
(14) \(<project>: = \<identifier>\n
(15) \(<dept>: = \<identifier>\n

(16)  <user-class> := a / b / c . . . / y / z

(17)  <filename> := <file-identifier> / & <file-identifier>

(18)  <file-identifier> := <identifier>

(19)  <user-group> :=
          <user-group-name> <user-group-description>

(20)  <user-group-name> := THE USER GROUP NAMED
          <user-group-identifier>

(21)  <user-group-identifier> := <identifier> / general

(22)  <user-group-description> := TO HAVE THESE MEMBERS:
          <user-number-list> / BY THE CONDITION NAMED
          <cond-identifier>

(23)  <user-number-list> := <user-number> / <user-number>,
          <user-number-list>

(24)  <resource> := <resource-unit> / <resource-unit-group>

(25)  <resource-unit> := <resource-unit-name> BY ASSIGNING
          <resource-var-assign>

(26)  <resource-unit-name> := THE RESOURCE UNIT NAMED
          <resource-unit-identifier>

(27)  <resource-unit-identifier> := <identifier>

(28)  <resource-var-assign> := <resource-var-assign>,
          <resource-var-assign> / file = <filename>/
          template = <user-tmp-identifier>/ keywords =
          (<keyword-list>)

(29)  <user-tmp-identifier> := <identifier>

(30)  <keyword-list> := <keyword> / <keyword>, <keyword-list>

(31)  <keyword> := <attribute> = <value>

(32)  <attribute> := <identifier>, the name of a field
          corresponding to the specified file in the
data base

(33)  <resource-unit-group> := <resource-group-name> TO HAVE
          THESE MEMBERS: <resource-identifier-list>
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

(34) \(<resource-group-name>:= \text{THE RESOURCE UNIT GROUP NAMED} \newline <resource-group-identifier>\)

(35) \(<resource-group-identifier>:= <identifier>\)

(36) \(<resource-identifier-list>:= \newline <resource-unit-identifier>/ <resource-unit-identifier>, <resource-identifier-list>\)

(37) \(<condition>:= \text{THE CONDITION NAMED} <cond-identifier> \text{ FOR} <cond-type>\)

(38) \(<cond-identifier>:= <identifier>\)

(39) \(<cond-type>:= <user-cond>/ <access-cond>/ <hist-cond>/ <aux-cond>\)

(40) \(<user-cond>:= \text{A USER GROUP BY} <user-cond-description>\)

(41) \(<access-cond>:= \text{ACCESS BY} <access-cond-description>\)

(42) \(<hist-cond>:= \text{HISTORY KEEPING BY} \newline <hist-cond-description>\)

(43) \(<aux-cond>:= \text{AUXILIARY PROGRAM INVOCATION BY} \newline <aux-cond-description>\)

(44) \(<user-cond-description>:= \text{.true.} / \text{.false.} / \text{a boolean} \newline \text{expression of} <user-var-comparison-exp> \text{connected by} \text{.NOT., .AND., .OR.}\)

(45) \(<user-var-comparison-exp>:= \newline <user-var><relational-op><value>/ <user-var><relational-op><user-var>\)

(46) \(<user-var>:= \text{id/ project/ dept/ name/ class/ name/ password/ login-time/ subsystem}\)

(47) \(<relational-op>:= \text{.EQ./ .NE./ .GT./ .LT./ .LE./ .GE.}\)

(48) \(<access-cond-description>:= \text{.TRUE./ .FALSE./ a boolean} \newline \text{expression of} <sys-var-comparison-exp>'s and <content-comparison-exp>'s\)

(49) \(<sys-var-comparison-exp>:= \newline <sys-var><relational-op><value>/ <sys-var><relational><sys-var>\)
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

\[(54) \text{<sys-var>} := \text{<identifier>}, \text{chosen as deemed appropriate to the particular system concerned.}\]

\[(51) \text{<value>} := \text{a valid numerical or ASCII value.}\]

\[(52) \text{<content-comparison-exp>} :=
\begin{align*}
\text{<attribute><relational-op><value>}, \\
\text{<attribute><relational-op><attribute>}
\end{align*}
\]

\[(53) \text{<hist-cond-description>} := \text{.TRUE./ .FALSE./ a boolean expression of <sys-var-comparison-exp>'s.}\]

\[(54) \text{<aux-cond-description>} := \text{.TRUE./ .FALSE./ a boolean expression of <sys-var-comparison-exp>'s.}\]

\[(55) \text{<delete-stmt>} := \text{DELETE <delete-entity>}\]

\[(56) \text{<delete-entity>} := \text{USERS <user-number-list>/ USER GROUPS NAMED <user-group-identifier-list>/ RESOURCE UNITS NAMED <resource-identifier-list>/ RESOURCE UNIT GROUPS NAMED <resource-group-identifier-list>/ <c-type> CONDITIONS NAMED <cond-identifier-list>}\]

\[(57) \text{<c-type>} := \text{USER-GROUP DEFINING/ ACCESS/ HISTORY KEEPING/ AUXILIARY PROGRAM INVOCATION}\]

\[(58) \text{<user-group-identifier-list>} := \text{<user-group-identifier>}, \text{<user-group-identifier-list>}\]

\[(59) \text{<cond-identifier-list>} := \text{<cond-identifier>}, \text{<cond-identifier-list>}\]

\[(60) \text{<resource-group-identifier-list>} :=
\begin{align*}
\text{<resource-group-identifier>}, \\
\text{<resource-group-identifier>}
\end{align*}\]

\[(61) \text{<test-stmt>} := \text{TEST <c-type> CONDITION NAMED <cond-identifier>}\]

\[(62) \text{<operational-stmt>} := \text{<confer>/ <divest>/ <name>/ <alter>/ <display>}\]

\[(63) \text{<confer>} := \text{<unnamed-confer>/ <named-confer>}\]

\[(64) \text{<unnamed-confer>} := \text{CONFIER <subject-name> WITH THE RIGHT TO <op-list> <object-name> <cond>}\]
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

(65) <named-confer> := <unnamed-confer> <confer-name>

(66) <confer-name> := AND CALL THIS FRANCHISE
    <franchise-identifier>

(67) <subject-name> := <user-id> / <user-group-name> / 
    <sys-proc-name>

(68) <op-list> := <op> / <op>, <op-list>

(69) <op> := read / write / own / subown / delete / display / open / 
    add / create / 
    compare / execute / update / do all operations

(70) <object-name> := <resource-unit-name> / 
    <resource-group-name>

(71) <franchise-identifier> := <identifier>

(72) <cond> := always / never / ON THE CONDITION NAMED
    <cond-identifier>

(73) <divest> := DIVEST <subject-name> OF THE RIGHT TO
    <op-list> <object-name> / DIVEST
    <franchise-name>

(74) <name> := ASSIGN THE NAME <franchise-identifier> TO
    <franchise-ref> TO <op-list> <object-name>

(75) <franchise-ref> := THE FRANCHISE OF <subject-name>

(76) <alter> := ALTER <alter-entity>

(77) <alter-entity> := <individual-users> / 
    <user-group-alter> / <resource-unit> / 
    <resource-group-alter> / <franchise-alter>

(78) <user-group-alter> := <user-group-name> BY
    <alter-action> THESE MEMBERS:
    <user-number-list>

(79) <alter-action> := ADDING / REMOVING

(80) <resource-group-alter> := <resource-group-name> BY
    <alter-action> THESE MEMBERS:
    <resource-identifier-list>

(81) <franchise-alter> := <franchise-description> BY
    REPLACING EXISTING <change-list>
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

(82) <franchise-description> ::= <franchise-name>/
     <franchise-ref> FOR <object-name>

(83) <change-list> ::= <op-change>/
     <cond-change>/ <op-change>
     AND <cond-change>/ <cond-change> AND
     <op-change>

(84) <op-change> ::= OPERATIONS WITH <op-list>

(85) <cond-change> ::= ACCESS CONDITIONS NAMED
     <cond-identifier-list> WITH
     <cond-identifier-list>

(86) <franchise-name> ::= THE FRANCHISE NAMED
     <franchise-identifier>

(87) <display> ::= DISPLAY <display-entity>

(88) <display-entity> ::= <definition>/ <name-list>

(89) <definition> ::= DEFINITION OF <def-display>/
     <franchise-ref>/<franchise-name>/
     <franchise-resource>/ <franchise-display>/
     <event-name>

(90) <name-list> ::= NAMES OF <name-type>

(91) <name-type> ::= USER GROUPS/ RESOURCE UNITS/ RESOURCE
     UNIT GROUPS/ FRANCHISES/ EVENTS

(92) <def-display> ::= <user-id>/ <user-group-name>/
     <resource-unit-name>/ <resource-group-name>/
     <c-type> CONDITION NAMED <cond-identifier>

(93) <protection-service-stmt> ::= <hist-stmt>/ <aux-stmt>/
     <extend-stmt>;

(94) <hist-stmt> ::= <hist-record>/ <hist-adjust>/
     <hist-disconnect>/ <hist-display>

(95) <hist-record> ::= <cond> <record-action> THE HISTORY OF
     <outcome> INVOLVING <franchise-name> AND
     CALL THIS EVENT <event-identifier>

(96) <record-action> ::= KEEP/ RESET

(97) <outcome> ::= SUCCESSFUL REQUESTS/ UNSUCCESSFUL ATTEMPTS/
     REQUESTS REGARDLESS OF OUTCOME
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

(98) <event-identifier>::=<identifier>

(99) <hist-adjust>::=<adjust-action>THE HISTORY OF ACCESSES FOR <event-name>

(100) <adjust-action>::=RESET/INCREMENT

(101) <event-name>::=THE EVENT NAMED <event-identifier>

(102) <hist-disconnect>::=DISCONNECT THE HISTORY OF ACCESSES INVOLVING <franchise-name>

(103) <hist-display>::=DISPLAY THE SYSTEM COUNTER FOR <event-name>

(104) <aux-stmt>::=<enter>;/<invoke>;/<pgm-remove>;/<pgm-delete>;/<pgm-test>;

(105) <enter>::=ENTER <pgm-name> FROM <filename>

(106) <pgm-name>::=THE AUXILIARY PROGRAM NAMED <pgm-identifier>

(107) <pgm-identifier>::=<identifier>

(108) <invoke>::=<cond>INVOKE <pgm-name> <invoke-time>

(109) <invoke-time>::=<access-cond-evaluation>THE ACCESS CONDITION OF <franchise-name>/AT LOGIN

(110) <access-cond-evaluation>::=BEFORE EVALUATING/ REGARDLESS OF/ WHEN SATISFYING/ WHEN NOT SATISFYING

(111) <pgm-remove>::=DISCONNECT <pgm-name> FROM <franchise-name>

(112) <pgm-delete>::=DELETE <pgm-name>

(113) <pgm-test>::=TEST <pgm-name>

(114) <extend-stmt>::=<extend>;/<extend-remove>;/<extend-display>

(115) <extend>::=EXTEND THE RESOURCE UNIT OF <franchise-name> TO <franchise-name>

(116) <extend-remove>::=REMOVE THE EXTENSION OF THE RESOURCE UNIT OF <franchise-name> TO <franchise-name>
APPENDIX C
SAMPLE PROTECTION LANGUAGE SYNTAX

(117) <extend-display> ::= DISPLAY THE EXTENSIONS OF THE RESOURCE UNIT OF <franchise-name>

(118) <semantic-stmt> ::= SELECT SEMANTIC PARAMETERS TO BE:
                     <sys-par-assign-list> END;

(119) <sys-par-assign-list> ::= <sys-par-assign> /
                         <sys-par-assign>, <sys-par-assign-list>

(120) <sys-par-assign> ::= <sys-par-assign>, <sys-par-assign> /
                         USER OPERATION <logical-op> /
                         RESOURCE OPERATION <logical-op> /
                         REJECTION <rejection> /
                         SYSADM OWNS <sysowns> /
                         LOCK-UNLOCK <answer> /
                         EXTENDED RESOURCES <answer> /
                         EXTENDABLE RESOURCES <proc-list>

(121) <logical-op> ::= and / or

(122) <rejection> ::= partial / full

(123) <sysowns> ::= system files / all

(124) <answer> ::= yes / no

(125) <proc-list> ::= <sys-proc> / <sys-proc>, <proc-list>

(126) <sys-proc> ::= system procedures

(127) <comment-stmt> ::= *<text>*

(128) <text> ::= any string of characters including blanks

(129) <identifier> ::= an authorizer supplied ASCII string of up to 32 characters

(130) <sys-proc-name> ::= THE SYSTEM PROCEDURE NAMED <sys-proc>

(131) <franchise-resource> ::= THE FRANCHISES REFERRING TO <object-name>

(132) <franchise-display> ::= <franchise-ref> TO <object-name>
In chapter IV the reader is referred to this appendix for a more formal statement of the authorization and enforcement processes. The validation rules of (4.1) can be restated, for application in state $s_I$ to an authorization $p = (a, U, E, R, S)$, as follows:

1) $\text{OWN } \in E(p) \implies a(p) = \text{SYSADMIN} \land d(\text{SYSADMIN}, \text{OWN}, R(p), s_I)$

2) $\text{SUBOWN } \in E(p) \implies d(a(p), \text{OWN}, R(p), s_I)$

3) $E(p) \subseteq [E - \{\text{OWN, SUBOWN}\}] \implies d(a(p), \text{OWN}, R(p), s_I) + d(a(p), \text{SUBOWN}, R(p), s_I)$

The Authorized Subspace for $n$ Authorizations

The expression given in (4.6) characterizes the resource and state projection of the authorized subspace of two authorizations matching in $a, U,$ and $E$. The general case for $n$ authorizations is developed briefly here for completeness. The authorized subspace is determined by the union of the parts found in each of the following steps.

1) Each of these cartesian products in the following union uses the resources specified in only one resource unit,
R[i], of the n authorizations, and therefore is subject to only one access condition—that of S[i].

\[
\bigcup_{i=1}^{n} \bigcup_{j=1}^{n} (R[i] \cap R[j]) \times S[i]
\]

for \( j < i \).

2) Each of the cartesian products in this union represents resources common to two resource units, \( R[i] \) and \( R[j] \), and therefore subject to conditions corresponding to both \( S[i] \) and \( S[j] \):

\[
\bigcup_{j=1}^{n-1} \bigcup_{i=1}^{n} (R[i] \cap R[j]) \times (S[i] \cap S[j])
\]

for \( j < i \).

3) Each of these cartesian products indicates the resources common to \( R[i] \), \( R[j] \), and \( R[k] \), subject to the conditions of \( S[i] \), \( S[j] \), and \( S[k] \):

\[
\bigcup_{k=1}^{n-2} \bigcup_{j=1}^{n-1} \bigcup_{i=1}^{n} (R[i] \cap R[j] \cap R[k]) \times (S[i] \cap S[j] \cap S[k])
\]

for \( i < j < k \), \( j < k < i \), and \( k < i \).

4) And so on, for resources common to four, five, etc. units up to

\[
(R[R_1 \cap R_2 \cap \ldots \cap R_n]) \times (S[S_1 \cap S_2 \cap \ldots \cap S_n])
\]

the cartesian product corresponding to resources common to all \( n \) of these authorizations.
The basic memoryless case of the enforcement process, as developed in chapter IV, is formalized as follows.

1) \( F(u) = \{ p \in \mathbb{P} : u(q) \in U(p) \} \)
2) \( F(e) = \{ p \in \mathbb{P} : e(q) \in E(p) \} \)
3) \( F(R) = \{ p \in \mathbb{P} : R(q) \subseteq R(p) \cup \emptyset \} \)
4) \( U(q) = F(u) \cap F(e) \cap F(R) \)
5) \( R^* = \bigcup_{i} R(p[i]) : p[i] \in U(q) \)
6) If any \( R \in R^* \) is locked, set \( d(q) = 0 \) and halt.
7) If \( R(q) \notin R^* \), set \( d(q) = 0 \) and halt.
8) Let \( R^* = \{ R_1, R_2, \ldots, R_n \} \). Partition \( D(q) \) into equivalence classes such that the \( i \)th class is:
   \( D[i](q) = \{ p \in U(q) : R(p) \cap R[i] \} \)
   for each \( R[i] \in R^* \). Construct a temporary representative authorization for each class, \( D[i](q) \):
   \( p[i] = (a, u(q), e(q), R[i], U_S(p[j] \in D[i](q))) \)
   Then \( F(u,q) = \bigcup_{i} p[i] \).
9) \( EAC = C(s, \bigcap_{i} A[p[i]]) \)
10) Set \( d(q) = EAC \) and halt.
APPENDIX E

EXAMPLES OF CONDITIONS

Since very little has been said about implementation related matters, the reader will have to exercise some imagination in this appendix, particularly with regard to the variable names used in the conditions and how their values are determined by the protection system.

User Group Defining Conditions

User groups can be defined in two ways. The first way, a simple enumeration of a list of identifiers (names or identification numbers), does not require the use of conditions. The examples in this section illustrate the second way, the use of conditions to define user groups. The simplest of these definitions is the one for a standing group called general. Its defining condition is:

\[ C(u, \text{general}) = \text{true} \]

Thus, every user is implicitly a member of the general group, sharing its minimum level of privileges.

The MULTICS system is one of the few which allows for groups of users which may have varying membership [SALTJ74]. However, in the MULTICS system this feature is greatly circumscribed. In that system the principal identifier of each user indicates his individual identification, his membership in projects, and the compartments in which he is authorized to operate. The compartment concept is a way to group users into logical subsets by a naming scheme. However, it lacks the ability to define logical subsets of users by condition instead of just by enumeration. The
model presented here provides this ability by the use of a condition. Let the assumption be made that this system has a file of user identifications which is consulted when a user logs into the system. Suppose that this information about the user, is found in his record: name, password, department number, project number, classification code, and organizational affiliation. Suppose also that during the login these items are added: terminal number, time of login, and the subsystem (software processor) specified for use in the login. The following are typical of the user group defining conditions which can be dynamically evaluated.

\[
\begin{align*}
C(u,U1) & : \text{dept}=13 \land \text{project}=\text{design} \\
C(u,U2) & : \text{classcode}=ab.x \land \text{organ}=\text{acm} \\
C(u,U3) & : \text{terminal}=42 \\
C(u,U4) & : \text{subsyst}=\text{fortran}
\end{align*}
\]

When any user logs in, if he is either in department 13 or part of the design project, he becomes a member of user group U1 for a certain period of time. If he is using terminal number 42, he also becomes a temporary member of U3, and so on. If he logs in to use FORTRAN, he will further be a member of U4 and subject to its access restrictions. Of course, these conditions need not be limited to information directly connected with the user's identification record. For example, the group of all users logging in during the morning are defined by:

\[
C(u,U5) : \text{time} < 12:00
\]

where \(\text{time}\) is the system time-of-day clock and "12:00" denotes noon.

Resource Unit Defining Conditions

One of the most challenging difficulties within an implementation of any model of protection is to provide resource definitions (data descriptions) that can be translated accurately and unambiguously in a convenient and economical way. A general solution to this problem is not claimed here. The naming of resources (especially data elements and sets of data elements), whether for access or for access control, is very dependent upon the basic structure of the system environment. It is beyond the scope
of this work to survey the typical structures currently used in data bases [LEFTNOY, DATEC75], but the basic classes represented are: hierarchical, network and relational structures. The Information Management System of IBM [INTER73] is a well known example of the tree-structured hierarchical approach and the work of the CODASYL Data Base Task Group [CODAS71] typifies the network approach. More recently, the relational model of data [CODDE70] has received a great deal of attention. Clearly the problem of data description for protection is precisely the problem of data description for storage and retrieval in a data base system. A great deal of work by many researchers and designers has been expended on this problem; see section 2 of [SIBLE73]. Their results are equally applicable to the counterpart problem in protection. For example, in a relational data base environment sublanguages, such as SEQUEL [ASTRM74] or SQUARE [BOYCH75], for expressing queries may be used to express resource definitions for protection. Also, any graphical aids, such as CUPID [McDON74], for query formulation are applicable. All expressible relations—constructed of named relations, generation names, roles, domains, logical connectives, quantifiers, and constant relations [CODDE70]—are valid descriptions of data to be protected. In most real systems it will often be desirable to use a combination of explicit naming and implicit conditional definitions. The explicit naming, such as by providing a file name, describes a domain in which the condition is to be applied.

Since the details of the naming process are dependent on the data structures, examples are chosen to be in the context of a specific organization. The attribute-value based structure [HSIAD70], the structure for which the model will be first implemented (see chapter VI), has been chosen. The attribute-value model of a data base does not fall exactly within one of the three classes of structures previously mentioned. It has similarities to network organizations with regard to implementational structure and some similarities with the relational approach in its logical characteristics. The records of attribute-based data systems are composed of attribute-value pairs. The attribute is a field name with which the value is associated. A keyword is an attribute-value pair which can be used for direct retrieval of data (e.g., name=jones or color=blue). Retrieval requests are expressed in queries, logical expressions of keywords used to access logical subsets of the data base. In a system where all attributes are allowed in keywords, subsets of data can be defined for protection exclusively by queries. A query is, in fact—by the definition in (2.2)—a condition, a resource unit
defining condition. In most real systems, however, a query is inadequate to logically describe all subsets of data for protection. This is because there usually are attribute-value pairs not used as keywords. For example, consider a personnel file. The chances of the salary figures for two or more employees being exactly the same are very low. Therefore, there will be almost as many different values in the salary field of a personnel record as there are employees. This makes the salary attribute a poor choice for use as a keyword. Since an index entry is often required for each keyword, the index would become overcrowded. Nonetheless, from a protection point of view, the salary field may be very important. For example, it may be desirable to identify a resource unit by this condition:

\[ C(r,R1) : \text{salary} < 20000 \]

Since salary cannot easily be a keyword, some processing of each record must be done subsequent to retrieval but prior to passing the results to the user in order to evaluate the above condition. If the value of salary turns out to be less than 20000 at the time of evaluation for a record \( r \), then \( r \) is defined to be a member of \( R1 \). In general, a resource unit defining condition may have both keyword and non-keyword terms. Supposing that dept is a keyword attribute, the following is an example of such a mixture:

\[ C(r,R2) : \text{dept}=7 \& \text{salary} < 20000 \]

An attempt has been made to make the use of keywords more efficient in access control by a concept called security keywords [MCCAEE75]. Here, a few of the attributes used in keywords are allowed in the naming of record aggregates for protection. In a system for which very few changes in protection requirements are anticipated, this approach enhances the efficiency. However, there are drawbacks. Now, one can define even fewer (by orders of magnitude) data subsets than one could by using keywords. The security attributes must be chosen \textit{a priori}, an impossible task in many systems. As soon as a list is decided upon, there always seem to be changes that are necessary. Also, the definitions for many kinds of subsets (a particular range of values being allowed, for example) must be "rigged" by adding a new attribute-value pair to each record as a switch to tell whether another attribute's value is within the stated range. If the range is to be changed, the entire data base must be reprocessed. If there is a change in the attribute used, the entire data base must be reorganized. This practice amounts to embedding security
policy in the data itself. Thus, security keywords are limited in the resources they can define. However, when an attribute-based system does employ security keywords, use can be made of them in conjunction with other definitions, thus adding to efficiency wherever possible, but remaining responsive to broader requirements.

Next, a user record template [HSIAD71] may be used as a part of a resource definition in an attribute-based system. A record template, a central description of the record organization of a file, requires only values and not the attributes to be stored in the records themselves. A user template describes those fields in a record as seen from that user's own point of view. If an attribute doesn't exist in a user's template, he has no means to access the corresponding field of any record. Thus appears the possibility of a field-naming modifier in a resource unit description. And, as previously mentioned, a name (such as a file name) can be used to name a domain to which the rest of the description is restricted. Each of the different parts of a resource unit definition are subject to different types of processing depending on the associated structures in the data base system. A general resource definition in an attribute-value based system has the following components and associated types of processing.

<table>
<thead>
<tr>
<th>component</th>
<th>type of processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>processed like retrieval keywords, before retrieval</td>
</tr>
<tr>
<td>security keywords</td>
<td>processed as part of retrieval</td>
</tr>
<tr>
<td>non-keyword attribute-value pair</td>
<td>processed per record after retrieval</td>
</tr>
<tr>
<td>field-name modifier</td>
<td>part of user template</td>
</tr>
<tr>
<td>file-name modifier</td>
<td>high level naming process, before retrieval or keyword processing</td>
</tr>
</tbody>
</table>

Table E.1 - Parts of a Resource Unit Definition
EXAMPLES OF CONDITIONS

for an Attribute-Based System

Here is an example of a resource unit defining condition with all of these parts, except security keywords:

\[ C(r,R3) : \text{file=} \text{personnel} \& \text{dept}=7 \& \text{salary} < 20000 \& (\text{field=} \text{name} + \text{field=} \text{address}) \]

This condition defines the set which is the name or address fields of all records in the personnel file for which dept=7 and salary < 20000.

Access Conditions

Although any condition can be used for defining any of the sets, access conditions, provisos which must be met before certain accesses are allowed, are generally not directly associated with users or data. Access conditions are likely to be associated with global system variables such as the time-of-day clock, modes of operation, status indicators, flags, and internal codes. Many of these indicators contain information deduced by the monitor or data base system about the current transaction. Also, occasionally, data content dependent conditions are more appropriate as access conditions than as part of resource definitions.

In the first example let it be assumed that there is a system status indicator called \text{mode}. There are two modes: regular and master-update. A certain field, perhaps containing a security keyword, of each of the records in a certain file can be changed only while the system is in the master-update mode. The corresponding access condition is simply:

\[ C(s,S1) : \text{mode=} \text{master-update} \]

Although the validity checking of input data is not a security function, input checks crucial to data integrity can be included in the access control rules. If, for example, the value of "field1" is expected to be between zero and ninety nine, one could use this access condition for the input operation:

\[ C(s,S2) : \text{field1} < 100 \]
EXAMPLES OF CONDITIONS

\[ C(s,S2) : \text{fieldl} > 0 \land \text{fieldl} \leq 99 \]

Of course, in this case the appropriate resource unit definition will be required to refer to the input record.

Suppose that certain accounting information can be entered only on Fridays. For 1975, this is represented by the following access condition:

\[ C(s,S3) : \text{MOD}(\text{day},7)=3 \]

where \( \text{MOD} \) is the modulo function and \( \text{day} \) is the day number (a number from 1 to 365).

And, of course, there must be a way to represent the two unconditional cases. Access under all conditions (always) is simply:

\[ C(s,S4) : \text{true} \]

which defines the entire set \( S \). No access, or access under no conditions (never), is indicated by:

\[ C(s,S5) : \text{false} \]

which defines the null set.

Other examples of conditions used in various ways may be found in sections on authorization specifications, history keeping, and auxiliary procedures in chapter III.

Classes of Conditions

The examples in the previous section were to show how conditions typically are used. Within the model there are no restrictions on what type of variables can be used in any of the conditions. For example, it is possible to define a user group in terms of the records retrieved, by this condition:

\[ C(u,U5) : \text{file}=\text{part} \land \text{color}=\text{blue} \]

Under this definition, if the value of the \textcolor{blue}{color} field is blue for a record that \( u \) retrieves from the part file, then \( u \) becomes a (temporary) member of \( U5 \). Many capabilities
like this are really redundant, since this definition can be more naturally given as either a resource definition or an access condition. Thus, it is perfectly reasonable to restrict each condition type to certain classes of variable types in a specific implementation. The above user group definition, for example, would be inconvenient to implement in most systems, since group membership then could not be determined until after the data is accessed. Furthermore, such a definition does not add anything that couldn't be expressed a better way. Some examples of these classes are as follows:

1) Resource-related
   a. Names—evaluated early by templates and simple tables
   b. Logical—evaluated after access to indexes and directories, but before access of data
   c. Data content-dependent—not evaluated until after final access of data
   d. Input content-dependent—a function of user's input (for writing and updating operations)

2) Dynamic
   a. System variables—clocks, indicators, mode switches, flags
   b. Access history information—defined later, included here for completeness
   c. Parameters passed to and from procedures used as extended resources

3) User-related
   a. User identification information
   b. Login associated system information
The Relation Between States and Conditions

Subspaces of the security space are determined by subsets of points (or intervals) on each of the five axes of the space. Of special interest are subsets of points along the state axis. For example, figure F.1 names all the possible states in a system with three binary-valued resources: r1, r2, and r3. In figure F.2., on the corresponding state axis of the security space, there are (for example) two state subsets, S1 and S2. Each one is a subset containing some points on the $S$ axis. In this case, two subsets overlap, having two states in common. This interval of overlap corresponds to common resource values shared by the two state subsets; namely, s2 and s3 share the common values of $v(r_1) = 0$ and $v(r_2) = 1$; therefore, $C(s_2, S_1 \cap S_2) = \{v(r_1)=0 & v(r_2)=1\}$.

In any real data base system, however, the sheer size of the number of resources and their possible values implies that the enumeration of the states in various subsets is impossible. Thus, a descriptive definition of the subsets is employed rather than relying on enumeration. Such descriptive definitions are known as conditions, which were defined in (2.2). A language at any level, in order to mention state subsets, will use conditions for that purpose. However, since the security space is based on states, enumeration of states is occasionally used to indicate points in the security space for some examples in this dissertation.
### Figure F.1 - States of a Binary Three-Resource System

<table>
<thead>
<tr>
<th>State</th>
<th>Resource Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$s_1$</td>
<td>0 0 1</td>
</tr>
<tr>
<td>$s_2$</td>
<td>0 1 0</td>
</tr>
<tr>
<td>$s_3$</td>
<td>0 1 1</td>
</tr>
<tr>
<td>$s_4$</td>
<td>1 0 0</td>
</tr>
<tr>
<td>$s_5$</td>
<td>1 0 1</td>
</tr>
<tr>
<td>$s_6$</td>
<td>1 1 0</td>
</tr>
<tr>
<td>$s_7$</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

### Figure F.2 - Overlapping of State Subsets
The Relation Between Authorizers and Users—The Valid Subspace

For the purpose of illustrating the relationships of the projections, this section will simplify the ownership rules to exclude the concepts of subownership and the SYSADMIN leaving e[l] (=OWN) as the only ownership related operation of interest. The reader can easily extend these notions to the full concept of ownership as it is discussed in the section on validation in chapter IV. With this simplification, then, for an authorization to be valid, the authorizer must own all data elements contained in the resource unit of the authorization. Consider the following authorization specification:

\[ p1 = (a1, (u2), (Read), (r1, r2, r3), (s7)) \]

issued in state s4.

To illustrate the security space, projections are required to reduce the number of dimensions down to a visualizable few. Furthermore, in each two-dimensional projection, a third dimension can be visualized by labelling and correlating points in the projections with respect to that dimension. The relationship between authorizers and users regarding the question of specification validity cannot be seen in a single projection of authorizers and users. Thus, several other projections must be examined. In figure F.3, the projection shows that a1, as an authorizer, has assigned u2, as a user, the right to read some resources. Figure F.3 does not imply, for example, that u2 is an authorizer. The question of the specification validity may be rephrased as: "Can a1 validly assign rights to access this resource to any user?" In other words, the question is: "Does a1 own the resource under consideration?" Figure F.3 cannot show the resource unit; another projection is needed. In figure F.4 it is learned that the resource unit R1 is (r1, r2, r3). The information in figure F.4 indicates only that a1 (also called u3, since A∈U) owns R1 sometimes. For an authorization to be valid, this ownership must prevail at the moment when the authorization is being processed. This is the moment that the system is in the state s4. Moving to figure F.5, but staying in the plane of figure F.4, it can be seen that this ownership relation holds for each state in S1. Thus, in state s4, a1 can validly assign rights to R1, and,
APPENDIX F
THE ROLE OF BASIC AUTHORIZATION

Figure F.3 - The Security Space Projection of Authorizers and Users

Figure F.4 - The Security Space Projection of Users and Resources
Figure F.5 - The Security Space Projection of Resources and States
therefore, the point in figure F.3 is said to be in the valid subspace of the security space. It may be noted that the ownership is required only at the time the authorization is issued. The authorizer al is not required to own R1 in s7, for example, for u2 to have READ access to R1 in s7, after the authorization has been issued and accepted in s4. Figure F.6 is a three dimensional composite of figures F.4 and F.5. This view illustrates that the prevailing ownership relation determines the validity of an authorization specification.

The Relation Between Users and User Groups—One

View of the Authorized Subspace

The relation between a user and the groups of which he is a member is of great importance in this model of access control. The following are valid authorizations:

\[ p_1 = (a, U_1, \{\text{READ}\}, \{r_1\}, \{s_1\}) \]

where \( U_1 = \{u_1, u_2, u_3\} \) and

\[ p_2 = (a, U_2, \{\text{READ}\}, \{r_1\}, \{s_2\}) \]

where \( U_2 = \{u_3, u_4, u_5\} \). The projection of users and resources in figure F.7 illustrates \( p_1 \) and \( p_2 \) as issued by authorizer a. As in the previous examples, simple state subsets are used to represent the access conditions. Figure F.8 depicts the first authorization in the user and state projection, indicating that any member of user group \( U_1 \) may read the resource \( r_1 \) when the system is in state \( s_1 \). Figure F.9 illustrates the corresponding situation for \( U_2 \) and \( s_2 \). The combined effect in the user and state projection of the two authorizations is shown in figure F.10. In this figure the user of prime interest is \( u_3 \). The points in his projection show that \( u_3 \) may read \( r_1 \) in either state \( s_1 \) or state \( s_2 \). Since \( u_3 \) is a member of two groups, he has the access rights of both groups. That is, his effective subset of states for access is \( (s_1) \cup (s_2) \) or \( (s_1, s_2) \). In terms of access conditions his effective access condition to \( r_1 \) is the logical OR of the separate access conditions specified for groups \( U_1 \) and \( U_2 \) with respect to \( r_1 \). Thus, a user in
Figure F.6 - The Security Space Projection of Users, Resources, and States
Figure F.7 - The Projection of Users and Resources

Figure F.8 - The User and State Projection for User Group U1
APPENDIX F
THE ROLE OF BASIC AUTHORIZATION

Figure F.9 - The User and State Projection for User Group U2

Figure F.10 - The User and State Projection for User Groups U1 and U2
only group U1 must satisfy one condition to access rl and a user in only U2 must satisfy another. But, a user in both groups may access rl by satisfying either condition.

The collection of authorization specifications received prior to a given time corresponds to a security subspace which is named the authorized subspace. Thus, at a given time, the authorization specifications partition the security space into a subspace which is authorized for access and a subspace which is not. For example, all of the points in the figures of this appendix which are labelled with access operations are in the authorized subspace and are due to various authorization specifications.

At the beginning, the SYSADMIN is alone in the authorized subspace. Typically, he enters authorization specifications to share his ownership of various parts of the data base with other users. Each new owner continues to add points to the authorization subspace by introducing authorizations to allow access to his data by still more users. New owners are added as users create data. Occasionally, access privileges are withdrawn or access restrictions are made more stringent, thereby reducing the authorized subspace. Thus, the shape of the authorized subspace is changeable.

Like the authorization specifications, access requests also determine subspaces in the security space. The task of the enforcement process is to allow only those requests whose subspaces are completely contained in the corresponding authorized subspace. This process is illustrated in the section of examples at the end of chapter IV.

The Relation Between Individual Resources and Resource Units—Another View of the Authorized Subspace

In this section the following authorizations are considered:

\[ p_1 = (a, (u1), (WRITE), R1, S1) \]

where \( R1 = (r1, r2, r3) \) and \( S1 = (s1, s2) \) and
p2 = (a,(u1),(WRITE),R2,S2)

where R2 = (r2,r3,r4) and S2 = (s2,s3). In figure F.11 the result of p1 is seen in the user and resource projection. Figure F.12 illustrates the effect of p2 in the same projection.

The effect of p1 alone in the resource and state projection is given in figure F.13. Authorization p2 alone results in the arrangement shown in figure F.14.

Figures F.13 and F.14 state that u1 must be in a state of S1 to access any element of R1 and in a state of S2 to access an element of R2. Therefore, for u1 to access an element which is in both R1 and R2, he must be in a state which is in both S1 and S2. In other words, to write in R1 ∩ R2 (i.e., (r2,r3)) u1 must be in a state of S1 ∩ S2 (i.e., (s2)). This requirement is illustrated in figure F.15, showing the combined effect of p1 and p2. Here, it can be seen (the circle) that the subset of states corresponding to access of r2 or r3 is S1 ∩ S2 (i.e., (s2)). In terms of conditions, u1's effective access condition to either r2 or r3 is the logical AND of the separate conditions specified for units R1 and R2, i.e., the AND of the conditions corresponding to S1 and S2. The points in figure F.15 represent a projection of the authorized subspace as specified in p1 and p2.

For more examples of how the authorization and enforcement processes interact in the security space, see the end of chapter IV.
APPENDIX F
THE ROLE OF BASIC AUTHORIZATION

Figure F.11 - p1 in the User and Resource Projection

Figure F.12 - p2 in the User and Resource Projection
APPENDIX F
THE ROLE OF BASIC AUTHORIZATION

Figure F.13 - p1 in the Resource and State Projection

Figure F.14 - p2 in the Resource and State Projection
Figure F.15 - $p_1$ and $p_2$ in the Resource and State Projection
APPENDIX G

THE ENFORCEMENT PROCESS WITH VARIABLE SEMANTICS

If it is desirable to change the semantic parameters within the life of a given implementation, it is possible to make the enforcement process dynamically sensitive to these parameters, instead of fixing them for each implementation. This appendix indicates how a few of the example semantic parameters given in chapters V and VI can be made to affect directly the full-specification description of the enforcement process from chapter IV. Other parameters are possible. The sequence of steps is as follows:

1) Determine the franchise of the user, as before. If EXTENDED RESOURCES = yes
   Then do
      If requested resources include an EXTENDABLE RESOURCE
         Then do
            Follow all links to find authorizations for the extended resources and include these in F(u), so that the additional access rights are used for this particular access decision.
   End
End

2) Determine the franchise of the operation, F(e).

3) Determine the franchise of the resource, F(R).

4) Compute D(q), the domain of authorization, the intersection of the franchises found in 1), 2), and 3).

5) Determine the resource reference, R*, as before.

6) If LOCK/UNLOCK = yes
   then do
      If any elements of the resource reference are locked
Then do
Deny the request by setting \( d(q) = 0 \)
Halt
End
End

7) Determine covering of the request by \( R^* \)
   If covering is not complete
      Then do
         If \( \text{REJECTION} = \text{full} \)
            Then do
               Deny the request by setting \( d(q) = 0 \)
               Halt
            End
         Else do
            Modify the requested resource per equation (4.9)
            End
      End
   End

8) Compute the domain of auxiliary invocation, the set of all auxiliary invocation triples associated with elements of \( D(q) \). Invoke all programs for which \( b[v]=0 \) and \( C(s,S[v]) \) holds. Recall that the results of these procedures can be passed to access conditions via system counters and other system variables.

9) Partition \( D(q) \) into equivalence classes based on common resource units and take the <EP USERS OPERATION> of access conditions within each class to produce a single effective authorization representing the class. The set of representative authorizations is the effective user franchise, \( F(u,q) \).

10) Compute and evaluate the effective access condition by performing the <EP RESOURCES OPERATION> of the access conditions in \( F(u,q) \) and render the corresponding access decision, \( d(q) \).

11) If history keeping is requested, increment the specified counters, \( k \), per the flags, \( b[h] \).

12) If auxiliary invocations are required, invoke all programs for which \( b[v]=1 \) and \( C(s,S[v]) \) holds. If the access decision from step 10) above is \( d(q)=1 \), execute all auxiliary programs for which \( b[v]=2 \) and for which \( C(s,S[v]) \) holds. If the decision is \( d(q)=0 \), execute all programs having \( b[v]=3 \) and for which \( C(s,S[v]) \) holds.
13) If \( d(q) = 1 \), complete the logical access operation by allowing the data base system to perform \( e(q) \) on \( R(q) \) for \( u(q) \).
CITED REFERENCES

ARNSC75 Arnst, Catherine, "DPMA President Urges Societies to Help Develop Privacy Laws," Computerworld (March 26, 1975), 9.


REFERENCES


CODASYL Data Base Task Group, Report to the Programming Language Committee, (Rev. April 1971)


Cohen, Edward, graduate student in the Department of Computer and Information Science, The Ohio State University, pointed out the problem of dominance as applied to history keeping mechanisms of access control, in data security project meetings (1974).


Date, C.J., An Introduction to Database Systems, Addison-Wesley (1975).


REFERENCES


FABRR74 Fabry, R. S., "Capability-Based Addressing," Comm. of the ACM 17, 7 (July 1974), 403-412.


FRENN75 French, Nancy, "Privacy Act of 1974 Marks 'End of Beginning'," Computerworld (March 26, 1975), 36.


REFERENCES


HANLJ74 Hanlon, Joseph, "UK White Paper Due This Month, May Disappoint Privacy Backers," Computerworld (December 11, 1974), 7.


HENNJ75 Hennings, James, M.S. Thesis, Department of Computer and Information Science, The Ohio State University (expected 1975).


REFERENCES


<table>
<thead>
<tr>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LAMPB73</strong> Lampson, Butler W., &quot;A Note on the Confinement Problem,&quot; <em>Comm. of the ACM</em> 16, 10 (October 1973), 613-615.</td>
</tr>
<tr>
<td><strong>LEDGH71</strong> Ledgard, Henry F., &quot;Ten Mini-Languages: A Study of Topical Issues in Programming Languages&quot;, <em>Computer Surveys</em> 3, 3 (September 1971), 115-146.</td>
</tr>
<tr>
<td><strong>MCDON74</strong> McDonald, Nancy, and Stonebraker, Michael, &quot;CUPID—The Friendly Query Language,&quot; Memorandum No. ERL-M487, College of Engineering, University of California, Berkeley (October 1974).</td>
</tr>
</tbody>
</table>
REFERENCES


PODGC73 Podgus Christopher, "Outwitting the Computer Swindler," *Computer Decisions* (September 1973), 12-16.


REFERENCES


WONGD75  Wong, David, M.S. Thesis, Department of Computer and Information Science, The Ohio State University (expected 1975).


INDEX

An index to this dissertation is given in the following pages. Index terms were assigned manually, but a computer program produced the two-at-a-time term coordination and formatted the output.
<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Process</td>
<td>14, 14, 14, 28</td>
</tr>
<tr>
<td>Ownership</td>
<td>14</td>
</tr>
<tr>
<td>Policy</td>
<td>28</td>
</tr>
<tr>
<td>Processes</td>
<td>14</td>
</tr>
<tr>
<td>Security Policy</td>
<td>28</td>
</tr>
<tr>
<td>Validity</td>
<td>14, 28</td>
</tr>
<tr>
<td>Authorization Specification</td>
<td>27, 30, 32, 61</td>
</tr>
<tr>
<td>Access Request</td>
<td>30</td>
</tr>
<tr>
<td>Auxiliary Program Invocation</td>
<td>27</td>
</tr>
<tr>
<td>Basic Specification</td>
<td>27</td>
</tr>
<tr>
<td>Definition</td>
<td>30</td>
</tr>
<tr>
<td>Examples</td>
<td>32</td>
</tr>
<tr>
<td>Extended Resources</td>
<td>27</td>
</tr>
<tr>
<td>Full Specification</td>
<td>27</td>
</tr>
<tr>
<td>Generalized</td>
<td>61</td>
</tr>
<tr>
<td>History Keeping</td>
<td>27</td>
</tr>
<tr>
<td>N-Tuples</td>
<td>61</td>
</tr>
<tr>
<td>Authorized Subspace</td>
<td>62, 162, 178, 182</td>
</tr>
<tr>
<td>Definition</td>
<td>62</td>
</tr>
<tr>
<td>Examples</td>
<td>62, 178, 182</td>
</tr>
<tr>
<td>Formalized</td>
<td>162</td>
</tr>
<tr>
<td>Generalized</td>
<td>162</td>
</tr>
<tr>
<td>Validation Process</td>
<td>162</td>
</tr>
<tr>
<td>Validity</td>
<td>162</td>
</tr>
<tr>
<td>Authorizers</td>
<td>19</td>
</tr>
<tr>
<td>Model</td>
<td>19</td>
</tr>
<tr>
<td>Sets</td>
<td>19</td>
</tr>
<tr>
<td>User Groups</td>
<td>19</td>
</tr>
<tr>
<td>Users</td>
<td>19</td>
</tr>
<tr>
<td>Auxiliary Program Invocation</td>
<td>27, 43, 44, 45</td>
</tr>
<tr>
<td></td>
<td>46, 47, 48, 49</td>
</tr>
<tr>
<td></td>
<td>74, 125</td>
</tr>
<tr>
<td>Activation Sequence</td>
<td>44</td>
</tr>
<tr>
<td>Alarm and Recovery</td>
<td>46</td>
</tr>
<tr>
<td>Applications</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>Audit</td>
<td>46</td>
</tr>
<tr>
<td>Authentication</td>
<td>45</td>
</tr>
<tr>
<td>Authorization Specification</td>
<td>27</td>
</tr>
<tr>
<td>Basic Specification</td>
<td>27</td>
</tr>
<tr>
<td>Conditions</td>
<td>44</td>
</tr>
<tr>
<td>Covering of Request</td>
<td>74</td>
</tr>
<tr>
<td>Domain of Authorization</td>
<td>74</td>
</tr>
<tr>
<td>Domain of Auxiliary Invocation</td>
<td>74</td>
</tr>
<tr>
<td>Examples</td>
<td>48, 49, 125</td>
</tr>
<tr>
<td>Extended Resources</td>
<td>27, 74</td>
</tr>
<tr>
<td>Franchise</td>
<td>74</td>
</tr>
<tr>
<td>Full Specification</td>
<td>27</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>AUXILIARY PROGRAM INVOCATION</td>
<td>27</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>74</td>
</tr>
<tr>
<td>INPUT DATA VALIDATION</td>
<td>47</td>
</tr>
<tr>
<td>INVOCATION CONDITION</td>
<td>125</td>
</tr>
<tr>
<td>JOURNALING</td>
<td>46</td>
</tr>
<tr>
<td>LOGIN PROGRAM</td>
<td>46</td>
</tr>
<tr>
<td>MULTICS</td>
<td>44</td>
</tr>
<tr>
<td>ON-CONDITION</td>
<td>48</td>
</tr>
<tr>
<td>PARTITIONING</td>
<td>74</td>
</tr>
<tr>
<td>PROGRESSIVE AUTHORIZATION</td>
<td>48</td>
</tr>
<tr>
<td>SURVEILLANCE</td>
<td>47</td>
</tr>
<tr>
<td>THREAT MONITORING</td>
<td>47</td>
</tr>
<tr>
<td>TRAIL</td>
<td>46</td>
</tr>
<tr>
<td>TRIPLE</td>
<td>44</td>
</tr>
<tr>
<td>USER IDENTIFICATION</td>
<td>45</td>
</tr>
<tr>
<td>BACKUS-NAUR-FORM</td>
<td>150</td>
</tr>
<tr>
<td>SAMPLE PROTECTION LANGUAGE</td>
<td>150</td>
</tr>
<tr>
<td>SYNTAX</td>
<td>150</td>
</tr>
<tr>
<td>BASE LANGUAGE</td>
<td>96</td>
</tr>
<tr>
<td>COMMON BASE LANGUAGE</td>
<td>96</td>
</tr>
<tr>
<td>SEMANTIC BASE</td>
<td>96</td>
</tr>
<tr>
<td>BASIC AUTHORIZATION, SEE: BASIC SPECIFICATION</td>
<td>173</td>
</tr>
<tr>
<td>BASIC MEMORYLESS CASE</td>
<td>164</td>
</tr>
<tr>
<td>ALGORITHM</td>
<td>164</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>164</td>
</tr>
<tr>
<td>FORMALIZED</td>
<td>164</td>
</tr>
<tr>
<td>BASIC SPECIFICATION</td>
<td>173</td>
</tr>
<tr>
<td>ALGORITHM</td>
<td>27</td>
</tr>
<tr>
<td>AUTHORIZATION SPECIFICATION</td>
<td>72</td>
</tr>
<tr>
<td>AUXILIARY PROGRAM INVOCATION</td>
<td>27</td>
</tr>
<tr>
<td>COVERING OF REQUEST</td>
<td>72</td>
</tr>
<tr>
<td>DOMAIN OF AUTHORIZATION</td>
<td>72</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>72</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>173</td>
</tr>
<tr>
<td>EXTENDED RESOURCES</td>
<td>27</td>
</tr>
<tr>
<td>FRANCHISE</td>
<td>72</td>
</tr>
<tr>
<td>FULL SPECIFICATION</td>
<td>27</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>27</td>
</tr>
<tr>
<td>RESOURCE REFERENCE</td>
<td>72</td>
</tr>
<tr>
<td>SECURITY SPACE</td>
<td>173</td>
</tr>
<tr>
<td>BOOLEAN EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>COMPARISON EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>25</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>25</td>
</tr>
<tr>
<td>EVALUATION OF CONDITION</td>
<td>25</td>
</tr>
<tr>
<td>SET MEMBERSHIP</td>
<td>25</td>
</tr>
<tr>
<td>CAPABILITY LIST</td>
<td>9</td>
</tr>
<tr>
<td>ACCESS MATRIX</td>
<td>9</td>
</tr>
<tr>
<td>Term</td>
<td>Pages</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>MULTICS</td>
<td>96</td>
</tr>
<tr>
<td>CLASSES OF CONDITIONS</td>
<td>171</td>
</tr>
<tr>
<td>COMMON BASE LANGUAGE</td>
<td>96</td>
</tr>
<tr>
<td>BASE LANGUAGE</td>
<td>96</td>
</tr>
<tr>
<td>SEMANTIC BASE</td>
<td>96</td>
</tr>
<tr>
<td>COMPARISON EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>BOOLEAN EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>25</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>25</td>
</tr>
<tr>
<td>EVALUATION OF CONDITION</td>
<td>25</td>
</tr>
<tr>
<td>SET MEMBERSHIP</td>
<td>25</td>
</tr>
<tr>
<td>CONDITIONAL OWNERSHIP</td>
<td>18</td>
</tr>
<tr>
<td>SUBOWNERSHIP</td>
<td>18</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>24  25 26 38</td>
</tr>
<tr>
<td></td>
<td>44 114 145 147</td>
</tr>
<tr>
<td></td>
<td>165 166 170</td>
</tr>
<tr>
<td>ACCESS CONDITION</td>
<td>26 147 170</td>
</tr>
<tr>
<td>ACTIVATION SEQUENCE</td>
<td>44</td>
</tr>
<tr>
<td>AUXILIARY PROGRAM INVOCATION</td>
<td>44</td>
</tr>
<tr>
<td>BOOLEAN EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>COMPARISON EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>25</td>
</tr>
<tr>
<td>DEFINITION OF</td>
<td>114</td>
</tr>
<tr>
<td>EVALUATION OF CONDITION</td>
<td>25</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>24 114 145 165</td>
</tr>
<tr>
<td></td>
<td>166 170</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>38</td>
</tr>
<tr>
<td>INVOCATION CONDITION</td>
<td>44</td>
</tr>
<tr>
<td>MULTICS</td>
<td>44</td>
</tr>
<tr>
<td>NOTATION</td>
<td>145</td>
</tr>
<tr>
<td>PREDICATE</td>
<td>24</td>
</tr>
<tr>
<td>RESOURCE UNIT DEFINING</td>
<td>26  166</td>
</tr>
<tr>
<td>SET MEMBERSHIP</td>
<td>25</td>
</tr>
<tr>
<td>SETS</td>
<td>145  147</td>
</tr>
<tr>
<td>STATES</td>
<td>24</td>
</tr>
<tr>
<td>TESTING</td>
<td>114</td>
</tr>
<tr>
<td>TRIPLE</td>
<td>38  44</td>
</tr>
<tr>
<td>USER GROUP DEFINING</td>
<td>26  165</td>
</tr>
<tr>
<td>CONFERENCE</td>
<td>118</td>
</tr>
<tr>
<td>ACCESS CONDITION</td>
<td>118</td>
</tr>
<tr>
<td>DIVEST</td>
<td>118</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>118</td>
</tr>
<tr>
<td>FRANCHISE</td>
<td>118</td>
</tr>
<tr>
<td>OPERATIONAL STATEMENTS</td>
<td>118</td>
</tr>
<tr>
<td>CONFIDENTIALITY</td>
<td>2</td>
</tr>
<tr>
<td>ACCESS CONTROL</td>
<td>2</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>2</td>
</tr>
<tr>
<td>Category</td>
<td>Pages</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>CONFIDENTIALITY</td>
<td>2</td>
</tr>
<tr>
<td>PRIVACY</td>
<td>2</td>
</tr>
<tr>
<td>PROTECTION</td>
<td>2</td>
</tr>
<tr>
<td>SECURITY</td>
<td>2</td>
</tr>
<tr>
<td>CONFINEMENT PRINCIPLE</td>
<td>39</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>39</td>
</tr>
<tr>
<td>SECURITY SPACE</td>
<td>39</td>
</tr>
<tr>
<td>COST</td>
<td>138</td>
</tr>
<tr>
<td>PERFORMANCE</td>
<td>138</td>
</tr>
<tr>
<td>PROGRAM PROVING</td>
<td>138</td>
</tr>
<tr>
<td>COUNTERS, SEE: SYSTEM COUNTERS</td>
<td></td>
</tr>
<tr>
<td>COVERING OF REQUEST</td>
<td></td>
</tr>
<tr>
<td>ALGORITHM</td>
<td>68</td>
</tr>
<tr>
<td>AND OPERATION (ENFORCEMENT)</td>
<td>72</td>
</tr>
<tr>
<td>AUXILIARY PROGRAM INVOCATION</td>
<td>74</td>
</tr>
<tr>
<td>BASIC SPECIFICATION</td>
<td>72</td>
</tr>
<tr>
<td>DOMAIN OF AUTHORIZATION</td>
<td>72</td>
</tr>
<tr>
<td>DOMAIN OF AUXILIARY INVOCATION</td>
<td>72</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>74</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>78</td>
</tr>
<tr>
<td>EXTENDED RESOURCES</td>
<td>78</td>
</tr>
<tr>
<td>FRANCHISE</td>
<td>78</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>74</td>
</tr>
<tr>
<td>INTERPRETATION</td>
<td>74</td>
</tr>
<tr>
<td>OR OPERATION (ENFORCEMENT)</td>
<td>74</td>
</tr>
<tr>
<td>PARTITIONING</td>
<td>72</td>
</tr>
<tr>
<td>RESOURCE REFERENCE</td>
<td>72</td>
</tr>
<tr>
<td>CRYPTOGRAPHY</td>
<td>6</td>
</tr>
<tr>
<td>DATA CONTENT</td>
<td></td>
</tr>
<tr>
<td>GROUPING</td>
<td>22</td>
</tr>
<tr>
<td>RESOURCE VALUES</td>
<td>22</td>
</tr>
<tr>
<td>SUBSETS</td>
<td>22</td>
</tr>
<tr>
<td>DATA OPERATIONS</td>
<td>20</td>
</tr>
<tr>
<td>DATA OPERATIONS</td>
<td></td>
</tr>
<tr>
<td>NOTATION</td>
<td>20</td>
</tr>
<tr>
<td>RESOURCE UNITS</td>
<td>20</td>
</tr>
<tr>
<td>RESOURCES</td>
<td>20</td>
</tr>
<tr>
<td>SETS</td>
<td>20</td>
</tr>
<tr>
<td>DEFINITION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>62</td>
</tr>
<tr>
<td>ACCESS CONTROL</td>
<td>2</td>
</tr>
<tr>
<td>ACCESS DECISION</td>
<td>35</td>
</tr>
<tr>
<td>ACCESS REQUEST</td>
<td>30</td>
</tr>
<tr>
<td>AUTHORIZATION SPECIFICATION</td>
<td>30</td>
</tr>
<tr>
<td>AUTHORIZED SUBSPACE</td>
<td>62</td>
</tr>
<tr>
<td>BOOLEAN EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>COMPARISON EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>14</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>ACCESS DECISION</td>
<td>14</td>
</tr>
<tr>
<td>ACCESS REQUEST</td>
<td>70</td>
</tr>
<tr>
<td>ALGORITHM</td>
<td>187</td>
</tr>
<tr>
<td>AND OPERATION (ENFORCEMENT)</td>
<td>68</td>
</tr>
<tr>
<td>AUTHORIZATION PROCESS</td>
<td>14</td>
</tr>
<tr>
<td>BASIC MEMORYLESS CASE</td>
<td>164</td>
</tr>
<tr>
<td>BASIC SPECIFICATION</td>
<td>72</td>
</tr>
<tr>
<td>COVERING OF REQUEST</td>
<td>68</td>
</tr>
<tr>
<td>DILEMMA IN ENFORCEMENT</td>
<td>65</td>
</tr>
<tr>
<td>DOMAIN OF AUTHORIZATION</td>
<td>70</td>
</tr>
<tr>
<td>EFFECTIVE ACCESS CONDITION</td>
<td>73</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>76</td>
</tr>
<tr>
<td>FORMALIZED</td>
<td>164</td>
</tr>
<tr>
<td>FRANCHISE</td>
<td>70</td>
</tr>
<tr>
<td>FULL SPECIFICATION</td>
<td>73</td>
</tr>
<tr>
<td>INTERNAL REPRESENTATION</td>
<td>14</td>
</tr>
<tr>
<td>INTERPRETATION</td>
<td>68</td>
</tr>
<tr>
<td>MODEL</td>
<td>65</td>
</tr>
<tr>
<td>MULTICS</td>
<td>70</td>
</tr>
<tr>
<td>OR OPERATION (ENFORCEMENT)</td>
<td>68</td>
</tr>
<tr>
<td>OVERVIEW</td>
<td>14</td>
</tr>
<tr>
<td>OWNERSHIP</td>
<td>14</td>
</tr>
<tr>
<td>PARTIAL VS. FULL REJECTION</td>
<td>65</td>
</tr>
<tr>
<td>POLICY</td>
<td>28</td>
</tr>
<tr>
<td>PROCESSES</td>
<td>14</td>
</tr>
<tr>
<td>QUERY MODIFICATION</td>
<td>65</td>
</tr>
<tr>
<td>RESOURCE REFERENCE</td>
<td>72</td>
</tr>
<tr>
<td>SECURITY POLICY</td>
<td>28</td>
</tr>
<tr>
<td>VALIDITY</td>
<td>14</td>
</tr>
<tr>
<td>VARIABLE SEMANTICS</td>
<td>167</td>
</tr>
<tr>
<td>EVALUATION OF CONDITION</td>
<td>25</td>
</tr>
<tr>
<td>BOOLEAN EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>COMPARISON EXPRESSION</td>
<td>25</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>25</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>25</td>
</tr>
<tr>
<td>SET MEMBERSHIP</td>
<td>25</td>
</tr>
<tr>
<td>EVENT</td>
<td>35</td>
</tr>
<tr>
<td>ACCESS DECISION</td>
<td>35</td>
</tr>
<tr>
<td>ACCESS DECISION KEEPING</td>
<td>36</td>
</tr>
<tr>
<td>ACCESS EVENT KEEPING</td>
<td>36</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>35</td>
</tr>
<tr>
<td>FAILURE EVENT</td>
<td>36</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>35</td>
</tr>
<tr>
<td>INDEPENDENT EVENT</td>
<td>36</td>
</tr>
<tr>
<td>P-EVENT</td>
<td>35</td>
</tr>
<tr>
<td>SUCCESS EVENT</td>
<td>36</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>11</td>
</tr>
<tr>
<td>------------------</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>182</td>
</tr>
<tr>
<td>ACCESS CONDITION</td>
<td>118</td>
</tr>
<tr>
<td>ACCESS DEPENDENCY</td>
<td>11</td>
</tr>
<tr>
<td>ADVANCED PROTECTION SERVICES</td>
<td>121</td>
</tr>
<tr>
<td>AND OPERATION (ENFORCEMENT)</td>
<td>78</td>
</tr>
<tr>
<td>APPLICATIONS</td>
<td>48</td>
</tr>
<tr>
<td>AUTHORIZATION PROCESS</td>
<td>76</td>
</tr>
<tr>
<td>AUTHORIZATION SPECIFICATION</td>
<td>32</td>
</tr>
<tr>
<td>AUTHORIZED SUBSPACE</td>
<td>62</td>
</tr>
<tr>
<td>AUXILIARY PROGRAM INVOCATION</td>
<td>40</td>
</tr>
<tr>
<td>BASIC SPECIFICATION</td>
<td>173</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>166</td>
</tr>
<tr>
<td>CONFER</td>
<td>118</td>
</tr>
<tr>
<td>COVERING OF REQUEST</td>
<td>78</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>62</td>
</tr>
<tr>
<td>DEFINITION OF</td>
<td>114</td>
</tr>
<tr>
<td>DEFINITIONAL STATEMENTS</td>
<td>113</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>120</td>
</tr>
<tr>
<td>DIVEST</td>
<td>118</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>76</td>
</tr>
<tr>
<td>EXTENDED RESOURCES</td>
<td>53</td>
</tr>
<tr>
<td>FRANCHISE</td>
<td>118</td>
</tr>
<tr>
<td>GENERALIZED</td>
<td>116</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>40</td>
</tr>
<tr>
<td>INVOCATION CONDITION</td>
<td>125</td>
</tr>
<tr>
<td>LOCK/UNLOCK</td>
<td>131</td>
</tr>
<tr>
<td>NOTATION</td>
<td>145</td>
</tr>
<tr>
<td>ON-CONDITION</td>
<td>48</td>
</tr>
<tr>
<td>OPERATIONAL STATEMENTS</td>
<td>118</td>
</tr>
<tr>
<td>OR OPERATION (ENFORCEMENT)</td>
<td>78</td>
</tr>
<tr>
<td>PARTIAL VS. FULL REJECTION</td>
<td>77</td>
</tr>
<tr>
<td>PARTITIONING</td>
<td>78</td>
</tr>
<tr>
<td>PREDICATE</td>
<td>24</td>
</tr>
<tr>
<td>PROGRESSIVE AUTHORIZATION</td>
<td>48</td>
</tr>
<tr>
<td>PROTECTION LANGUAGE CONSTRUCTS</td>
<td>112</td>
</tr>
<tr>
<td>PROTECTION REQUIREMENTS</td>
<td>11</td>
</tr>
<tr>
<td>RESOURCE UNIT DEFINING</td>
<td>166</td>
</tr>
<tr>
<td>RESOURCE UNIT GROUPS</td>
<td>116</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>OR Operation (Enforcement)</td>
<td>66</td>
</tr>
<tr>
<td>AND Operation (Enforcement)</td>
<td>68</td>
</tr>
<tr>
<td>Covering of Request</td>
<td>68</td>
</tr>
<tr>
<td>Enforcement Process</td>
<td>68</td>
</tr>
<tr>
<td>Examples</td>
<td>78</td>
</tr>
<tr>
<td>Interpretation</td>
<td>68</td>
</tr>
<tr>
<td>Lock/Unlock</td>
<td>69</td>
</tr>
<tr>
<td>Partial vs. Full Rejection</td>
<td>66</td>
</tr>
<tr>
<td>Partitioning</td>
<td>78</td>
</tr>
<tr>
<td>Query Modification</td>
<td>66</td>
</tr>
<tr>
<td>Resolution of Enforcement</td>
<td>66</td>
</tr>
<tr>
<td>Semantic Specifications</td>
<td>131</td>
</tr>
<tr>
<td>Overview</td>
<td>14</td>
</tr>
<tr>
<td>Access Control Information</td>
<td>14</td>
</tr>
<tr>
<td>Access Control Processes</td>
<td>14</td>
</tr>
<tr>
<td>Access Decision</td>
<td>14</td>
</tr>
<tr>
<td>Access Request</td>
<td>14</td>
</tr>
<tr>
<td>Authorization Process</td>
<td>14</td>
</tr>
<tr>
<td>Enforcement Process</td>
<td>14</td>
</tr>
<tr>
<td>Internal Representation</td>
<td>14</td>
</tr>
<tr>
<td>Ownership</td>
<td>14</td>
</tr>
<tr>
<td>Processes</td>
<td>14</td>
</tr>
<tr>
<td>Validity</td>
<td>14</td>
</tr>
<tr>
<td>Ownership, See Also: Validity</td>
<td></td>
</tr>
<tr>
<td>Access Control Information</td>
<td>14</td>
</tr>
<tr>
<td>Access Control Processes</td>
<td>14</td>
</tr>
<tr>
<td>Access Decision</td>
<td>14</td>
</tr>
<tr>
<td>Access Request</td>
<td>14</td>
</tr>
<tr>
<td>Authorization Process</td>
<td>14</td>
</tr>
<tr>
<td>Enforcement Process</td>
<td>14</td>
</tr>
<tr>
<td>Group Ownership</td>
<td>17</td>
</tr>
<tr>
<td>Internal Representation</td>
<td>14</td>
</tr>
<tr>
<td>Overview</td>
<td>14</td>
</tr>
<tr>
<td>Processes</td>
<td>14</td>
</tr>
<tr>
<td>Rules</td>
<td>58</td>
</tr>
<tr>
<td>SysAdmin</td>
<td>17</td>
</tr>
<tr>
<td>Validity</td>
<td>14</td>
</tr>
<tr>
<td>P-Event</td>
<td>35</td>
</tr>
<tr>
<td>Access Decision</td>
<td>35</td>
</tr>
<tr>
<td>Definition</td>
<td>35</td>
</tr>
<tr>
<td>Event</td>
<td>35</td>
</tr>
<tr>
<td>History Keeping</td>
<td>35</td>
</tr>
<tr>
<td>Pair</td>
<td>54</td>
</tr>
<tr>
<td>Definition</td>
<td>54</td>
</tr>
<tr>
<td>Extended Resources</td>
<td>54</td>
</tr>
<tr>
<td>Topic</td>
<td>Pages</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>PAIR</strong></td>
<td>54</td>
</tr>
<tr>
<td>INTERLOCKING</td>
<td>54</td>
</tr>
<tr>
<td>SHARED RESOURCES</td>
<td></td>
</tr>
<tr>
<td><strong>PARTIAL VS. FULL REJECTION</strong></td>
<td>65</td>
</tr>
<tr>
<td>DILEMMA IN ENFORCEMENT</td>
<td>65</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>65</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>77</td>
</tr>
<tr>
<td>MODEL</td>
<td>65</td>
</tr>
<tr>
<td>OR OPERATION (ENFORCEMENT)</td>
<td>66</td>
</tr>
<tr>
<td>QUERY MODIFICATION</td>
<td>65</td>
</tr>
<tr>
<td>RESOLUTION OF ENFORCEMENT</td>
<td>66</td>
</tr>
<tr>
<td><strong>PARTITIONING</strong></td>
<td>74</td>
</tr>
<tr>
<td>AND OPERATION (ENFORCEMENT)</td>
<td>78</td>
</tr>
<tr>
<td>AUXILIARY PROGRAM INVOCATION</td>
<td>74</td>
</tr>
<tr>
<td>COVERING OF REQUEST</td>
<td>74</td>
</tr>
<tr>
<td>DOMAIN OF AUTHORIZATION</td>
<td>74</td>
</tr>
<tr>
<td>DOMAIN OF AUXILIARY INVOCATION</td>
<td>74</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>78</td>
</tr>
<tr>
<td>EXTENDED RESOURCES</td>
<td>74</td>
</tr>
<tr>
<td>FRANCHISE</td>
<td>74</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>74</td>
</tr>
<tr>
<td>OR OPERATION (ENFORCEMENT)</td>
<td>78</td>
</tr>
<tr>
<td><strong>PERFORMANCE</strong></td>
<td>138</td>
</tr>
<tr>
<td>COST</td>
<td>138</td>
</tr>
<tr>
<td>PROGRAM PROVING</td>
<td></td>
</tr>
<tr>
<td><strong>PHYSICAL PROTECTION</strong></td>
<td>4</td>
</tr>
<tr>
<td>INFORMATION PROTECTION</td>
<td>4</td>
</tr>
<tr>
<td>LOGICAL PROTECTION</td>
<td>4</td>
</tr>
<tr>
<td><strong>POLICIES VS. MECHANISMS</strong></td>
<td>16</td>
</tr>
<tr>
<td>HYDRA</td>
<td>16</td>
</tr>
<tr>
<td><strong>POLICY</strong></td>
<td>28</td>
</tr>
<tr>
<td>ACCESS REQUEST</td>
<td>28</td>
</tr>
<tr>
<td>AUTHORIZATION PROCESS</td>
<td>28</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>28</td>
</tr>
<tr>
<td>SECURITY POLICY</td>
<td>28</td>
</tr>
<tr>
<td>VALIDITY</td>
<td>28</td>
</tr>
<tr>
<td><strong>PREDICATE</strong></td>
<td>24</td>
</tr>
<tr>
<td>ACCESS DECISION PREDICATE</td>
<td>59</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>24</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>24</td>
</tr>
<tr>
<td>PROJECTION FUNCTIONS</td>
<td>59</td>
</tr>
<tr>
<td>STATES</td>
<td>24</td>
</tr>
<tr>
<td>VALID SUBSPACE</td>
<td>59</td>
</tr>
<tr>
<td><strong>PRIVACY</strong></td>
<td>2</td>
</tr>
<tr>
<td>ACCESS CONTROL</td>
<td>2</td>
</tr>
<tr>
<td>CONFIDENTIALITY</td>
<td>2</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>2</td>
</tr>
<tr>
<td>PROTECTION</td>
<td>2</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>RESOURCE SECURITY SYSTEM</td>
<td>21</td>
</tr>
<tr>
<td>MULTICS</td>
<td>21</td>
</tr>
<tr>
<td>NOTATION</td>
<td>21</td>
</tr>
<tr>
<td>SETS</td>
<td>21</td>
</tr>
<tr>
<td>STRUCTURING</td>
<td>21</td>
</tr>
<tr>
<td>RESOURCE UNIT DEFINING</td>
<td>26 166</td>
</tr>
<tr>
<td>ACCESS CONDITION</td>
<td>26</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>26 166</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>166</td>
</tr>
<tr>
<td>USER GROUP DEFINING</td>
<td>26</td>
</tr>
<tr>
<td>RESOURCE UNIT GROUPS</td>
<td>116</td>
</tr>
<tr>
<td>DEFINITIONAL STATEMENTS</td>
<td>116</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>116</td>
</tr>
<tr>
<td>GENERALIZED</td>
<td>116</td>
</tr>
<tr>
<td>SINGLE USER DEFINITIONS</td>
<td>116</td>
</tr>
<tr>
<td>USER DEFINITIONS</td>
<td>116</td>
</tr>
<tr>
<td>USER GROUPS</td>
<td>116</td>
</tr>
<tr>
<td>RESOURCE UNITS</td>
<td>20 60 113</td>
</tr>
<tr>
<td>DATA OPERATIONS</td>
<td>20</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>60</td>
</tr>
<tr>
<td>DEFINITIONAL STATEMENTS</td>
<td>113</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>113</td>
</tr>
<tr>
<td>GENERAL (GROUP)</td>
<td>60</td>
</tr>
<tr>
<td>GRANTING ACCESS RIGHTS</td>
<td>60</td>
</tr>
<tr>
<td>NOTATION</td>
<td>20</td>
</tr>
<tr>
<td>RESOURCES</td>
<td>20</td>
</tr>
<tr>
<td>SETS</td>
<td>20</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>113</td>
</tr>
<tr>
<td>USER GROUPS</td>
<td>60 113</td>
</tr>
<tr>
<td>RESOURCE VALUES</td>
<td>22</td>
</tr>
<tr>
<td>DATA CONTENT</td>
<td>22</td>
</tr>
<tr>
<td>GROUPING</td>
<td>22</td>
</tr>
<tr>
<td>SUBSETS</td>
<td>22</td>
</tr>
<tr>
<td>RESOURCES</td>
<td>20</td>
</tr>
<tr>
<td>DATA OPERATIONS</td>
<td>20</td>
</tr>
<tr>
<td>NOTATION</td>
<td>20</td>
</tr>
<tr>
<td>RESOURCE UNITS</td>
<td>20</td>
</tr>
<tr>
<td>SETS</td>
<td>20</td>
</tr>
<tr>
<td>RESTRICTIONS (RESOURCE VALUES)</td>
<td>23</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>23</td>
</tr>
<tr>
<td>STATES</td>
<td>23</td>
</tr>
<tr>
<td>RULES</td>
<td>58</td>
</tr>
<tr>
<td>OWNERSHIP</td>
<td>58</td>
</tr>
<tr>
<td>VALIDITY</td>
<td>5P</td>
</tr>
<tr>
<td>SAMPLE PROTECTION LANGUAGE</td>
<td>102 150</td>
</tr>
<tr>
<td>BACKUS-NAUR-FORM</td>
<td>150</td>
</tr>
<tr>
<td>PROTECTION LANGUAGE CONSTRUCTS</td>
<td>102</td>
</tr>
<tr>
<td>PROTECTION LANGUAGES</td>
<td>102</td>
</tr>
<tr>
<td>SYNTAX</td>
<td>150</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>SECURITY</td>
<td>2</td>
</tr>
<tr>
<td>ACCESS CONTROL</td>
<td>7</td>
</tr>
<tr>
<td>CONFIDENTIALITY</td>
<td>2</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>2</td>
</tr>
<tr>
<td>PRIVACY</td>
<td>2</td>
</tr>
<tr>
<td>PROTECTION</td>
<td>2</td>
</tr>
<tr>
<td>SECURITY POLICY</td>
<td>28</td>
</tr>
<tr>
<td>ACCESS REQUEST</td>
<td>28</td>
</tr>
<tr>
<td>AUTHORIZATION PROCESS</td>
<td>28</td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>28</td>
</tr>
<tr>
<td>POLICY</td>
<td>28</td>
</tr>
<tr>
<td>VALIDITY</td>
<td>28</td>
</tr>
<tr>
<td>SECURITY SPACE</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>80</td>
</tr>
<tr>
<td>BASIC SPECIFICATION</td>
<td>173</td>
</tr>
<tr>
<td>CONFINEMENT PRINCIPLE</td>
<td>39</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>29</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>173</td>
</tr>
<tr>
<td>HISTORY KEEPING</td>
<td>8</td>
</tr>
<tr>
<td>MULTICS</td>
<td>8</td>
</tr>
<tr>
<td>PROTECTION IN DBL'S</td>
<td>8</td>
</tr>
<tr>
<td>PROTECTION IN PROGRAMMING</td>
<td>8</td>
</tr>
<tr>
<td>5-TUPLE</td>
<td>29</td>
</tr>
<tr>
<td>SEMANTIC</td>
<td>93</td>
</tr>
<tr>
<td>PROTECTION LANGUAGES</td>
<td>93</td>
</tr>
<tr>
<td>TRANSLATION</td>
<td>93</td>
</tr>
<tr>
<td>SEMANTIC BASE</td>
<td>96</td>
</tr>
<tr>
<td>BASE LANGUAGE</td>
<td>96</td>
</tr>
<tr>
<td>COMMON BASE LANGUAGE</td>
<td>96</td>
</tr>
<tr>
<td>SEMANTIC CONSTRUCTS, SEE ALSO: PROTECTION LANGUAGE CONSTRUCTS</td>
<td></td>
</tr>
<tr>
<td>SEMANTIC CONSTRUCTS</td>
<td>133</td>
</tr>
<tr>
<td>DESIGN</td>
<td>133</td>
</tr>
<tr>
<td>LEVELS OF SOPHISTICATION</td>
<td>133</td>
</tr>
<tr>
<td>PROTECTION LANGUAGE CONSTRUCTS</td>
<td>133</td>
</tr>
<tr>
<td>PROTECTION LANGUAGES</td>
<td>133</td>
</tr>
<tr>
<td>SEMANTIC MODEL</td>
<td>13</td>
</tr>
<tr>
<td>SEMANTIC PARAMETERS, SEE: SEMANTIC SPECIFICATIONS</td>
<td></td>
</tr>
<tr>
<td>SEMANTIC SPECIFICATIONS</td>
<td>130</td>
</tr>
<tr>
<td>SYSDES</td>
<td>130</td>
</tr>
<tr>
<td>VARIABLE SEMANTICS</td>
<td>130</td>
</tr>
<tr>
<td>SEMANTIC SPECIFICATIONS</td>
<td>101</td>
</tr>
<tr>
<td>AND OPERATION (ENFORCEMENT)</td>
<td>131</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>112</td>
</tr>
<tr>
<td>LOCK/UNLOCK</td>
<td>131</td>
</tr>
<tr>
<td>OR OPERATION (ENFORCEMENT)</td>
<td>131</td>
</tr>
<tr>
<td>PROTECTION LANGUAGE CONSTRUCTS</td>
<td>112</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Semantic Specifications</td>
<td>112</td>
</tr>
<tr>
<td>Translation</td>
<td>101</td>
</tr>
<tr>
<td>Set Membership</td>
<td>25</td>
</tr>
<tr>
<td>Boolean Expression</td>
<td>25</td>
</tr>
<tr>
<td>Comparison Expression</td>
<td>25</td>
</tr>
<tr>
<td>Conditions</td>
<td>25</td>
</tr>
<tr>
<td>Definition</td>
<td>25</td>
</tr>
<tr>
<td>Evaluation of Condition</td>
<td>25</td>
</tr>
<tr>
<td>Set Notation, See: Notation</td>
<td>19, 20, 21, 145</td>
</tr>
<tr>
<td>Access Condition</td>
<td>147</td>
</tr>
<tr>
<td>ADEPT-50</td>
<td>21</td>
</tr>
<tr>
<td>Authorizers</td>
<td>19</td>
</tr>
<tr>
<td>Conditions</td>
<td>145, 147</td>
</tr>
<tr>
<td>Data Operations</td>
<td>20</td>
</tr>
<tr>
<td>Examples</td>
<td>145</td>
</tr>
<tr>
<td>Model</td>
<td>19</td>
</tr>
<tr>
<td>MULTICS</td>
<td>21</td>
</tr>
<tr>
<td>Notation</td>
<td>20, 21, 145</td>
</tr>
<tr>
<td>Resource Security System</td>
<td>21</td>
</tr>
<tr>
<td>Resource Units</td>
<td>20</td>
</tr>
<tr>
<td>Resources</td>
<td>20</td>
</tr>
<tr>
<td>Structuring</td>
<td>21</td>
</tr>
<tr>
<td>User Groups</td>
<td>19</td>
</tr>
<tr>
<td>Users</td>
<td>19</td>
</tr>
<tr>
<td>Shared Resources</td>
<td>54</td>
</tr>
<tr>
<td>Definition</td>
<td>54</td>
</tr>
<tr>
<td>Extended Resources</td>
<td>54</td>
</tr>
<tr>
<td>Interlocking</td>
<td>54</td>
</tr>
<tr>
<td>Pair</td>
<td>54</td>
</tr>
<tr>
<td>Simple</td>
<td>113</td>
</tr>
<tr>
<td>Definitional Statements</td>
<td>113</td>
</tr>
<tr>
<td>Examples</td>
<td>113</td>
</tr>
<tr>
<td>Resource Units</td>
<td>113</td>
</tr>
<tr>
<td>User Groups</td>
<td>113</td>
</tr>
<tr>
<td>Single User Definitions</td>
<td>116</td>
</tr>
<tr>
<td>Definitional Statements</td>
<td>116</td>
</tr>
<tr>
<td>Examples</td>
<td>116</td>
</tr>
<tr>
<td>Generalized</td>
<td>116</td>
</tr>
<tr>
<td>Resource Unit Groups</td>
<td>116</td>
</tr>
<tr>
<td>User Definitions</td>
<td>116</td>
</tr>
<tr>
<td>User Groups</td>
<td>116</td>
</tr>
<tr>
<td>States</td>
<td>23, 24</td>
</tr>
<tr>
<td>Conditions</td>
<td>24</td>
</tr>
<tr>
<td>Definition</td>
<td>23</td>
</tr>
<tr>
<td>Examples</td>
<td>24</td>
</tr>
<tr>
<td>USER GROUPS</td>
<td>19</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>AUTHORIZED</td>
<td>19</td>
</tr>
<tr>
<td>DEFINITION</td>
<td></td>
</tr>
<tr>
<td>DEFINITIONAL STATEMENTS</td>
<td></td>
</tr>
<tr>
<td>EXAMPLES</td>
<td></td>
</tr>
<tr>
<td>GENERAL (GROUP)</td>
<td></td>
</tr>
<tr>
<td>GENERALIZED</td>
<td></td>
</tr>
<tr>
<td>GRANTING ACCESS RIGHTS</td>
<td></td>
</tr>
<tr>
<td>MODEL</td>
<td></td>
</tr>
<tr>
<td>RESOURCE UNIT GROUPS</td>
<td></td>
</tr>
<tr>
<td>RESOURCE UNITS</td>
<td></td>
</tr>
<tr>
<td>SETS</td>
<td></td>
</tr>
<tr>
<td>SIMPLE</td>
<td></td>
</tr>
<tr>
<td>SINGLE USER DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>USER DEFINITIONS</td>
<td></td>
</tr>
<tr>
<td>USERS</td>
<td></td>
</tr>
<tr>
<td>USER IDENTIFICATION</td>
<td>5</td>
</tr>
<tr>
<td>APPLICATIONS</td>
<td></td>
</tr>
<tr>
<td>AUTHENTICATION</td>
<td></td>
</tr>
<tr>
<td>AUXILIARY PROGRAM INVOCATION</td>
<td></td>
</tr>
<tr>
<td>USERS</td>
<td>19</td>
</tr>
<tr>
<td>AUTHORIZED</td>
<td>19</td>
</tr>
<tr>
<td>MODEL</td>
<td>19</td>
</tr>
<tr>
<td>SETS</td>
<td>19</td>
</tr>
<tr>
<td>USER GROUPS</td>
<td>19</td>
</tr>
<tr>
<td>VALID SUBSPACE</td>
<td></td>
</tr>
<tr>
<td>ACCESS DECISION PREDICATE</td>
<td></td>
</tr>
<tr>
<td>EXAMPLES</td>
<td></td>
</tr>
<tr>
<td>PREDICATE</td>
<td></td>
</tr>
<tr>
<td>PROJECTION FUNCTIONS</td>
<td></td>
</tr>
<tr>
<td>VALIDATION, SEE: VALIDITY</td>
<td></td>
</tr>
<tr>
<td>VALIDATION PROCESS</td>
<td>162</td>
</tr>
<tr>
<td>AUTHORIZED SUBSPACE</td>
<td>162</td>
</tr>
<tr>
<td>FORMALIZED</td>
<td>162</td>
</tr>
<tr>
<td>GENERALIZED</td>
<td>162</td>
</tr>
<tr>
<td>VALIDITY</td>
<td>162</td>
</tr>
<tr>
<td>VALIDITY, SEE ALSO: OWNERSHIP</td>
<td></td>
</tr>
<tr>
<td>VALIDITY</td>
<td></td>
</tr>
<tr>
<td>ACCESS CONTROL INFORMATION</td>
<td>14</td>
</tr>
<tr>
<td>ACCESS CONTROL PROCESSES</td>
<td>14</td>
</tr>
<tr>
<td>ACCESS DECISION</td>
<td>14</td>
</tr>
<tr>
<td>ACCESS REQUEST</td>
<td>14</td>
</tr>
<tr>
<td>AUTHORIZATION PROCESS</td>
<td>14</td>
</tr>
<tr>
<td>AUTHORIZED SUBSPACE</td>
<td></td>
</tr>
<tr>
<td>ENFORCEMENT PROCESS</td>
<td>14</td>
</tr>
<tr>
<td>FORMALIZED</td>
<td>162</td>
</tr>
</tbody>
</table>
VALIDITY
  GENERALIZED  162
  INTERNAL REPRESENTATION  14
  OVERVIEW  14
  OWNERSHIP  14  58
  POLICY  28
  PROCESSES  14
  RULES  58
  SECURITY POLICY  28
  VALIDATION PROCESS  162
VARIABLE SEMANTICS  101  130  187
  ALGORITHM  187
  ENFORCEMENT PROCESS  187
  SEMANTIC SPECIFICATIONS  130
  SEMANTIC SPECIFICATIONS  101
  SYSDES  130
WITHDRAWING ACCESS RIGHTS  63
  EXAMPLES  63
5-TUPLE  29
  DEFINITION  29
  SECURITY SPACE  29