SRC

SOME RESULTS IN A THEORY OF PROBLEM SOLVING

(PART II)

R.B. Banerji

SRC-68-10

SOME RESULTS IN A THEORY OF PROBLEM SOLVING (PART II)

R.B. BANERJI

SRC-68-10

SYSTEMS RESEARCH CENTER

CASE WESTERN RESERVE UNIVERSITY

UNIVERSITY CIRCLE

CLEVELAND, OHIO 44106

NOVEMBER, 1967

This work was sponsored in part by the USA Air Force Office of Scientific Research under grants AF-OSR-125-67, AF-OSR-125-65, and by the National Science Foundation under grants GK 1386, GK185 and GP 658.

FOREWORD

This report is a continuation of SRC-126-A-67-59 entitled, "Some Results in A Theory of Problem Solving," which were the first three chapters of a book to be published by American Elsevier Publishing Company, Inc.

In the previous report it was mentioned that the chapters of the book following Chapter III would not be published in report form. Nevertheless, the present report is being published for two reasons. One is that the chapters reported here contain much material which did not appear in previous publications. These additions are both in the way of formalizations and interpretations of previous work.

Another reason for publishing this report is the fact that on seeing the first report, many people on our mailing list have asked me to "send Chapters IV and V as soon as they are written." This report enables us to do just that. As a result, this report is merely the first draft of these chapters. I believe that the final version, both of this report and the previous one, will be much more readable than these.

The work reported on here was carried out by myself and graduate students supported by the U. S. Air Force Office of Scientific Research under grants AF-OSR-125-67, AF-OSR-125-65, and by the National Science Foundation under grants GK 1386,

GK 185 and GP 658. Some of the graduate students whose work is reported here were supported by NDE' NSF and Case Fellowships.

A word of apology has to be added regarding the format of this report. It starts on "page 164" and "Chapter IV." The main reason is time pressure. The Table of Contents show similar pecurliarity of behavior. The page numbers shown for Chapters I - III are only approximate. The contents of Chapters I and II are exactly as in the previous report. In Chapter III, what is called Section 9 in the Table of Contents did not appear in the previous report. What is called Section 10 in that chapter appeared as Section 9 in the previous report. The section called Section 9 of Chapter II in the Table of Contents appears as an appendix to the present report. The page numbers shown in the Table of Contents for Chapters IV and V refer to the page numbers of this present report.

TABLE OF CONTENTS

Preface

Chapter I-Introduction. 1

Section 1-The Field of Endeavor 1

Section 2-Outline of the Basic Models 11

Section 3-A Set-Theoretic View of Pattern Recognition 17

Section 4-The Arrangement of the Book 31

Chapter II-Problems and Solution Methods 32

Section 1-Introduction. 32

Section 2-Some Properties of M-Situations 34

Section 3-W-Problems and M-Situations 38

Section 4-A Simple Example of a W-Problem: The Tower of Hanoi. 46

Section 5-The Logic Theorist - Another Example. . . 51

Section 6-Strategies and their Description. 57

Section 7-Evaluations: A Method for Defining Strategies. 64

Section 8-Strategies Based on T'. 71

Section 9-Strategies Based on Subgoals -- The General Problem Solver. 75

Section 10-Sundry Remarks Regarding Search for Winning Sequences 85

Chapter III-Games and Solution Methods. 94

Section 1-Introduction. 94

Section 2-Game Situations and Strategies. 96

Section 3-Winning Solutions in Board Games. 105

Section 4-The NIM Class of Games - An Example . . . 118

Section 5-The Tic-Tac-Toe-Like Games - Another Example 121

Section 6-Evaluating Strategies in Board Games. . . 128

Section 7-Strategies Based on Graph Decomposition . 138

Section 8-Some Examples of Strategy-Construction. . 159

Section 9-Approximation to Strategies in Tic-Tac-Toe-Like Games. 164

Section 10-Recognizing Forcing States Through Linear Evaluation 173

Chapter IV-Describing Patterns 181

Section 1-Introduction. 181

Section 2-Some Basic Terms and Discussions. 186

Section 3-Conceptions - A Description Language. . . 194

Section 4-A Recognition Algorithm using Conceptions 202

Section 5-Conjunctive and Simple Concepts 216

Section 6 A Generalized Description Language: Syntactic Axiomatisations 231

Section 7-Other Description Languages 259

Chapter V-Learning and Generalization 266

Section 1-Introduction. 266

Section 2-Learning Conjunctive Concepts 269

Section 3-Learning Simple Concepts. 277

Section 4-Problems of Learning and Feature Extraction. 281

Section 5-Generalization - "Concept Formation" and Languages 295

Section 6-Learning Games by Generalization - Importance of Description Languages . 308

References. 312

CHAPTER IV - DESCRIBING PATTERNS

1. Introduction

The importance of pattern recognition to solutions of problems and games was discussed briefly in Section 3 of Chapter I. In Chapters II and III, as sets like T_i , T', S^{o}_{fX} , W_i , K_i , and blocks of E_i or that of the Kernels of functions like Q were discussed, it was implicitly or explicitly stated that the use of these sets in the construction of solution methods are practicable if and only if they can be described efficiently.

This chapter will be devoted entirely to the discussion of descriptions, and the way the efficiency of description depends on the set being described and the language used for describing it. Precise definitions will be attempted for the ideas and terms involved. All the motivations for the formalisms introduced will not be repeated here since some of them have already been discussed in Chapter 1. Specifically, the reader is reminded that a "pattern" is defined to be a subset of a pre-defined Universe of Discourse. However, it may be worthwhile at this point to include an informal discussion of terms whose precise definition will not be stressed in this chapter.

By the term "language" will be meant a combination of two things - a set of syntactic expressions (generated, for instance, by some generative grammar[26]) and the interpretations

of these expressions as denoting sets. For this latter to be possible, it is necessary that some of the syntactic entities be predicates defining certain "intuitively recognizable" sets of objects in the Universe of Discourse. In addition, the syntax has to have various ways of combining predicates to yield compound statements. These compound statements will denote sets which are uniquely related to the intuitively recognizable sets in a way dictated by the structure of the compound statements.

The first few sections of this chapter will be devoted to the development of some formal definitions and then to some specific languages which are meaningful in a wide variety of Universes of Discourse. The major emphasis will be on the "efficiencies" of these languages.

By the efficiency of a language for the description of a given set will be meant the "size" (in some sense) of the "shortest" expression which denotes that set. This size depends on the set as well as the predicates of the language and the repertoire of combination modes available. It will be taken for granted that new predicates can be defined for enriching the language. That is, some compound statements may be replaced by shorter expressions by defining new syntactic entities in the language.

The definition of the word, "size" was kept purposefully vague in the last paragraph; because a precise definition (being heavily dependent on technology) is hard to give in

absolute terms. In a very rough way one may say that the size of an expression is measured by the number of symbols in it. In the author's own thinking (for reasons which will be clarified in the proper context) certain symbols in the expression (like "or" of Propositional Calculus) seem to have greater size than others (like "and").

The discussion above has been limited to the nature of statements which describe "given" sets. If by "given set" one means a set for which a description is available then the problem of obtaining a short description turns out to be a trivial one - or at least a problem of transliteration. However, if by a "given set" one means a set whose elements are all available as a list, then one can consider the problem of generating a succinet statement in a language which will be satisfied by every element of the list and by none else. This, roughly, is the problem of "concept learning."

Since such lists are impossibly large in practical cases one may, instead, consider a case where only some members of a set are exhibited on a short list. This, however, can give no meaningful clue to a learning program. One can infer, without any contradiction from the presentation, that every object belongs to the set. It is essential that at least some members of the complement of the set are also exhibited in another short list. One can then consider the problem of generating a succinct statement which will denote some set which contains every element of the first list and none of the elements

of the second list. Typically, such a statement will be satisfied by certain objects which do not appear on either list. Thus, the expression will have "generalized" on the examples given. The mode of this generalization will be dependent on the method used for generating the describing expressions and to a certain extent on the language, since the language determines the succinctness of the statements. However, the "correctness" of the resulting generalization - whether the description actually denotes the set one "had in mind" in constructing the lists - is not at all determinable from the method of description generation alone.

The next chapter will describe certain algorithms for generalizations of the restricted ("not necessarily correct") variety. It will also discuss the possible situations under which generalizations may turn out to be correct.

Rough definitions of a few more terms may be useful for the reduction of confusion. In the literature, the term "pattern recognition" is used in two implied senses. In one sense it stands for what has been called generalization above. In this book "Pattern Learning" and "Concept Learning" or simply "Learning" will often be meant to signify the same phenomenon. In the other sense the term "Pattern Recognition" means the recognition of an object as belonging to a pattern of known description. In this book the terms "Pattern Recognition," "Recognition" and "Object Recognition" will all be used in this sense.

Another term, "Concept Formation" is often used for what has been called "Concept Learning" above. In the next chapter a much more complex phenomenon will be called "concept formation."

The word "concept" will often be used for the word "pattern" in this book. The reason for this is historical - the initial models and languages developed at Case,[27, 28] were developed for understanding the psychological process of concept formation.[19] The relevance of the ideas to the field of pattern recognition was realized only later. This realization immediately led to the need for further developments of the formalisms. It is the author's belief that these further developments have made the theory even more relevant to psychology then they were before. The theory, in its present form, therefore, will use both the terms.

2. Some Basic Terms and Discussions

The present section will formalize some of the basic ideas referred to in Section 3 of Chapter I to initiate the discussion.

A pattern recognition environment (called environment for short) is an ordered pair $\langle U, \mathcal{P} \rangle$ where U is an abstract set and $\mathcal{P}$ is a family of non-trivial partitions on U. In much of what follows, the family $\mathcal{P}$ and each of its elements will be considered to be finite, although some of the definitions and results are meaningful even if some elements of $\mathcal{P}$ are infinite classes.

U will be referred to as the Universe of Discourse (or Universe for short). Each element of $\mathcal{P}$ will be called a Property. If P is a property, then each element of $p \in P$ (where p, clearly is a subset of U) will be called a value of P.

The reader will notice that the word "value" is used for certain pre-defined subsets of the Universe of Discourse while in most of mathematical literature the word "property" is used in this sense. However, it may be worthwhile to recall that the property "redness" and the property "color" are two distinct things. "Redness" is a property in the usual mathematical sense. But "color" is also referred to as property in common parlance: a fact one would like to recognize in the theory. Psychologists use the words "characteristic" or "dimension" instead of the word "property."[29]

A concept (or a pattern) is defined recursively as

follows:

(Ci) A value of a property is a concept

(Cii) If A and B are concepts then $A \cup B$ is a concept

(Ciii) If A is a concept, then the complement $\bar{A}$ of A is a concept

(Civ) Nothing is a concept unless its being so follows from (i), (ii) and (iii) above.

In most of the previous work at Case, (iii) above was replaced by, "If A and B are concepts, then $A \cap B$ is a concept." However, in such cases, the class of concepts do not form a Boolean Algebra except for the cases where each element of $\mathcal{P}$ was a finite partition. This is because complements of concepts may not always be concepts if partitions have an infinite number of blocks. The difficulty is removed by the definitions above. It could also have been removed by allowing infinite unions and intersections: however, since the description languages presupposed in this book begins to have practical difficulties any time infinite operations are used (difficulties shared by any pattern recognition scheme using infinitary processes) it was considered more meaningful to have the definitions as above. One can motivate the above definition of a concept by saying, "A concept is a set of things whose elements are recognizable as belonging to it by virtue of their properties."

For convenience of later discussion, we shall define an environment to be <u>finite</u> if $\mathcal{P}$ is a finite family and each element of $\mathcal{P}$ is a finite partition.

Given a subfamily $\mathcal{P}'$ of $\mathcal{P}$ one defines a subclass $C_{\mathcal{P}'}$ of the class of all concepts as follows:

(i) Any value of any element of $\mathcal{P}'$ is a member of $C_{\mathcal{P}'}$.

(ii) If A and B are members of $C_{\mathcal{P}'}$, then $A \cup B$ and $\bar{A}$ are members of $C_{\mathcal{P}'}$.

(iii) Nothing is a member of $C_{\mathcal{P}'}$ unless its being so follows from (i) and (ii) above.

By this definition, the class $C_{\mathcal{P}}$ is the class of all concepts.

A subfamily $\mathcal{P}'$ of $\mathcal{P}$ is called a <u>fine structure</u> family if and only if $C_{\mathcal{P}'} = C_{\mathcal{P}}$.

A finite fine structure family $\mathcal{P}' = \{P_1, P_2, \ldots P_n\}$ is said to be <u>full</u> if

$$p_{1i_1} \cap p_{2i_2} \cap \ldots \cap p_{ni_n} \neq \emptyset \qquad (4.1)$$

for each $p_{ri_r} \in P_r$.

A fine structure family of properties in any environment set the limit to the distinguishability of members of the Universe, as will be shown presently. If the fine structure family is much smaller than the set $\mathcal{P}$, then the properties outside the fine structure family merely affect the efficiency of description and not the ultimate capability of description. Nevertheless, since efficiency of description is crucial, the distinction between $\mathcal{P}$ and $\mathcal{P}'$ is essential to the considerations of this chapter. To keep this basic role of the fine structure family clear one defines as follows.

A <u>real environment</u> is a triple $\langle U, P, P' \rangle$ where $\langle U, P \rangle$ is an environment and P' is a fine structure subfamily of P. (P' is not necessarily a proper subfamily of P, although in all interesting cases it would be.) P' will be called the <u>input properties</u> of the environment.

In the next few sections only finite real environments will be considered. In the work described in the next section, real environments with a full fine structure family of input properties will be assumed.

All description languages discussed in this chapter will have as its motivation the Boolean Algebraic structure of the class of concepts as defined above. Although the languages will differ in the mode of describing concepts, one aspect of them will remain the same. This pertains to the fact that any concept of the form shown in expression 4.1 above is in one sense, very basic: two members of U both of which belong to the set

$$p_{1i_1} \quad p_{2i_2} \quad \cdots \quad p_{ni_n}$$

are indistinguishable: if one of them belongs to any concept C, then the other must also belong to it. This can be readily shown by induction over the least number of set theoretical operators needed to exhibit the concept to be one according to (Ci), (Cii) and (Ciii) above. In fact, even a stronger version of this statement can be made.

<u>Theorem 4.1:</u> Let P' be indexed by the (not necessarily finite) set I so that $P' = P_{i_1 \quad i_I}$. For each $i \quad I$, let

$p_i \in P_i$. Let a and b be two elements of U such that for each $i \in I$, both a and b is a member of p_i. Let C be any concept. Then $a \in C$ if and only if $b \in C$.

<u>Proof:</u> Let C be a concept according to (Ci) in the definition. Then there is a $P_i \in \mathcal{P}'$ and a $p_{i'} \in P_i$ such that $C = p_{i'}$. Since $a \in p_i$ and $a \in C = p_{i'}$, we have $p_i \cap p_{i'} \neq \emptyset$. But since P_i is a partition, $p_i \cap p_{i'} \neq \emptyset$ implies $i = i'$. Since by hypothesis $b \in p_i$, one has $b \in C$ also. The converse follows. Let now the theorem be true for any concept which can be constructed with n or less set theoretic connectives. Let C be constructed by n+1 set theoretic connectives. If $C = A \cup B$, then either $a \in A$ or $a \in B$. Let $a \in A$. Since A is constructible with less than n connectives, $b \in A$ by induction hypothesis, so that $b \in A \cup B$. Similarly if $a \in B$. The converse follows also. If $C = \overline{A}$, then if b is not in C, it is in A. But A has less than n connectives. Hence, since $b \in A$ and by the symmetry of the theorem $a \in A$ which is impossible since $a \in \overline{A}$. Hence, $b \in C$.

Any object in U then, can be completely specified (so far as its membership in all concepts are concerned) by indicating its membership in one element of each of the properties in $\mathcal{P}'$. On the basis of this fact one can make the following definitions.

Given a finite real environment $\langle U, \mathcal{P}' \rangle$, a <u>generalized object</u> is a string of characters of the form $(p_{i_1}, p_{i_1}\ P_{i_2}, p_{i_2}, \ldots P_{i_n}, p_{i_n})$ where n is a finite integer

such that for each k, $P_{i_k} \in \mathcal{P}'$, $p_{i_k} \in P_{i_k}$ and

$p_{i_1} \cap p_{i_2} \cap \ldots \cap p_{i_k} \cap \ldots \cap p_{i_n} \neq \emptyset$.

A generalized object is an <u>object</u> if for all $P \in \mathcal{P}$ and $p \in P$, $p_{i_1} \cap \ldots \cap p_{i_n}$ is either contained in or disjoint from p.

For any finite environment (or even an environment where $\mathcal{P}'$ is finite) n in an object may be considered to be equal to the cardinality of $\mathcal{P}'$ even where the environment is not full. In a full environment, of course, it is necessary to have n equal to the cardinality of $\mathcal{P}'$ in any object. It may be noticed that an object defines a concept, to wit the concept $p_{i_1} \cap p_{i_2} \cap \ldots \cap p_{i_n}$. The use of the symbols P_{i_k} may, therefore, seem unnecessary in the definition of objects. However, retaining the properties, together with the values, has some important uses which will become clear towards the end of this chapter. Meanwhile, we shall refer to the concept defined by an object also as the object. There will be very little confusion arising from this double use - when they do arise they can be resolved from the context. On the other hand, the double use of the term will often considerably reduce the complexity of discussion and will introduce stronger motivation.

The basic predicates out of which the descriptions of concepts will be built will consist of statements of the form $P(x) = p$ where P is some property, p a value of P and x is a

variable. The predicate will be true for all members of the value p of the property P. Any member of the object $(P_{i_1}, p_{i_1}; \ldots; P_{i_n}, p_{i_n})$, then satisfies the statement S(x), where S(x) denotes the statement

$$(P_{i_1}(x) = p_{i_1}) \wedge (P_{i_2}(x) = p_{i_2}) \wedge \ldots \wedge (P_{i_n}(x) = p_{i_n}).$$

This statement may be considered to "describe" the object $(P_{i_1}, p_{i_1}; \ldots; P_{i_n}, p_{i_n})$ in the sense that the sentence S(a) will be true for all elements a of the object.

Obviously, concepts other than objects can be similarly "described" by statements involving the basic predicates $P(x) = p$ where $P \in \mathcal{P}$, $p \in P$ and usual logical connectives. Given any object and the description of any concept in such a language, one can readily determine whether the object is contained in the concept or not. Algorithms and formats used for such recognition processes will be described presently. Meanwhile, certain important aspects of this elementary language will bear discussion.

The central question regarding descriptions are the following:

(i) Given a concept, how should its description be stored so as to use as small an amount of memory as possible?

(ii) How should the description of a concept be stored and processed so that, given an object and a concept one can determine as efficiently as possible

whether the object is contained in the concept?

(iii) Given two sets of objects, how should one construct a short description of a concept which contains all elements of the first set and no elements of the second set?

In this and the next chapter some of the earlier alternative attempts at answering these questions will be described. They are included here because they compare favorably with some published work by other workers,[30, 31] and, in the author's opinion, sheds some light on the nature of the problem.

3. Conceptions - A Description Language

The discussions in this section are based on the work of the author,[28] and Pennypacker.[32] The formalism developed here grew out of some previous thoughts of the author,[27] which had led to a more primitive description language which was later abandoned in view of its inefficiency. However, some of the basic ideas relevant to that work have been retained: these have been discussed in the previous section.

Given an environment $\langle U, \mathcal{P} \rangle$ and a concept C, a property is called Directly Relevant to a non-empty concept if and only if it has at least one value whose intersection with the concept is empty.

A property P is called relevant to a non-empty concept C with respect to a family $\mathcal{F}$ of properties if and only if either it is directly relevant to C or if there exists a property $Q(\neq P)$ in $\mathcal{F}$ with a value q such that $q \cap C$ is non-empty and P is relevant to $q \cap C$ with respect to $\mathcal{F}$.

In short, a property is not relevant to a concept when knowing about the value of this property for an object does not (either by itself or in conjunction with other properties) help in the recognition of the object as belonging in the concept. This statement will be formalized presently; the following definitions and theorem will be needed for this formalization.

Given an environment $\langle U, \mathcal{P} \rangle$ and a concept C, a finite subfamily $\{P_1, P_2, \ldots P_n\}$ of $\mathcal{P}$ is called sufficient for C if

and only if either

(Si) $n = 1$ and C has non-empty intersections with more than one value of P_1

or

(Sii) $C = \bigcap_{k=1}^{n} p_{ij_i}$ where $p_{ij_i} \in P_i$ and there is no subset of $\{P_i | 1 \leq i \leq n\}$ with this property. (Note: n can be 1 in this case also.)

A set of properties $\mathfrak{F}$ is called a <u>sufficiency family</u> for a concept C if either $\mathfrak{F}$ is sufficient for C according to (Sii) above or there is a member $P \in \mathfrak{F}$ which is sufficient for C by (Si) above and $\mathfrak{F}$ is the union of P with some sufficiency family of each of the non-empty intersections of C with the values of P.

<u>Theorem 4.2:</u> Let $\{P_i | (1 \leq i \leq n)\}$ be a sufficiency family for C. Let $p_i \in P_i$ for each i. Then either $p_1 \cap p_2 \cap \ldots \cap p_n \subseteq C$ or $p_1 \cap p_2 \cap \ldots \cap p_n \cap C = \emptyset$.

<u>Proof:</u> If there is no property P_k in $\{P_i\}$ such that P_k has more than one value with non-empty intersections with C, then $C = p_1 \cap p_2 \cap \ldots \cap p_n$ and the theorem is evident. Let the theorem be true if there are k properties in $\{P_i\}$ with more than one value with non-empty intersections with C. The theorem is true for $k = 0$. If it is true for $k = m$, let $k = m+1$. Assume (without loss of generality) that P_1 has more than one value with non-empty intersection with C and for any $p' \in P_1$ such that $C \cap p' \neq \emptyset$, $C \cap p'$ has a sufficiency family which is a subset

of $\{P_i\}$. If $C \cap p_1 = \emptyset$, then the theorem follows immediately. Otherwise the sufficiency family for $C \cap p_1$ which is a subset of $\{P_i\}$ contains less than or equal to m properties which have more than one value with non-empty intersection with $C \cap p_1$. Hence, $p_1 \cap p_2 \cap \ldots \cap p_n \subseteq C \cap p_1 \subseteq C$ by induction hypothesis. This theorem leads to the following explication of the significance of relevant properties.

<u>Theorem 4.3:</u> Let $\mathfrak{F} = \{P_1, P_2 \ldots P_n\}$ be a sufficiency family for the concept C. Let $P_1 \in \mathfrak{F}$ be not relevant to C with respect to $\mathfrak{F}$. Let $P_1 = \{p_{11}, p_{12}, \ldots p_{1m}\}$. For any set $\{p_i | p_i \in P_i, 2 \leq i \leq n\}$ if $\emptyset \neq p_{11} \cap p_2 \cap \ldots \cap p_n \cap C$ then for all $k (1 \leq k \leq m)$ $p_{1k} \cap p_2 \cap \ldots \cap p_n \cap C \neq \emptyset$.

<u>Proof:</u> By Theorem 4.2 the hypothesis $p_{11} \cap p_2 \cap \ldots \cap p_n \cap C \neq \emptyset$ implies $p_{11} \cap p_2 \cap \ldots \cap p_n \subseteq C$, or $p_{11} \cap p_2 \cap \ldots \cap p_n \cap C = p_{11} \cap p_2 \cap \ldots \cap p_n \neq \emptyset$. If $p_{1k} \cap p_2 \cap \ldots \cap p_n = \emptyset$, then P_1 is directly relevant to $p_2 \cap \ldots \cap p_n \cap C (\neq \emptyset)$ and hence, relevant to C, leading to contradiction.

This theorem indicates that when testing an object for inclusion in a concept, irrelevant properties need not be tested.

Let C be a concept with sufficiency family $\{P_1, P_2, \ldots P_n\}$ and let each property P_i be relevant to C with respect to $\{P_1, P_2, \ldots P_n\}$. Then a list of k lists, headed by the name "C" is called a <u>conception list of C</u> if

either (Ci) $k = 1$, the unique list is headed by the

name "P_i" where $1 \leq i \leq n$ and P_i has more than one value with non-empty intersection with P_i. It is a list of ordered pairs consisting of the names of the values of P_i with non-empty intersections with C together with the names for these intersections.

or (Cii) $n = k$ and each list is headed by a name "P_i" and contains a single ordered pair consisting of the name of the unique value $p_i \in P_i$ which has non-empty intersection with C and of the name "C."

A set of conception lists form a <u>conception</u> of a concept C if and only if it contains a conception list of C and a conception of every concept whose names occur in the conception list. It is clear that a conception list of C satisfying (Cii) above is a conception in itself.

Some of the ideas associated with the above definitions and assertions can be exemplified by considering a specific Universe. Consider an Universe of Discourse, consisting of 40 elements as described below and exhibited in Figure 4.1. (This same basic Universe will also be used in exemplifying the ideas introduced in Section 5.)

The 40 elements will be denoted by the consecutive positive integers. The following subsets of the Universe will be taken to form the elements of the basic partitions.

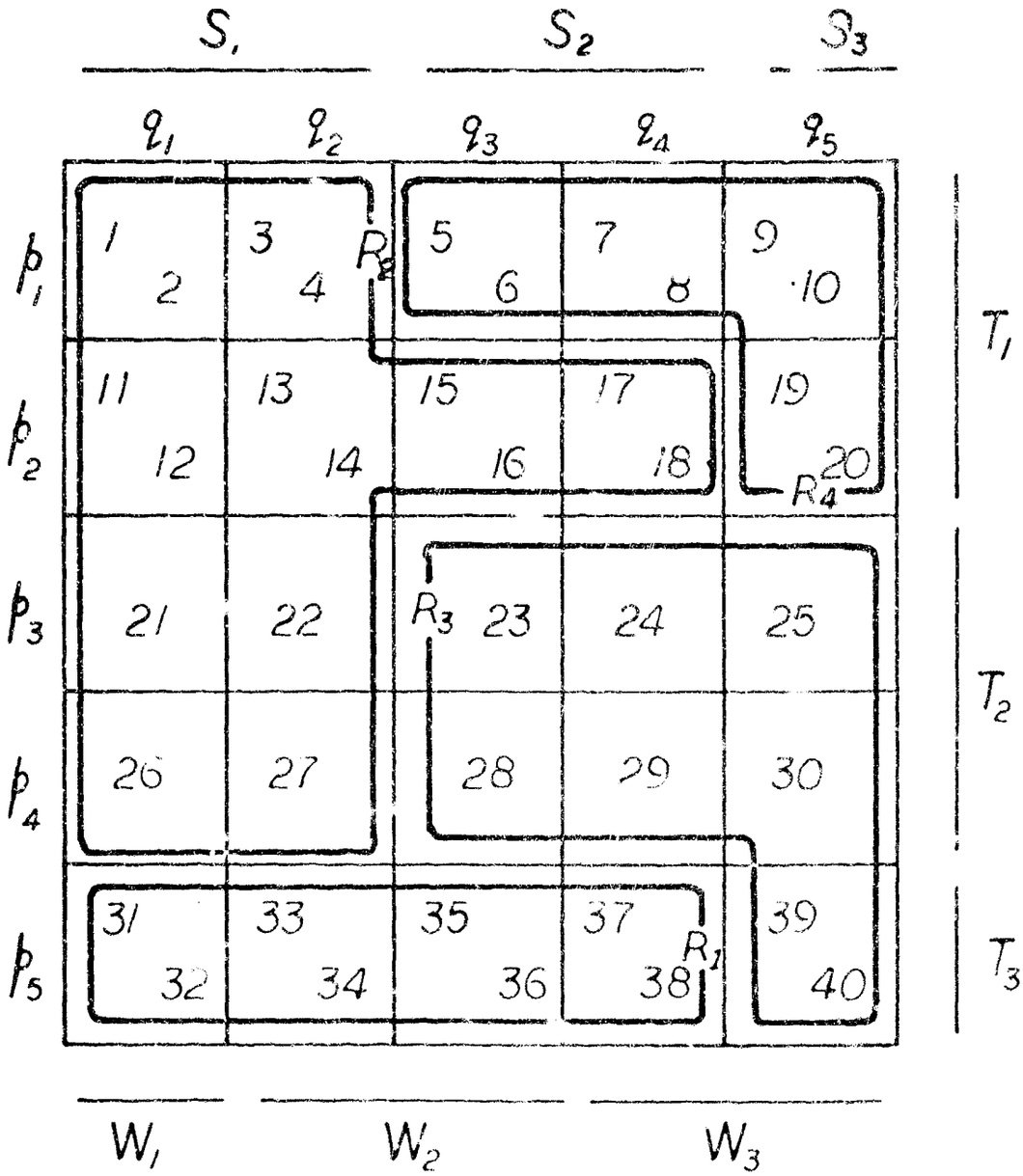

S1
S2
S3
q1
q2
q3
q4
q5
p1
p2
p3
p4
p5
T1
T2
T3
W1
W2
W3
R1
R2
R3
R4
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

Figure 4.1

$p_1 = \{1,2,\ldots,10\}$

$p_2 = \{11,12,\ldots,20\}$

$p_3 = \{21,22,23,24,25\}$

$p_4 = \{26,27,28,29,30\}$

$p_5 = \{31,32,\ldots,40\}$

$q_1 = \{1,2,11,12,21,26,31,32\}$

$q_2 = \{3,4,13,14,22,27,33,34\}$

$q_3 = \{5,6,15,16,23,28,35,36\}$

$q_4 = \{7,8,17,18,24,29,37,38\}$

$q_5 = \{9,10,19,20,25,30,39,40\}$

$R_1 = p_5 - q_5$; $R_2 = q_1 \cup q_2 - p_5 \cup (p_2 \cap (q_3 \cup q_4))$

$R_3 = q_5 \cup \{(q_3 \cup q_4) \cap (p_3 \cup p_4)\} - p_1 - p_2$;

$R_4 = U - R_1 - R_2 - R_3$

$S_1 = q_1 \cup q_2$; $S_2 = q_3 \cup q_4$; $S_3 = q_5$

$T_1 = p_1 \cup p_2$; $T_2 = p_3 \cup p_4$; $T_3 = p_5$

$W_1 = q_1$; $W_2 = q_2 \cup q_3$; $W_3 = q_4 \cup q_5$

Figure 4.2 identifies the basic sets and the elements of the Universe.

There are five properties in this environment

$S = \{S_1, S_2, S_3\}$

$T = \{T_1, T_2, T_3\}$

$W = \{W_1, W_2, W_3\}$

$$R = \{R_1, R_2, R_3, R_4\}$$

$$p = \{p_1, p_2, p_3, p_4, p_5\}$$

$$q = \{q_1, q_2, q_3, q_4, q_5\}$$

$\{p,q\}$ is a fine structure family for this environment; so is $\{p,W,S\}$. The family $\{p,q\}$ is a full fine structure family, while $\{p,W,S\}$ is not full. In the present discussion, p and q will be taken to be input properties, yielding the real environment $\langle U, \{S,T,W,p,q,R\}, \{p,q\}\rangle$. In this environment $\{11,13\}$ is not a concept, for example.

If one considers the concept T_1, it can be seen that q is not relevant to it with respect to $\{p,q,T\}$. However, q is relevant to T_1 with respect to $\{q,W,p\}$. R is not directly relevant to q_3, although it is relevant to q_3 with respect to $\{p,R\}$.

The conception of the concept $A = W_2 \cap R_2$ (i.e., the set $\{3,4,13,14,15,16,22,27\}$) can be written variously.

$$\begin{array}{l} A \\ | \\ W - (W_2, A) \\ | \\ R - (R_2, A) \end{array}$$

would be a possible (and the shortest possible) conception. One other would be

$$\begin{array}{l} A \\ | \\ p - (p_2, B) - (p_3, C) - (p_4, D) - (p_1, G) \end{array} \qquad \begin{array}{l} B \\ | \\ q - (q_2, E) - (q_3, F) \end{array}$$

$$\begin{array}{ccccc} C & D & E & F & G \\ | & | & | & | & | \\ p - (p_3, C) & p - (p_4, D) & p - (p_2, E) & p - (p_2, F) & p - (p_1, G) \\ | & | & | & | & | \\ q - (q_2, C) & q - (q_2, D) & q - (q_2, E) & q - (q_3, F) & q - (q_2, G) \end{array}$$

using $\{p,q\}$ as a sufficiency family. Another, somewhat shorter, would be the following

$$\begin{array}{l} A \\ | \\ p - (p_2, B) - (p_1, G) - (p_4, D) - (p_3, C) \end{array} \qquad \begin{array}{l} B \\ | \\ p - (p_2, B) \\ | \\ W - (W_2, B) \end{array}$$

$$\begin{array}{ccc} C & D & G \\ | & | & | \\ p - (p_3, C) & p - (p_4, D) & p - (p_1, G) \\ | & | & | \\ q - (q_2, C) & S - (S_1, D) & q - (q_2, G) \\ & | & \\ & W - (W_2, D) & \end{array}$$

which uses (p,q,S,W) as sufficiency family. It can be seen easily that (p,q,W) or (p,S,W) could be used as sufficiency families also. It can also be seen that the size of the sufficiency family used has no strong effect on the size of the conception. While (R,W) is a very effective sufficiency set for A, $\{p,q\}$ is not. The size of the conception gets smaller when we augment this last sufficiency set to (p,q,S,W). Changing (p,q,S,W) to (p,q,W) actually decreases the size of the conception.

It should also be noted that when one uses a full fine structure sufficiency family, the conception of any concepts other than values of fine structure properties and their intersections end up containing the conception of every object

contained in the concept. This is the consideration which leads to the need for properties other than input properties for purposes of description. This point will be discussed again in later sections.

4. A Recognition Algorithm using Conceptions

The importance of conceptions arises because there exists an algorithm which can recognize a given object in a real environment as belonging to a concept whose conception is given. This algorithm will be given later, after some other ideas associated with recognition have been discussed. However, the basic idea involved in the algorithm can be indicated here quite easily.

Let there be given an object $(P_1, p_1; P_2, p_2; \ldots; P_n, p_n)$ and the conception of a concept C. One can determine whether the object belongs to the concept as follows. If the conception list of C includes only one relevant property (according to (Ci)) P, then C has non-empty intersections with more than one value of P. Assume for simplicity that P is an input property. Then the object indicates the value of P in which it is contained. If this value does not occur in the conception list of C, then the object is not contained in C. (See Lemma 4.7 below.) On the other hand, if the object is contained in a value p_i of P, then one does not know for certain that the object is contained in C. (A red ball is not a red ball !) One then interrogates the conception of $C \cap p_i$, whose name appears in the conception list of C. The program then recursively determines if the object is contained in $C \cap p_i$. The adequacy of this procedure is indicated in Lemma 4.4 below.

One has to be careful of one thing, however, that

the recursive determination of the containment of the object in $C \cap p_1$ does not involve one in an infinite loop. That this does not occur is indicated by Lemma 4.6 below, which indicates that if a property occurs alone in the conception list of C, then further tests are unnecessary.

One also needs to discuss the case where a set of properties P, P', P"... occur in the conception of C according to (Cii) above. If this is an unit set, then the procedure is as indicated above, except that the name of $C \cap p_1$ is C and by Lemma 4.6 below the object is known to be contained in C. Otherwise, C has the form $p_1 \cap p' \cap p'' \cap \ldots$ and one merely tests successively to see if the object is contained in p', p", etc. till the list is exhausted. The validity of this process is brought out by Lemma 4.5 below.

The following four lemmata, although quite trivial are included here for completeness and to establish that the structure "conception" is designed with the care that should go into the design of every complicated data structure, no matter how complicated.

<u>Lemma 4.4:</u> If $p_1 \in P_1$ has non-empty intersection with C and if X is an object with non-empty intersection with p_1 then $X \sim C$ if and only if $X \sim C \cap p_1$.

<u>Proof:</u> The "if" part is obvious. For the "only if" part, one notes that if $X \cap p_1 \neq \emptyset$ then $X \sim p_1$ by Theorem 4.1 and by the fact that p_1 is a concept. (Note that if $X \not\sim p_1$ then $X \cap \overline{p_1}$ is non-empty and hence, some objects of X are elements of

p_1 and others are not, contradicting Theorem 4.1.) Hence, if $X \subseteq C$, then $X = X \cap p_1 \subseteq C \cap p_1$.

<u>Lemma 4.5:</u> If $C = p_{1i_1} \cap p_{2i_2} \cap \ldots \cap p_{ni_n}$ and $X \cap p_{1i_1} \neq \emptyset$ then $X \subseteq C$ if and only if $X \subseteq p_{2i_2} \cap \ldots \cap p_{ni_n}$.

<u>Proof:</u> If $n = 1$ the set $\{p_{2i_2}, \ldots p_{ni_n}\}$ is empty and their intersection is the Universe. Also, $X \subseteq p_{1i_1}$ & $C = p_{1i_1}$. Hence, $X \subseteq C$. The converse follows trivially since $X \subseteq C \subseteq U$.

Let $n > 1$.

If $X \subseteq p_{1i_1}$ & $X \subseteq p_{2i_2} \cap \ldots \cap p_{ni_n}$ then $X \subseteq p_{1i_1} \cap \ldots \cap p_{ni_n} = C$. Again, if $X \subseteq C = p_{1i_1} \cap \ldots \cap p_{ni_n}$ then $X \subseteq p_{2i_2} \cap \ldots \cap p_{ni_n}$.

<u>Lemma 4.6:</u> If $\{P\}$ is sufficient for C, and p_1 is a value of P such that $C \cap p_1 \neq \emptyset$, then $\{P\}$ is sufficient for $C \cap p_1$ only if $C = p_1$.

<u>Proof:</u> Only one value of P has non-empty intersection with $C \cap p_1$. Hence, $\{P\}$ can be sufficient for $C \cap p_1$ only by virtue of (Sii) above. Hence, if $\{P\}$ is sufficient for $C \cap p_1$ then $C = p_1$.

<u>Lemma 4.7:</u> If $p_1 \cap C = \emptyset$ & $X \subseteq p_1$ then $X \cap C = \emptyset$.

The proof is evident.

It was assumed in our qualitative discussion that the

properties mentioned in the conceptions are all input properties and hence, are listed in the object. In this case, determination of the truth of statements like $X \subseteq p_1$ above is a trivial matter of look-up.

However, the test to be performed to find whether $X \subseteq p_1$ is not always such a straight-forward process. If P is not a member of $\mathcal{P}'$, one needs extra information to know if $X \subseteq p_1$. This information will be codified in the present recognition scheme by stipulating that conceptions for the values of P is available in an acceptable form. The following definitions clarify what is meant by "acceptable" here.

The <u>description list of the Universe</u> is a list of lists headed by the name "U." Each list in the list of lists is headed by the name of a property in $\mathcal{P}$, relevant to U with respect to $\mathcal{P}'$ and there is a list headed by the name of each relevant property in $\mathcal{P}$. The list headed by P is a list of ordered pairs containing the names of the values of P and the name of their intersections with U.

A set of lists is a <u>description list structure of the Universe</u> if it contains the description list of the Universe and a conception for all concepts whose name occurs in the description list of the Universe.

It is to be noted that given a real environment $\langle U, \mathcal{P}, \mathcal{P}' \rangle$, the description list of the Universe is unique However, the conception list of any other concept is not necessarily unique: nor is a description list structure of the

Universe unique. However, the following theorem is true.

Theorem 4.7: In an environment $\langle U, \mathcal{P}, \mathcal{P}' \rangle$, each element of $\mathcal{P} - \mathcal{P}'$ is relevant to the Universe with respect to $\mathcal{P}'$.

Proof: Since $\mathcal{P}'$ is a fine structure family of properties, any concept is a Boolean function of value of elements of $\mathcal{P}'$. Also, since each element P of $\mathcal{P}$ is a non-trivial partition, any value p of P is a proper subset of the Universe. Hence, there is an object X which is not contained in p. Let $X = p_1 \cap p_2 \cap \ldots \cap p_n$ where n is the cardinality of $\mathcal{P}'$. Then, since $p \cap p_1 \cap p_2 \cap \ldots \cap p_n = \emptyset$, P is directly relevant to $p_1 \cap p_2 \cap \ldots \cap p_n$ and hence, relevant to the Universe.

Hence, the name of every element of $\mathcal{P} - \mathcal{P}'$ heads some list in the description list of the Universe. However, this does not necessarily make a description list structure of the Universe "acceptable" information for finding whether an object X is contained in some value p of a non-input property. Some further definitions are needed.

Given a description list structure of the Universe, a property $P \in \mathcal{P}$ is called pre-defined if and only if either

(Di) $P \in \mathcal{P}'$

or (Dii) All properties whose names occur in the conceptions of all values of P are pre-defined properties.

A description list structure of the Universe is called valid if every element of $\mathcal{P}$ is pre-defined. To exemplify

validity, the following is a description list of the Universe shown in Figure 4.1 in Section 3. The description list of U is

$$
\begin{array}{l}
U \\
\mid \\
R - (R_1, R_1) - (R_2, R_2) - (R_3, R_3) - (R_4, R_4) \\
\mid \\
S - (S_1, S_1) - (S_2, S_2) - (S_3, S_3) \\
\mid \\
T - (T_1, T_1) - (T_2, T_2) - (T_3, T_3) \\
\mid \\
W - (W_1, W_1) - (W_2, W_2) - (W_3, W_3)
\end{array}
$$

A possible description list structure of U might contain in addition to the above, the following conceptions

$$
\begin{array}{l}
R_1 \\
\mid \\
q - (q_1, \alpha) - (q_2, \beta) - (q_3, \gamma) - (q_4, \delta)
\end{array}
\qquad
\begin{array}{l}
\alpha \\
\mid \\
p - (p_5, \alpha) \\
\mid \\
q - (q_1, \alpha)
\end{array}
$$

$$
\begin{array}{l}
\beta \\
\mid \\
p - (p_5, \beta) \\
\mid \\
q - (q_2, \beta)
\end{array}
\qquad
\begin{array}{l}
\gamma \\
\mid \\
p - (p_5, \gamma) \\
\mid \\
q - (q_3, \gamma)
\end{array}
\qquad
\begin{array}{l}
\delta \\
\mid \\
p - (p_5, \delta) \\
\mid \\
q - (q_4, \delta)
\end{array}
$$

$$
\begin{array}{l}
R_2 \\
\mid \\
S - (S_1, M) - (S_2, N)
\end{array}
\qquad
\begin{array}{l}
M \\
\mid \\
T - (T_1, J) - T_2(K)
\end{array}
$$

$$
\begin{array}{l}
N \\
\mid \\
q - (q_3, F) - (q_4, H)
\end{array}
\qquad
\begin{array}{l}
F \\
\mid \\
p - (p_2, F) \\
\mid \\
q - (q_3, F)
\end{array}
\qquad
\begin{array}{l}
H \\
\mid \\
p - (p_2, H) \\
\mid \\
q - (q_4, H)
\end{array}
$$

$$\begin{array}{l} J \\ | \\ T - (T_1, J) \\ | \\ S - (S_1, J) \end{array} \qquad \begin{array}{l} K \\ | \\ T - (T_2, K) \\ | \\ S - (S_1, K) \end{array} \qquad \begin{array}{l} R_3 \\ | \\ S - (S_2, L) - (S_3, Z) \end{array}$$

$$\begin{array}{l} L \\ | \\ S - (S_2, L) \\ | \\ T - (T_2, L) \end{array}$$

$$\begin{array}{l} Z \\ | \\ T - (T_2, \varepsilon) - (T_3, I) \end{array} \qquad \begin{array}{l} \varepsilon \\ | \\ T - (T_2, \varepsilon) \\ | \\ q - (q_5, \varepsilon) \end{array} \qquad \begin{array}{l} I \\ | \\ T - (T_3, I) \\ | \\ S - (S_3, I) \end{array}$$

$$\begin{array}{l} R_4 \\ | \\ p - (p_1, \varphi) - (p_2, \Psi) \end{array} \qquad \begin{array}{l} \Psi \\ | \\ p - (p_2, \Psi) \\ | \\ q - (q_5, \Psi) \end{array} \qquad \begin{array}{l} \varphi \\ | \\ S - (S_2, \xi) - (S_3, \eta) \end{array}$$

$$\begin{array}{l} \xi \\ | \\ S - (S_2, \xi) \\ | \\ p - (p_1, \xi) \end{array} \qquad \begin{array}{l} \eta \\ | \\ q - (q_5, \eta) \\ | \\ p - (p_1, \eta) \end{array}$$

$$\begin{array}{l} S_1 \\ | \\ q - (q_1, W_1) - (q_2, q_2) \end{array} \qquad \begin{array}{l} S_2 \\ | \\ q - (q_2, q_3) - (q_4, q_4) \end{array}$$

$$\begin{array}{l} S_3 \\ | \\ q - (q_5, S_3) \end{array}$$

W_1 $\qquad$ W_2

$|$ $\qquad$ $|$

$q - (q_1, W_1)$ $\qquad$ $q - (q_2, q_2) - (q_3, q_3)$

W_3

$|$

$q - (q_4, q_4) - (q_5, S_3)$

T_1 $\qquad$ T_2 $\qquad$ T_3

$|$ $\qquad$ $|$ $\qquad$ $|$

$T - (T_1, T_1)$ $\qquad$ $T - (T_2, T_2)$ $\qquad$ $T - (T_3, T_3)$

This description list structure would not be valid, since the concept T_1 is described in terms of T, which is not a predefined property; also, descriptions of the concepts q_2, q_3, q_4, whose names occur in the descriptions of values of S, do not occur in the list. In the above list, one replaced the conceptions of T_1, T_2 and T_3 by

T_1 $\qquad$ T_2

$|$ $\qquad$ $|$

$p - (p_1, p_1) - (p_2, p_2)$ $\qquad$ $p - (p_3, p_3) - (p_4, p_4)$

T_3

$|$

$p - (p_5, T_3)$

and added the conceptions

q_2 $\qquad$ q_3 $\qquad$ q_4

$|$ $\qquad$ $|$ $\qquad$ $|$

$q - (q_2, q_2)$ $\qquad$ $q - (q_3, q_3)$ $\qquad$ $q - (q_4, q_4)$

$$\begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \\ | & | & | & | \\ p - (p_1, P_1) & p - (p_2, P_2) & p - (p_3, P_3) & p - (p_4, P_4) \end{array}$$

the description list structure would be valid.

<u>Theorem 4.8:</u> Given any object X, a property P, $p \in P$, and a valid description list structure of a finite Universe, it can be determined by a finite process whether $X \subseteq p$.

<u>Proof:</u> One first associates an integer with every property as follows. With each element of P' one associates the integer 1. For any other property P, one associates an integer n_P defined as follows

$n_P = 1 + \max \{n_{P'} \mid P'$ occurs in the conception of some value of $P\}$.

With a valid conception of the Universe, n_P is uniquely defined for every property. The proof is by induction over n_P.

If $n_P = 1$, then $P \in P'$ and the name of a value of P occurs in X. $X \subseteq p$ if and only if this name is identical with p. Hence, the theorem is true for P if $n_P = 1$.

Let the theorem be true if $n_P \leq k$. If $n_P = k+1$, then in the conception of p, only such properties Q occur such that $n_Q \leq k$.

If the conception list of p contains the name of more than one value of a property Q, then $X \subseteq p$ only if $X \cap q \neq \emptyset$ for exactly one $q \in Q$. (Otherwise X is properly contained in two disjoint sets.) Since there are only a finte number of such values, the containment of X in one of them can be determined by a finite process.

If the conception list of p has one value from a finite number of properties, then, since for each property Q in this list $n_Q \leq k$, the containment of X in p can be determined by a finite process.

To understand the way the integers $\{n_p\}$ are associated with the properties in the above proof, one can once more invoke the valid description list structure of the Universe exemplified in Figure 4.1. The integers associated with the various properties according to the scheme described in Theorem 4.8 is shown in Table 4.1. It will be also noticed that conceptions for q_5, p_5 and q_1 did not have to be included in the valid description list structure since their names never occured in the right hand side of any ordered pair in any of the conceptions.

In view of the discussions of this section and the last, the reader will be able to convince himself that the process indicated by the recursive flow chart shown in Figure 4.2 can effectively determine whether an object X belongs to a concept C, if the conception of C and a valid description list structure of the Universe is available. In this flow chart three push down stacks are used: j, P and C. The list L is a list of ordered pairs which is to be empty at the first entry to the program. The name of the concept is to be entered in stack C before starting the program. In the flow chart, all variables are to be interpreted in the normal manner as denoting the content of the address named. In the case of stacks,

P	1
q	1
R	3
S	2
T	2
W	3

Table 4.1

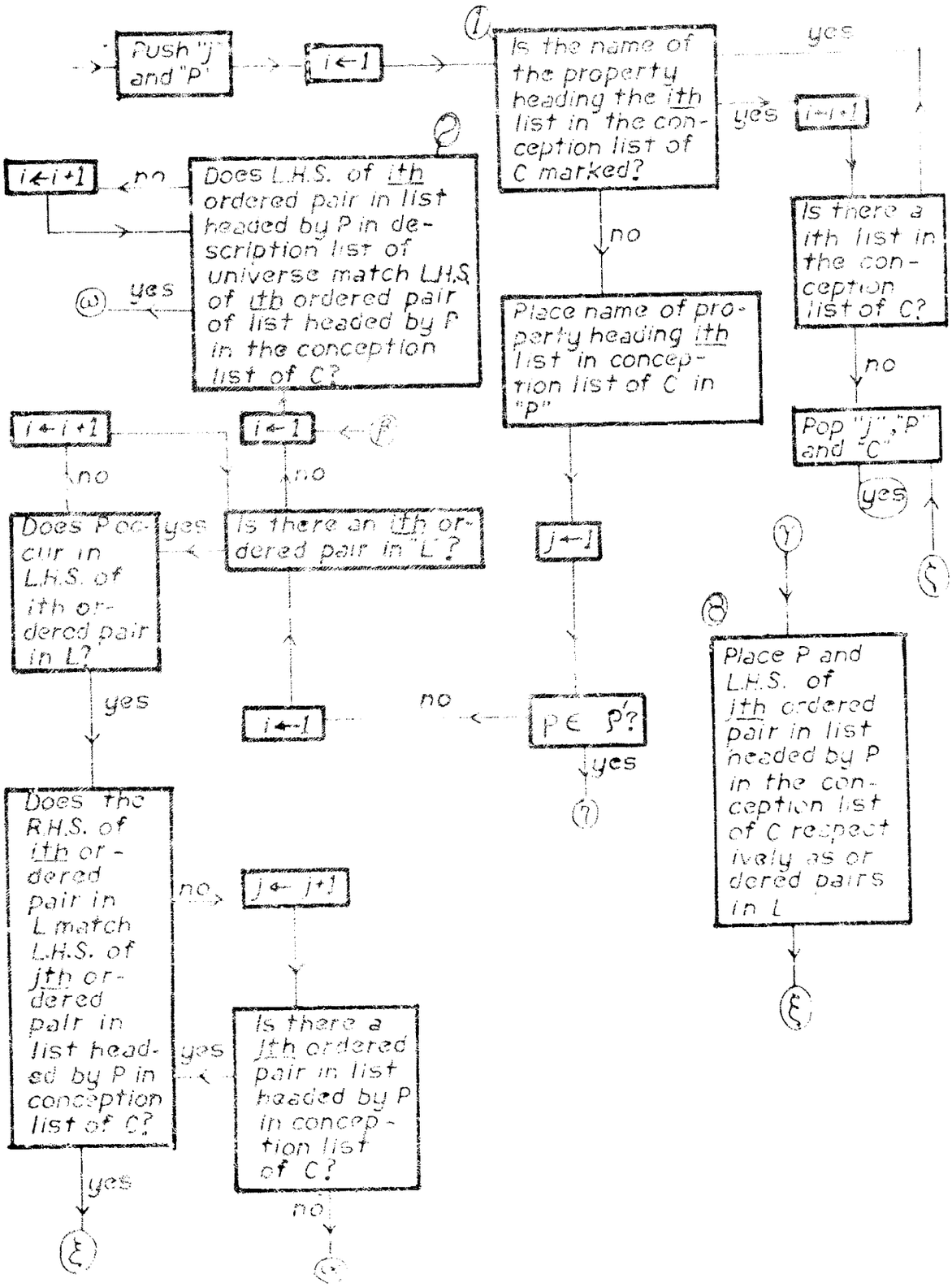


Figure 4.'
(part 1)

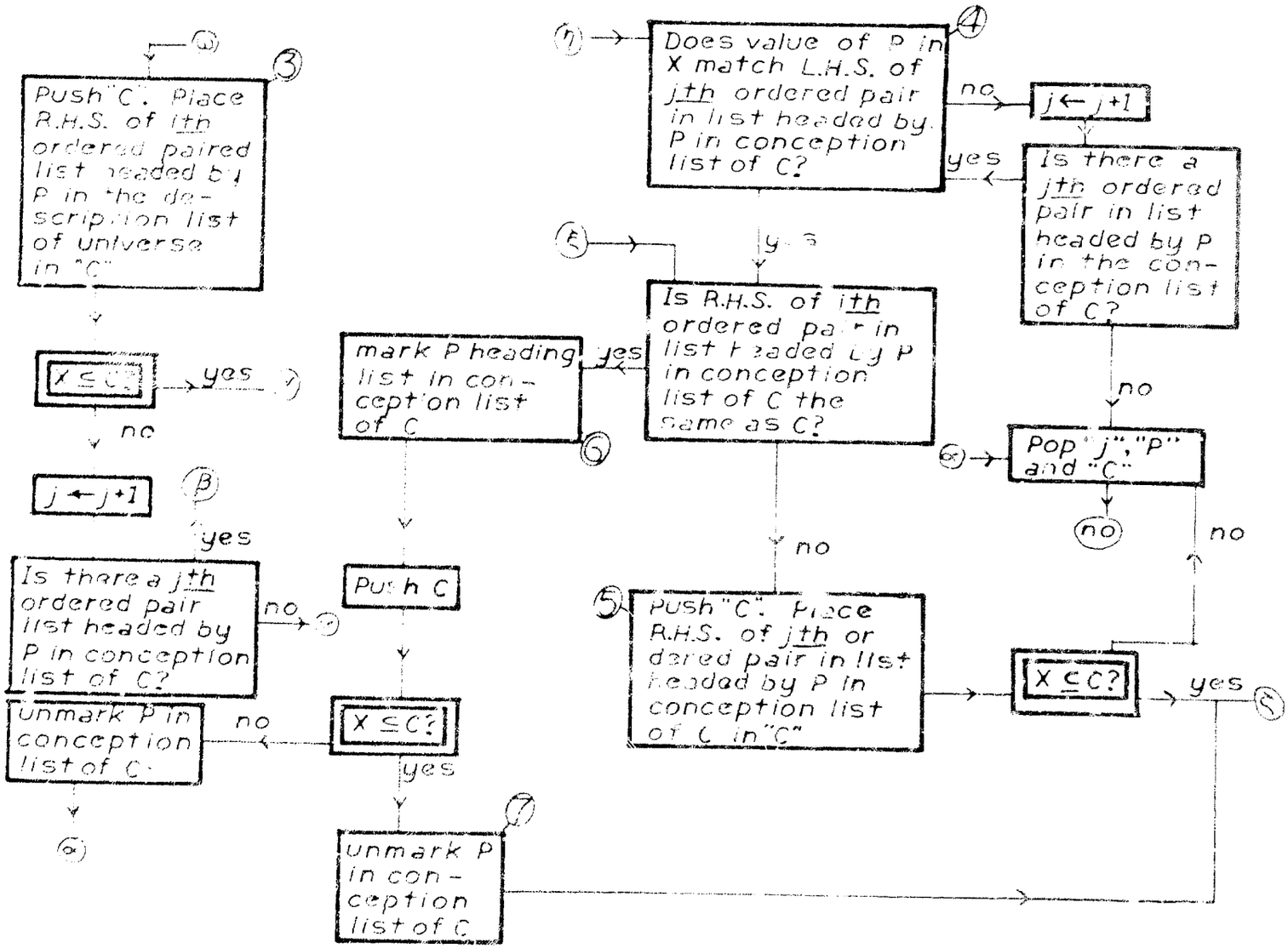


Figure 4.2
(part 2)

also, the same convention is followed except that the content of the latest call is denoted by the stack name. When the stacks are referred to rather than their contents, quotes are used.

One can recall at this point the different conceptions for the concept A in the previous example. It can be seen that the second conception (having (p,q) as sufficiency set), although the most space consuming, could be used most efficiently, because the route marked 2 in Figure 4.1 (necessitating the use of the description list structure of the Universe) is never used. On the other hand, the first conception shown (using W and R) needs a description list structure of the Universe.

It may be of some interest to indicate by an example the way the valid description list structure is used in the operation of the flow chart indicated in Figure 4.1 for recognition of an object being a subset of the concept A.

The assumption will be made that the conception of A being used happens to be

$$\begin{array}{l} A \\ \vdots \\ W - (W_2, A) \\ \vdots \\ R - (R_2, A) \end{array}$$

and the object involved is $(p, p_3, \ldots, q_4)$.

Initially, the list L is empty and "C" contains the name "A." The test in Box 1 finds W unmarked in the conception list of A and the name W is entered into "P." Since W ≠ "",

the list headed by W in the description list of the Universe is searched by box 2 for the name W_2, which matches the value of W in the conception list of A. Then box 3 re-enters the program recursively to test if the object is in W_2 (the intersection of W_2 with the Universe). q is unmarked in the conception list of W_2 and hence is entered in "P." Since $q \neq$ "", box 4 matches the value q_2 in the object to q_2, the L.H.S. of the first ordered pair in the conception list of W_2. The intersection of q_2 with W_2 being distinct from W_2, box 5 enters the program recursively, to test if the object is in q_2. The list heading q in description of q_2 being unmarked and since $q \neq$ "", box 4 matches the value q_2 (L.H.S. of first ordered pair in list headed by q in conception list of q_2) to the value of x in the object. The R.H.S. of this ordered pair being q_2, box 6 marks q in conception list of q_2 and re-enters the program testing for A being contained in q_2. On this entry box 1 fails to find any unmarked property of q_2 so on the recursive "yes" exit, box 7 unmarks q (which has popped back in P following the recursive entry). As a result there is a "yes" exit into box 8, which places (A, W_2) into the list L. The intersection of W_2 with A_2 being A again, box 9 marks W in conception list of A and re-enters the program testing X as subset of A, but with W marked. Box 1 isolates R, the next unmarked property in conception list of A, and places it in P. Since $R \neq$ "" and an ordered pair with R at L.H.S. exists in L, box 2 isolates R_2 in list headed by R in description list of Universe [illegible]

R_2 in the conception list of A. As a result, box 3 re-enters the program, testing the object as subset of R_2. The property S occuring in conception list of R_2 being unmarked, not a member of P, and not occuring in L, box 2 matches S_1 in the description list of the Universe with S_1 occuring in the conception list of R_2. Hence, box 3 re-enters the program testing the object as a subset of S_1. q is unmarked, not in L and occurs in P. So box 4 matches q_2 in the conception list of S_1 with the value of q in the object. Hence, box 5 tests the object as a subset of q_2 (as it did testing for W_2) and on success, box 8 places (S, S_1) in the list L, following which box 5 re-enters the program testing the object as a subset of M (the intersection of S_1 with R_2). T is not marked in the conception list of M; nor is it in P; hence, box 2 isolates the value T_1 in the description list of the Universe and box 3 tests the object as a subset of T_1. p, being unmarked in conception list of T_1 not occuring in L and not being a member of P', an attempt is made in box 4 to match the value of p in the object (p_3) with the value p_1 and p_2 occuring in the conception list of T_1. This results in a failure exit and box 3 tests the object as a subset of T_2. This succeeds (in the same way as $X \subseteq W_2$ succeeded). Hence, (T, T_2) is placed in L by box 8 and box 5 tests the object as a subset of K. The first unmarked property in the conception list of K (T), occurs in L (saving the trouble of a re-evaluation), its value (T_2) matches the value of T in K. Box 6 marks T in the conception list of K,

tests for the object as subset of K with T marked, finds S, the next unmarked property of K in L; matches its value (S_1) in L with that in the conception list of K; hence, S is marked in the conception list of K and the next re-entry exits, unmarking properties of K, then recognizing " and hence, box 8 places (R, R_2) in L. The intersection of R_2 with A being A, R is marked in the conception list of A and the next recursive entry exits with success, finding the object as a subset of A and unmarking the conception list of A.

5. Conjunctive and Simple Concepts

As has been pointed out before, every description language is constructed out of a set of predicates and a mode of combination of predicates to yield compound statements with only one free variable so that in its interpretation it denotes a subset of the Universe of Discourse. In the languages discussed in this section, the basic predicates are unary also (containing a single variable). In the language discussed so far (whose sentences are conceptions) each set is described either as the union of a class of disjoint sets or as the intersection of a class of property-values. The basic building blocks of the concepts then are the class of concepts each of which are intersections of a class of property-values. The building blocks will be called "conjunctive concepts" for the purposes of the present discussion.

The size of a conception describing a conjunctive concept certainly depends on the number of property values to be intersected to obtain the concept. The size of a conception describing concepts other than conjunctive concepts is larger than the sum of the sizes of the conceptions describing the disjoint conjunctive concepts of which the given concept is the union. If there is more than one conception for the same concept, then it is quite difficult to decide without careful study as to which of the given conceptions has the minimum size. It can be surmised that in a real environment where $P = P'$ and is full, the conception of a concept will be smaller, the fewer

the number of conjunctive concepts used as building blocks for the concept. In what follows the supposition will be made that if a conception describes a concepts as a conjunctive concept, then this is the smallest conception for the concept. Whether such a conception exists or not for a concept certainly depends on the environment, i.e., on the structure of the properties available.

Given a certain real environment and a certain conception, it may be of interest to find a shorter conception which denotes the same concept. A method for doing this has been developed by J. C. Pennypacker.[32] Other related methods developed by him will be discussed later.

Given an Universe (for instance, the set of all occurances of bit-configurations on a square grid of photo-cells) whose elements can be coded into computer inputs, one can generally come up with some fine structure family of properties for that Universe. In the case of the square grid of photo-cells, for instance, the excitation value of a particular photo-cell divides the set of all bit-configurations on the grid into two disjoint subsets. The family of properties defined by the class of all photo-cells forms a full fine structure family. The Universe of all configurations on a chess board has as fine structure family the occupancy of each square on the board. (As an aside, this fine structure family, of course, is not full - not more than one white square can be occupied by a black biship, for instance.) One can surmise with some confidence

that finding a fine structure family of properties for an Universe is a problem that can be safely relegated to the intuition of the experimenter.

However, in most Universes (except those specially designed by psychologists for specific tests) one is specially interested in having descriptions for certain given concepts (the set of all "B"'s on a photo-cell grid: the set of all forcing situations on a chess-board, etc.). Generally these concepts are not conjunctive concepts if one restricts oneself to the input properties alone. In the interest of practicable brevity, it is essential to have properties in the environments such that the conceptions for these concepts be short and (if practicable) conjunctive. A large part of the effort in the field of pattern recognition is directed towards the search for suitable properties (the values of these properties are called "features" in the field). Often acceptable looking features are assumed to exist and statistical methods are developed to reduce the probability of incorrect classification by choosing the least harmful conjunctive concept to approximate the concept at hand. Concepts other than conjunctive ones are often succinctly expressed by invoking modes of combination other than the ones used in Logic. These will be discussed appropriately later. Meanwhile, one may be tempted to pose the following general problem, "Given a class of concepts in a given real environment $\langle U, P, P' \rangle$, to enlarge the class of properties such that each concept in the class is conjunctive." In this form,

the problem has a trivial solution, "Use each concept in the class togehter with its complement as a property." This, of course, does not reduce the memory size in any way. For a more realistic posing of the problem, one needs to take into account the size increase involved in incorporating these new properties into the description list structure of the Universe. The problem called, "feature extraction" is closely related to this problem. To the best of the author's knowledge, such a problem has not been taken up in the literature in this form. Also, since the measure of size is highly language dependent, the development of more powerful description languages is a pre-requisite.

The major point that will be considered in this section will be a mode of combining unary predicates which renders it easy to have short descriptions, not only for conjunctive concepts but for a much larger class of concepts which shall be called "Simple" concepts. The theory developed for the purpose will also indicate methods for describing non-simple concepts by the use of simple concepts which approximate it. Also, in a later chapter it will be shown how one can use this language for "generalization" or concept-learning.

At the present level of development of this theory, no distinction is made between input and non-input properties. Since given an environment $\langle U, \mathcal{P} \rangle$, $\mathcal{P}$ itself is a fine structure family, one can say that the theory deals with a real environment $\langle U, \mathcal{P}, \mathcal{P} \rangle$. At the present stage of thought it is not clear

whether the extension of the theory to the case where $\mathcal{P}' \neq \mathcal{P}$ in any but the most trivial way will be of use or not. A large amount of theoretical development also is needed because the class of simple concepts indicate close relationships to topologies on the one hand and with decomposition of games on the other hand. This will be indicated in detail later.

As before finite environments will be considered. Let $\mathcal{P} = \{P_1, P_2 \ldots P_n\}$ and let for each i $(1 \leq i \leq n)$, $P_i = \{p_{i1}, p_{i2}, \ldots p_{ir_i}\}$. Given a concept X, define a set X^s, called the <u>superconcept of X</u> as follows

$$X^s = \bigcap_{i=1}^{n} \bigcup \{p_{ij} | p_{ij} \cap X \neq \emptyset\}$$

That is, for each i, one defines the set X_i which is the union of those values of P_i which have non-empty intersection with X. X^s is obtained by taking the intersection of X_i for all values of i. As an example in the environment indicated in Figure 4.2 the superconcept of the concept $\alpha = \{5,6,7,8,9,10,13,14,15,16,19,20\}$ would be the concept $T_1 \cap (W_2 \cup W_3) \cap (p_1 \cup p_2) \cap (q_2 \cup q_3 \cup q_4 \cup q_5) \cap (R_2 \cup R_4) \cap (S_1 \cup S_2 \cup S_3)$; that is $\alpha^s = \{3,4,5,6,7,8,9,10,13,14,15,16,17,18,19,20\}$. α^s has α as its subset. This is true in general. That is

<u>Theorem 4.9:</u> For any concept X, $X \subseteq X^s$

<u>Proof:</u> $X = X \cap U = X \cap \bigcup_{j=1}^{r_i} p_{ij} = \bigcup_{j=1}^{r_i} \{X \cap p_{ij}\}$

for each i $(1 \leq i \leq n)$.

However,

$$\bigcup_{j=1}^{r_i} \{X \cap p_{ij}\} = \bigcup \{X \cap p_{ij} | X \cap p_{ij} \neq \emptyset\}$$

Hence,

$$X = \bigcup \{X \cap p_{ij} | X \cap p_{ij} \neq \emptyset\}$$

But,

$$X \cap p_{ij} \subseteq p_{ij}$$

Hence,

$$X \subseteq \bigcup \{p_{ij} | X \cap p_{ij} \neq \emptyset\}$$

for each i ($1 \leq i \leq n$)

Hence,

$$X \subseteq \bigcap_{i=1}^{n} \bigcup \{p_{ij} | X \cap p_{ij} \neq \emptyset\} = X^s$$

From this theorem it follows that X^s can be taken as an approximation for X in the sense that any element which is not a member of X^s is certainly not a member of X. As a matter of fact, a much stronger statement can be made regarding the approximating ability of superconcepts. It can be noticed for instance, that $\overline{\alpha^s} = (R_1 \cup R_2 \cup R_3) \cap (W_1 \cup W_2 \cup W_3) \cap (S_1 \cup S_2 \cup S_3) \cap (T_1 \cup T_2 \cup T_3) \cap (p_1 \cup p_2 \cup p_3 \cup p_4 \cup p_5) \cap (q_1 \cup q_2 \cup q_3 \cup q_4 \cup q_5)$ whose complement is the set $\overline{\alpha^s} = \{5, 6, 7, 8, 9, 10, 19, 20\}$ which is a subset of α. This also is true in general: that is

<u>Corollary 4.10:</u> For any concept X, $\overline{X^s} \subseteq X$

<u>Proof:</u> $\overline{X^s} \supseteq \overline{X}$

Hence, $X \supseteq \overline{\overline{X^s}}$

Thus, $\overline{\overline{X^s}}$ and X^s can be looked upon as lower and upper bounds of X. Hence, if one stores descriptions of X^s and $\overline{\overline{X^s}}$, one can recognize various objects as being definitely contained in X, others as definitely not being contained in X.

In addition to the fact that the superconcepts of a concept and its complement yield two approximations to a concept, it is to be noted that they also have rather simple descriptions in a specific language. This is brought out by the following theorem.

<u>Theorem 4.11:</u> For any concept

$$X^s = \overline{\bigcup \{p_{ij} | p_{ij} \in P_i \ , \ P_i \in P, \ p_{ij} \cap X = \emptyset\}}$$

<u>Proof:</u>

$$X^s = \bigcap_{i=1}^{n} \bigcup \{p_{ij} | p_{ij} \in P_i \ , \ p_{ij} \cap X \neq \emptyset\}$$

$$= \bigcap_{i=1}^{n} \overline{\bigcup \{p_{ij} | p_{ij} \in P_i \ , \ p_{ij} \cap X = \emptyset\}}$$

$$= \overline{\bigcup_{i=1}^{n} \bigcup \{p_{ij} | p_{ij} \in P_i \ , \ p_{ij} \cap X = \emptyset\}}$$

$$= \overline{\bigcup \{p_{ij} | p_{ij} \in P_i \ , \ P_i \in P, \ p_{ij} \cap X = \emptyset\}}$$

Hence, the superconcept of any concept X can be described by storing the list of those property values which

have empty intersections with X. Thus, the description of α^s could be stored as

$$\alpha^s = \overline{\bigcup (\{p_3, p_4, p_5, q_1, R_1, R_3, T_2, T_3, W_1\})}$$

Clearly, this new mode of description of concepts makes it necessary to have a new algorithm for determining whether a given object is contained in a superconcept or not. Such a program will be discussed later. Meanwhile, it is worthwhile pointing out that the present language of description (describing the superconcept of a concept and its complement) does not restrict one to storing approximations alone. At some extent, all concepts can be described exactly if one allows in the language the capability of expressing the union of described sets. To see how this can be done the following are introduced.

<u>Theorem 4.11:</u> For all concepts $X^s = (X^s)^s$

<u>Proof:</u>

$$X^s = \overline{\bigcup \{p_{ij} | p_{ij} \in P_i, P_i \in P, p_{ij} \cap X = \emptyset\}}$$

Hence, for any $p_{ij} \in P_i$ such that $P_i \in P$, $p_{ij} \cap X = \emptyset$ implies $p_{ij} \subset \overline{X^s}$ that is $p_{ij} \subset \overline{X}$ implies $p_{ij} \subset \overline{X^s}$; replacing X by X^s one obtains $p_{ij} \subset \overline{X^s}$ implies $p_{ij} \subset \overline{(X^s)^s}$, i.e., $p_{ij} \subset \overline{X}$ implies $p_{ij} \subset \overline{(X^s)^s}$.

Hence,

$$\overline{X^s} = \bigcup \{p_{ij} | p_{ij} \in P_i \ , \ P_i \in P, \ p_{ij} \cap X = \emptyset\}$$

$$\subseteq \bigcup \{p_{ij} | p_{ij} \in P_i \ , \ P_i \in P, \ p_{ij} \subseteq \overline{(X^s)^s}\} \subseteq \overline{(X^s)^s}$$

or

$$X^s \subseteq (X^s)^s$$

however, by Theorem 4.9

$$(X^s)^s \subseteq X^s$$

and the theorem follows.

Concepts which are equal to their superconcepts will be called <u>simple concepts</u>. The superconcept of any concept is simple.

<u>Theorem 4.12</u>: All conjunctive concepts are simple.

<u>Proof:</u> Let X be conjunctive, that is, there exists a subset $P' = \{P_{i_1}, P_{i_2}, \ldots, P_{i_m}\}$ of P and a value $p_{i_k j_k} \in P_{i_k}$ for each k $(1 \leq k \leq m)$ such that

$$X = \bigcap_{k=1}^{m} p_{i_k j_k}$$

We have

$$X^s = \bigcap_{i=1}^{m} \bigcup \{p_{ij} | p_{ij} \cap X \neq \emptyset\} \supseteq \bigcup \{p_{i_k j} | p_{i_k j} \cap X \neq \emptyset\} \supseteq p_{i_k j_k}$$

$$(1 \leq k \leq m)$$

Hence,

$$X^s \supseteq \bigcap_{k=1}^{m} p_{i_k j_k} = X$$

the reverse of equality follows for Theorem 4.9. Hence

$X = X^S$ showing that X is simple.

Since it has been indicated in the previous section that any concept can be described as an union of conjunctive concepts, any concept may be described as the union of simple concepts. However, the number of simple concepts involved in the union may be much smaller than the number of conjunctive concepts needed. All simple concepts which are not conjunctive (the concept a^S in the previous example, for instance) can be stored as a single description.

A very straight forward algorithm exists for testing whether an object is contained in a simple concept or not. More generally, given two simple concepts (it is to be noted that an object, being a conjunctive concept is simple) it is extremely easy to determine if one is contained in the other. This will be established through the following definitions and theorems.

Let K denote the set of the names of all property-value, i.e., the set

$$\{p_{11}, p_{12}, \ldots, p_{1r_1}, p_{21}, \ldots, p_{2r_2}, \ldots, p_{nr_n}\}.$$

With each subset of K, $\{t_1, t_2, \ldots, t_r\}$ one can associate the simple concept

$$X = \overline{\{t_1, t_2, \ldots, t_r\}} = \overline{t_1 \cup t_2 \cup \ldots t_r}$$

Let $H(\{t_1, t_2, \ldots, t_r\})$ denote this concept X. Thus, H is a mapping from the set of all subsets of K to the set of all simple concepts. It has been shown by Windeknecht that a

sublattice of the set of all subsets of K, considered as a lattice under inclusion is anti-homomorphic to the lattice of all simple concepts under inclusion.[33] For the purpose of this section it is only needed to show that the set of all subsets of K, partially ordered by inclusion is anti-homomorphic to the set of all simple concepts, partially ordered by inclusion. The mapping H, described above is the anti-homomorphism involved. This is shown in the following theorem.

<u>Theorem 4.13:</u> Let α and β be subsets of K and let $H(\alpha)$ and $H(\beta)$ be the corresponding simple concepts. Then $\alpha \subseteq \beta$ implies $H(\beta) \subseteq H(\alpha)$.

<u>Proof:</u> If $\alpha \subseteq \beta$ then $p_{ij} \in \alpha$ implies $p_{ij} \in \beta$ for each $p_{ij} \in P_i$, $P_i \in P$.

Hence,

$$\bigcup \{p_{ij} | p_{ij} \in \alpha\} \subseteq \bigcup \{p_{ij} | p_{ij} \in \beta\}.$$

Hence,

$$H(\beta) = \overline{\bigcup \{p_{ij} | p_{ij} \in \beta\}} \subseteq \overline{\bigcup \{p_{ij} | p_{ij} \in \alpha\}} = H(\alpha).$$

The converse of this theorem is not necessarily true: there may be more than one α with the same value for $H(\alpha)$. However, among all α having the same value for $H(\alpha)$ an unique one can be chosen.

<u>Theorem 4.14:</u> If $H(\alpha) = H(\alpha) = A$, then $H(\alpha \cup \beta) = A$.

<u>Proof:</u> Let α and β be two arbitrary subsets of K,

$$p_{ij} \in \alpha \text{ implies } p_{ij} \cap H(\alpha) = \emptyset$$

whence

$$p_{ij} \subseteq \overline{H(\alpha)}$$

similarly,

$$p_{ij} \in \beta \text{ implies } p_{ij} \subseteq \overline{H(\beta)}$$

hence,

$$p_{ij} \in \alpha \cup \beta \text{ implies } p_{ij} \subseteq \overline{H(\alpha)} \cup \overline{H(\beta)}$$

or

$$p_{ij} \subseteq \overline{H(\alpha) \cap H(\beta)}$$

whence,

$$\cup \{p_{ij} | p_{ij} \in \alpha \cup \beta\} \subseteq \overline{H(\alpha) \cap H(\beta)}$$

or

$$H(\alpha \cup \beta) \supseteq H(\alpha) \cap H(\beta)$$

However, by Theorem 4.13

$$H(\alpha) \supseteq H(\alpha \cup \beta)$$

$$H(\beta) \supseteq H(\alpha \cup \beta)$$

so that

$$H(\alpha) \cap H(\beta) \supseteq H(\alpha \cup \beta)$$

which, with the previous inequality, shows

$$H(\alpha) \cap H(\beta) = H(\alpha \cup \beta)$$

If

$$H(\alpha) = H(\beta) = A$$

as given by the hypothesis

$$A = H(\alpha) = H(\alpha) \cap H(\beta) = H(\alpha \cup \beta)$$

Since the set of all subsets of K is finite, the set of all subsets α of K such that $H(\alpha) = A$ is a finite class of sets. If M(A) is the union of all subsets α of K such that $H(\alpha) = A$, then this M(A) is an unique set such that $H(M(A)) = A$. In the following theorem, discussion will be limited to those subsets of K which are M(A) for some simple concept A.

<u>Theorem 4.15:</u> If A and B are simple concepts then $A \subseteq B$ implies $M(B) \subseteq M(A)$.

<u>Proof:</u>

$$A \subseteq B$$

Hence,

$$A \cap B = A$$

Since,

$$A = H(M(A)) \text{ and } B = H(M(B))$$

$$H(M(A) \cup M(B)) = A \cap B = A$$

as indicated in proof of Theorem 4.14.

But by definition of the function M

$$H(M(A) \cup M(B)) = A \text{ implies } M(A) \cup M(B) \subseteq M(A)$$

whence,

$$M(B) \subseteq M(A)$$

Given any simple concept A, M(A) can be found effectively. It will be shown in the next chapter on concept learning that a rather straight-forward algorithm exists which can find the value of M for the smallest simple concept containing a

given set of exemplars. James Snediker has written a program based on Vindekilde-Gangkofner's work which learns superconcepts of concepts from examples and stores them as values of the M function.[34] For the purposes of this chapter, it will be assumed that the descriptions of simple concepts A are stored as the lists M(A). According to the last two theorems given two concepts A and B, one can find by merely comparing the lists M(A) and M(B), whether A is a subset of B or not.

If now one has the values of $M(A^S)$ and $M(\overline{A}^S)$ stored in a computer memory, one can deduce if a certain object X is a member of A in an approximate manner. If $M(X) \not\supseteq M(\overline{A}^S)$ then $X \not\subseteq \overline{A}^S$. However, X, being an object, is either contained in $\overline{A}^S$ or is disjoint from it, according to Theorem 4.1. Hence, $X \subseteq \overline{\overline{A}^S} \subseteq A$. If, on the other hand, $M(X) \not\supseteq M(A^S)$ then $X \not\subseteq A^S$. In this case, $X \subseteq \overline{A^S}$, and since $A \subseteq A^S$, $X \not\subseteq A$. If neither of these cases hold, then no conclusion can be drawn regarding the inclusion of X in A.

This approximate procedure is an analog of the Pennypacker recognition procedure described in the previous section. However, it is done by very simple programs based on very simple data structures.

If one can find simple methods for calculating the M-functions $\{\alpha\}$ whose corresponding simple concepts $\{H(\alpha)\}$ yield a given concept by union, then the above approximate recognition method can be improved quite easily. In that case,

the problem reduces once more to assuring oneself that the number of simple concepts needed to describe a concept be not excessively large - even though one already has the assurance that it would not be larger than the number of conjunctive concepts needed to describe a concept.

Before this last aspect of efficiency of description is discussed (and this will be discussed in a more generalized case in the next section) one must also point out that for the list comparison algorithm to be effective the description M for the concepts involved has to contain every property in the environment, not merely a fine structure family of properties. Hence, it is required to introduce some further algorithms for deducing property-values from the input properties as was done in the previous section. This would require the storing of certain property-values as described concepts. Efficient methods for this have not been developed yet.

6. A Generalized Description Language: Syntactic Axiomatisations

In this section the ideas introduced previously will be generalized and given a syntactic form similar to that of a formal logic. This will enable generalizations to description languages of greater flexibility and descriptive strength than are presently available. Several stages of development of such a language will then be exhibited and their use exemplified. Indications will be given later of how far these uses have been implemented on a computer.

It probably need not be established in any great detail to any discerning reader that what have been called conceptions and subsets of K in the previous sections are merely different ways of putting together atomic formulas to yield statement forms. These statement forms are characterized by the fact that they have only a single free variable. The component predicate letters are all unary (have only one argument) and give rise to predicates with one free variable. When these are put together to form statements, they all have the same free variable so that the resulting statement form has one free variable only.

What have been called objects in the previous sections are also examples of compound statement forms with a single free variable. These statements with a single free variable define sets of elements for which the statements are true. On the basis of this, tests have been described so far which test the

inclusion of one set in another. However, it has been tacitly implied that these tests will be used more often (and in the case of Henrypacker and the author's work, exclusively) when the included set is denoted by an object.

In the cases treated so far, where the environments have been finite, one could construct objects such that the name of every input property occured in it. As a result, the sets indicated in Theorem 4.1 could actually be denoted by a finite statement corresponding to an object. Since two elements in such a set cannot be distinguished by any concept, one might consider each object as denoting a single element. As a result, the objects themselves could be considered as the elements of the Universe. This point of view will be persued in the rest of this section even though some of the discussions of this paragraph are invalid in cases where the Universe is not finite.

When an object k is an element of the value p_{ij} of the property P_i, this fact can be expressed by a statement of the form $k \in p_{ij}$. This would necessitate that values of different properties to have different names. However, there are certain advantages to using the same symbols for the names of values of different properties. Some of these advantages were indicated in Section 3 of Chapter I. If such similarities of names are allowed, then a statement of the form $k \in p_{ij}$ becomes ambiguous since p_{ij} and $p_{i'j'}$ may be the same symbol. It is more advantageous to express the fact that the object k is an element of the value p_{ij} of property P_i by a statement of the form $P_i(k) = p_{ij}$.

In the past, names of concepts have been attached to the conceptions or the lists M(A). In effect, the descriptions have stood for statements S(x) with a single free variable x, and the names C of the concepts attached to the descriptions have indicated in effect that an object k is an element of the concept C if and only if the sentence S(k) is true; in effect, these have stood for statements of the form $x \varepsilon C \equiv S(x)$.

So far it has been assumed that names of properties, their values and concepts are symbols. However, it has already been indicated in Chapter I that various concepts can be given short descriptions if one allows the processing of the names of properties and values. It will, therefore, be of advantage if these are allowed to be objects, so that set theoretical processes can be carried out on them. This would allow the same programs that process objects to process the names also and a large amount of descriptive power would be obtained without vitiating the flexibility of the processing and without increasing unduly the size of the programs which do the processing.

One, of course, must take cognizance of the fact that the set theoretical processes discussed so far are merely capable of working on sets defined by unary predicates, while most of the time the processing of names that goes on in pattern recognition activities involves the calculation of functions and the ascertainment of relations. This, however, presents no problems, since functions and relations, being sets of

ordered n-tuples, may themselves be considered to be concepts in an Universe of ordered n-tuples. Also, n-tuples have the obvious n properties defined by each of their components. This fact will be made use of repeatedly in the examples that follow. It may be pointed out, of course, that this point of view indicates that some objects under discussion will be constructed out of property names which are entirely different from the property names used for constructing other objects. This may be looked upon as indicating the existence of a set of environments rather than a single one. Alternatively, one may consider that each object is constructed out of only a subset of properties. This certainly would perclude the objects from being unit sets. Since such a possibility has to be admitted in any infinite environment (and, as will be seen presently, this is a very natural requirement), one need not disregard this latter interpretation of an object. There is, moreover, a philosophic justification to it when one considers that the description of an "object" (in common parlance) often depends on the context. When one talks about a person, for instance, he may be talking about every appearence of the person on every occasion ("He expresses himself well") or of a specific appearence of a specific trait ("He was angry to-day"). When one says, "The letter X" he may be talking of a class of letters or a single mark on paper viewed from five directions or the same single mark seen in a certain illumination at a certain time.

In any case, the complete set theoretical interpretation of the syntactic structure has not been investigated yet. Especially in its fully developed form, discussed at the end of this section, this lack of interpretation raises several questions regarding the formal properties of the logic involved. For the present, attention will be directed towards the syntactic properties of the language only and its interpretation will be taken informally to be as motivated by the discussion above.

The major syntactic features of the language have already been discussed. Initially, one assumes a set of symbols, which will be taken to be countably infinite. To give syntactic meaning to such a set, they will be identified with words on a finite alphabet. The set of symbols, together with the special symbols (,),,,:,=,≤,,&,→,≡ and ~ and a specific string IN (standing for a single variable "Input") will construct all valid phrases of the language. The most obvious interpretations will be on the set of all "objects" as defined before. In what follows the basic definitions of the syntactic entities will be given and later exemplified by some examples to bring out the power of the language under some interpretations.

1. A finite sequence of lower case latin letters is a <u>symbol</u>.
2. A symbol is a <u>term</u>.
3. If α is a term and β is a term, then α,β is an

ordered pair. α is the left hand element of the ordered pair and β is the right hand element of the ordered pair $\alpha.\beta$.

4. An ordered pair is an ordered pair string; if α and β are ordered pair strings, then $\alpha:\beta$ is an ordered pair string.

5. If α is an ordered pair string then (α) is an object.

6. An object is a term.

An example of the syntactic appearence of an object may be worth giving here

(name, harry; house, (number, five; street, luther))

denotes (under interpretation) a person called harry whose house is distinguished from others by an address. The value of the property "house" itself is an object here.

7. The string IN is a term.

8. If α is a term and β is a term then $\alpha(\beta)$ is a term.

9. If α is a term and β is a term, then $(\alpha=\beta)$ is a statement.

10. If α is a statement and β is a statement, then $\sim\alpha$, $(\alpha\vee\beta)$, $(\alpha\&\beta)$, $(\alpha\rightarrow\beta)$ are statements.

11. If α is a term and β is a symbol, then $(\alpha\epsilon\beta)$ is a statement.

12 If α is a symbol and β is a statement, then $IN\epsilon\alpha\equiv\beta$ is a concept. α is the name of the

concept and β the <u>intention</u> of the concept.

It is worth remarking at this point that the syntactic entity "concept," as defined here is at variance with the meaning attached to the word in previous discussions. In the parlance of the previous discussion, a concept would be the set of all elements of the Universe which satisfies the intention of a syntactic entity, "concept." The intention is what has previously been alluded to as the "description" of the concept.

Among the set of statements defined above a subset will now be defined to be the set of "true statements." For this some auxilliary definitions are needed. This involves a mapping (called "value") from a subset of the set of all terms and ordered pair strings into the set of all terms and ordered pair strings, defined as follows.

13. The value of a symbol is itself.

14. The value of IN is not defined.

15. The value of a term of the form α(β) is defined if and only if the values of α and β are defined, the value of β is an object and the value of α is the left hand element of some unique ordered pair in the value of β. In this case, the value of α(β) is the right hand element of the ordered pair of which the value of α is the left hand element.

As examples, the value of color((shape, square; color, blue)) is blue while the values of color((color, red;

color, blue)) and color((shape, square; size, big)) is undefined.

16. The value of an ordered pair α,β is defined if and only if the values of α and β are defined. In that case its value is α',β' where α' and β' are the values of α and β respectively.

17. The value of an ordered pair string $\alpha;\beta$ is defined if and only if the values of α and β are defined. In that case its value is $\alpha';\beta'$ where α' and β' are the values of α and β respectively.

18. The value of (α) is defined if and only if the value of α is defined. In that case the value of (α) is (α') where α' is the value of α.

19. An object is an <u>exemplar</u> if its value is itself.

20. Two symbols are <u>identical</u> if they constitute the same string of characters.

21. Two ordered pairs are <u>identical</u> if their left hand elements and their right hand elements are identical.

22. Two exemplars are <u>identical</u> if each ordered pair of one is identical to some ordered pair of the other and vice versa.

23. A statement $(\alpha=\beta)$ is <u>true</u> if and only if the values of α and β are defined and the value of α is identical to the value of β.

24. Given a set D of concepts a statement is <u>D-true</u> if and only if it is true or if the statement is of the form $(\alpha\varepsilon\beta)$, β is the name of some concept K in D and the statement obtained by replacing every occurence of IN in the intention of K by α, a D-true statement results.

25. $(\alpha\vee\beta)$ is <u>D-true</u> if and only if either α or β is D-true; $(\alpha\&\beta)$ is <u>D-true</u> if and only if both α and β is D-true. $(\alpha\rightarrow\beta)$ is <u>D-true</u> if and only if $(\sim\alpha\vee\beta)$ is D-true; $\sim\alpha$ is <u>D-true</u> if and only if α is not true and does not contain the term IN.

It may be worthwhile at this point to exemplify the utility of this system in terms of the example used at the end of Section 3 of Chapter I. Let D consist of the single concept

IN$\varepsilon\alpha$≡((head(borders(IN)) = t) & ((head(crosses(IN)) = f)
∨((tail(borders(IN)) = t) & (tail(crosses(IN)) = f))))).

Then the statement ((crosses,(head,f;tail,t);borders(head,t,tail, f))$\varepsilon\alpha$) is a D-true statement. This can be seen as follows. Since the statement is of the form $\alpha\varepsilon\beta$, one obtains by definition 24 above that the statement is true if and only if the statement obtained by replacing all occurences of IN in the statement to the right of ≡ in the concept above by (crosses,(head,f;tail,t); borders,(head,t;tail,f)) is D-true. This statement is of the form $\alpha\&\beta$. The right hand conjunct of this statement is (head(borders((crosses,(head,f;tail,t);borders,(head,t;tail,f)))) = t). By definition 23 this is true if the values of the terms

on the left and right of the = sign be identical. The value of the term t is t by definition 13. The value of the left hand term is defined by rule 15 to be the value of head((head, t;tail;f)) which is t again. So one of the conjuncts of the intention of the concept named "a" is true. The other conjunct is of the form $\alpha \vee \beta$ and is true by rule 25 if either of the two disjuncts is true. The first disjunct is

(head(crosses((crosses,(head,f;tail,t);
borders,(head,t;tail,f)))) = f)

which is again true by definitions 23 and 15 and 13. One disjunct being true, the statement is true.

Before exhibiting by some more examples the extent of the power of the language, it is worthwhile to point out that any statement in this language which does not contain IN can be tested for truth by carrying out an algorithm on the statement which is closely related to the definitions 1-25 above. This algorithm is shown in Figure 4.3 in flow chart form.

In indicating the flow chart it has been assumed that the tests indicated in the control boxes of the flow chart can indeed be performed. The assumption can be justified on the basis of what is known about syntax-directed parsing to-day.[35] The point will not be belaboured here.

Since the programs are recursive, some of the variables used are actually stacks. To distinguish them from other variables, their names have been written in upper case

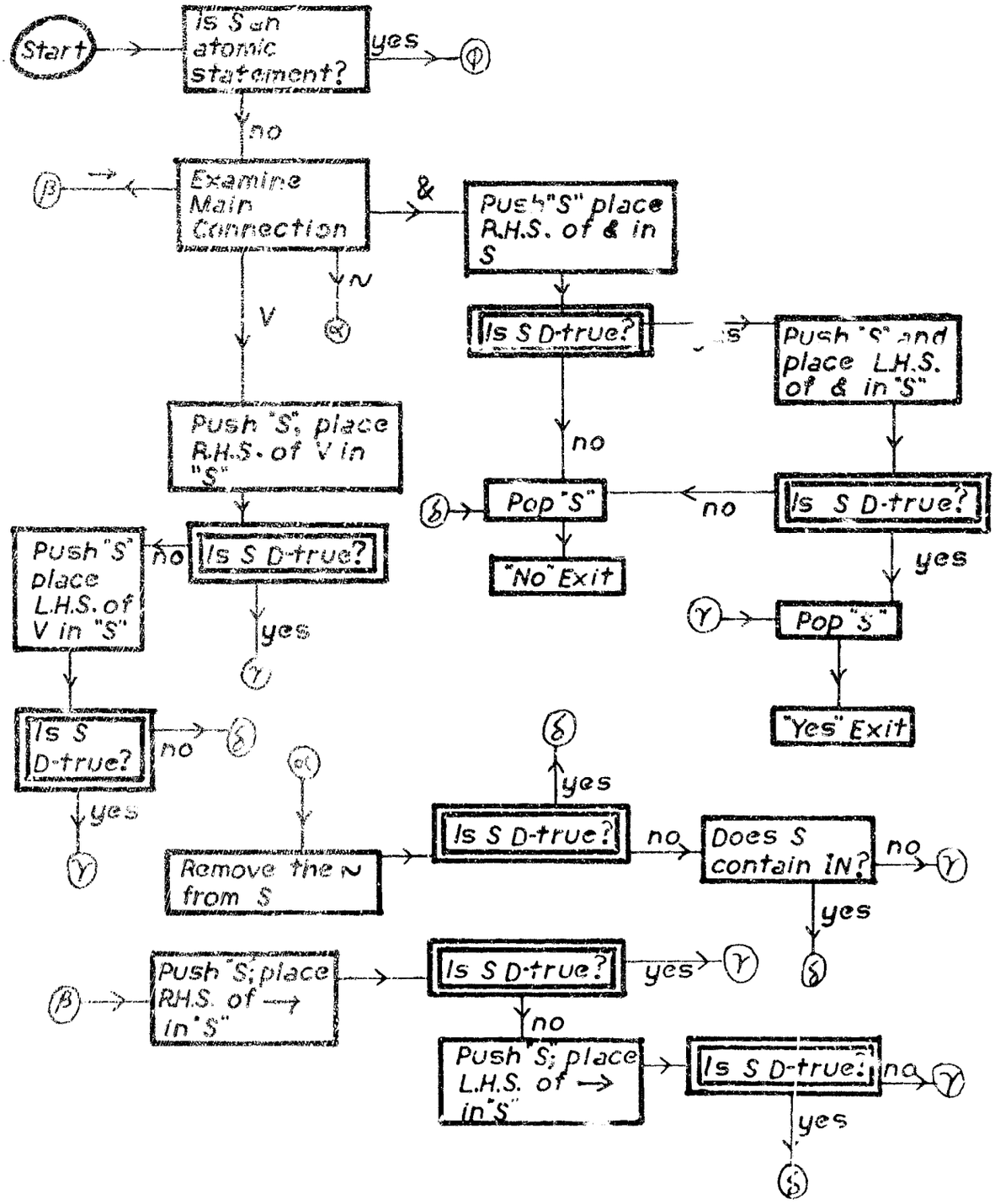


Figure 4.3 (a)
(part 1)

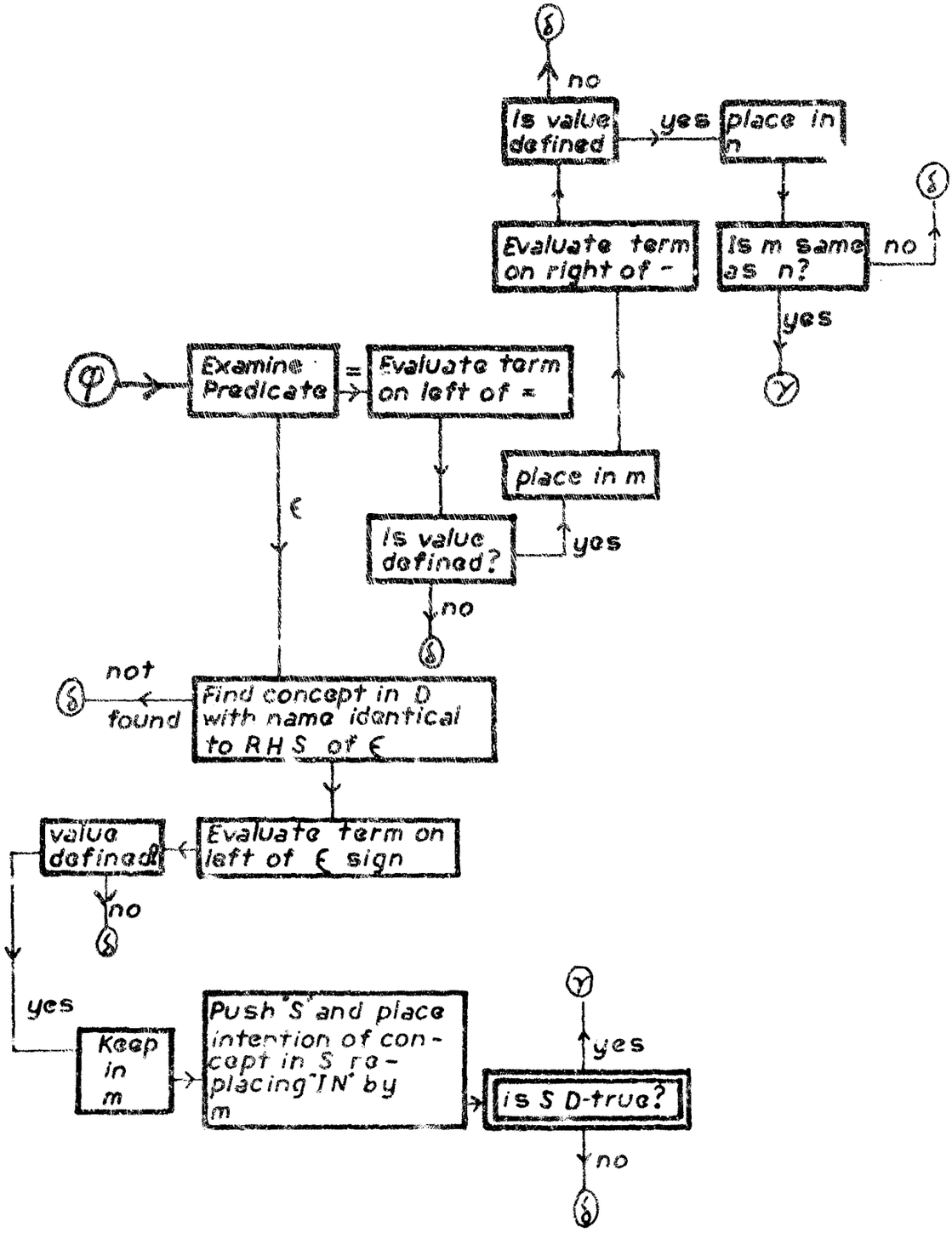


Figure 4.3 (a)
(part 2)

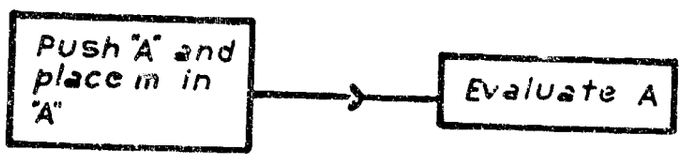


Figure 4.3 (b)

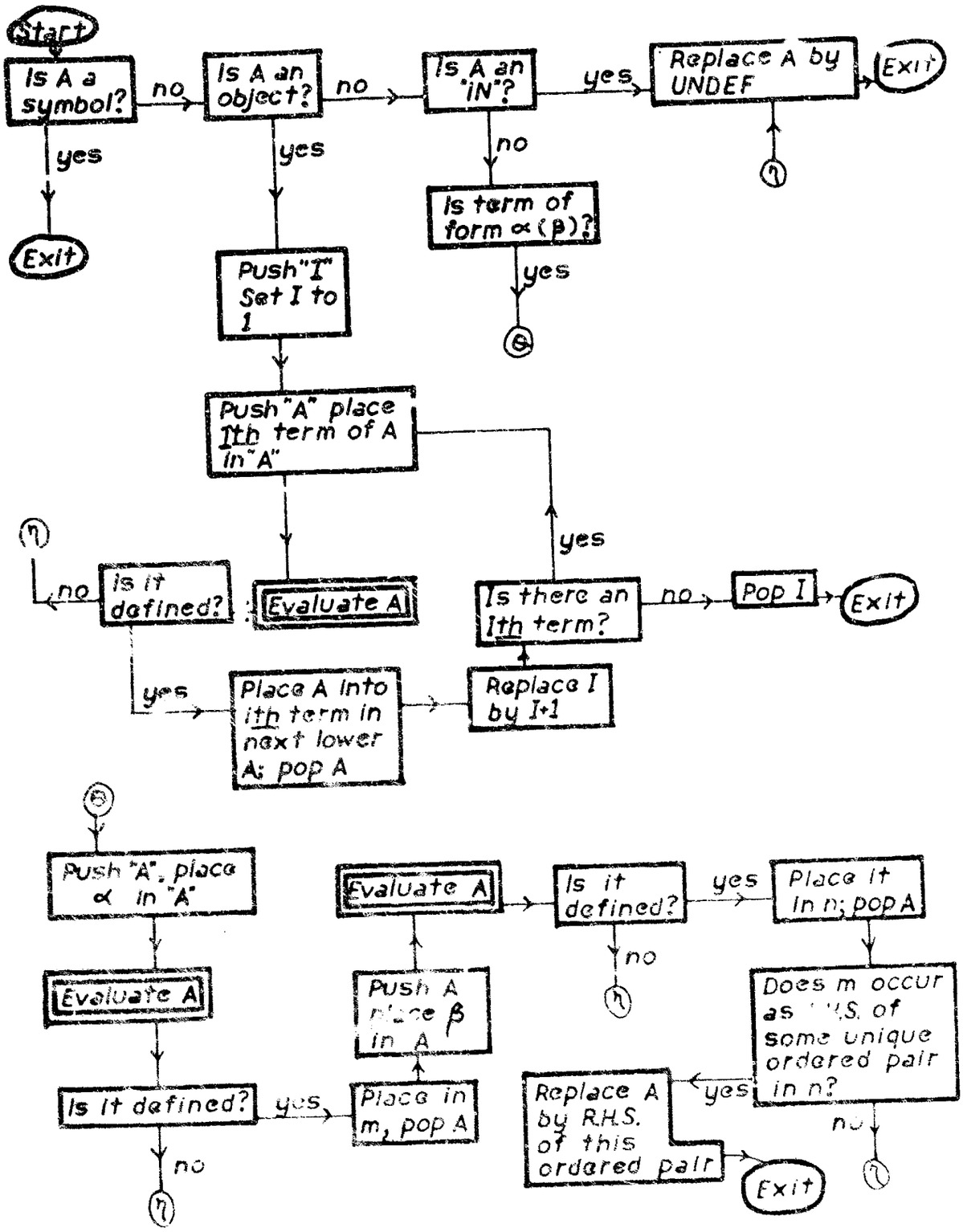

Start
Is A a symbol?
no
yes
Exit
Is A an object?
no
yes
Push "I" Set I to 1
Push "A" place Ith term of A in "A"
Evaluate A
Is it defined?
no
yes
Place A into ith term in next lower A; pop A
Replace I by I+1
Is there an Ith term?
yes
no
Pop I
Exit
Is A an "IN"?
yes
Replace A by UNDEF
Exit
no
Is term of form α(β)?
yes
Push "A", place α in "A"
Evaluate A
Is it defined?
yes
no
Push A place β in A
Evaluate A
Is it defined?
yes
Place in m, pop A
no
Is it defined?
yes
Place it in n; pop A
no
Does m occur as L.H.S. of some unique ordered pair in n?
yes
no
Replace A by R.H.S. of this ordered pair
Exit

Figure 4.3 (c)

letters while other variables are named in lower case letters. As before names are written in quotes and their content without quotes.

It might have been useful at this point to include a proof that the algorithm exhibited in Figure 4.3 does terminate for every statement not containing IN. However, the proof is simple and the reader may be left to convince himself of the fact. However, the fact that the algorithm does terminate leads to the useful fact that all statements in this language can be recognized as true or false. The language is "complete" in that sense. It is also "consistent" in the sense that all statements are not true. Also, it is "decidable" in the sense that given a statement it can be decided with a finite number of operations whether it is true or not. However, these assertions have very little significance in the mathematical sense, since what has been described above is not, in a strict sense, a logical theory. Later on in this section, in the interest of greater strength, the language will be extended into a logic. Meanwhile, the following examples will show that the language, even in its present form, has considerable strength.

The first example indicates how representations of integers and operations on integers can be described within the machinery of the language described so far. It will be considered that the integers are expressed by binary numerals. Each numeral will be considered to have two properties, "head"

and "tail." The values of "tail" are " and "1," and stand for the least significant digit of the numeral. The values of "head" are either "null" or an integer, representing the more significant digits of the numeral. To make sure that confusions do not result from leading zeros, they will be disallowed in the description. In what follows a set of concepts will be introduced which will define positive integers, the relation of natural ordering among integers and sums of integers. From these, the reader will convince himself, the arithmetic of positive integers can be defined. Zero and negative integers can also be defined with some work.

```
INεdigit≡((IN = 0) ∨ (IN = 1))
INεlessd≡((first(IN) = 1) & (second(IN) = 0))
INεnumer≡(((head(IN) = null) & (tail(IN) = 1)) ∨ ((head(IN)
        ε numer) & (tail(IN) ε digit)))
 INεless≡(((head(first(IN)) = null) & ~(head(second(IN))
        = null))) ∨ ((first,head(first(IN));second,head
        (second(IN))) ε less)
        ∨ ((head(first(IN)) = head(second(IN)))
        & ((first,tail(first(IN));second,tail(second(IN)))
        ε lessd))
 INεsumd≡((second(IN) = 0) & (first(IN) = third(IN)))
        ∨ ((second(IN) = 1) & ~(first(IN) = third(IN)))
INεcarry≡((first(IN) = 1) & (second(IN) = 1))
  INεsum≡((first,tail(first(IN));second,tail(second(IN));
        third,tail(third(IN))) ε sumd) & ((~((first,tail
```

```
            (first(IN));second,tail(second(IN))) ε carry) &
            ((first,head(first(IN));second,head(second(IN));
            third,head(third(IN))) ε sum))
            ∨ (((first,tail(first(IN));second,tail(second(IN)))
            ε carry) & ((first,head(first(IN));second,head
            (second(IN))·third,head(third(IN)) ε ripple)))
            ∨ ((first(IN) = null) & (second(IN) = third(IN)))
            ∨ ((second(IN) = null) & (first(IN) = third(IN))))
IN∈ripplecar≡((first(IN) = 1) ∨ (second(IN) = 1))
   IN∈ripple≡(~((first,tail(first(IN));second,tail(second(IN));
            third,tail(third(IN))) ε sum) & ((~((first,tail
            (first(IN));second,tail(second(IN))) ε ripplecar)
            & ((first,head(first(IN));second,head(second(IN));
            third,head(third(IN))) ε sum)) ∨ (((first,tail(first
            (IN));second,tail(second(IN))) ε ripplecar) &
            ((first,head(first(IN));second,head(second(IN));
            third,head(third(IN))) ε ripple)))) ∨ ((first(IN)
            = null) & ((first,second(IN);second,(head,null;
            tail,1);third,third(IN)) ε sum)) ∨ ((second(IN)
            = null) & ((first,first(IN);second,(head,null;
            tail,1);third,third(IN)) ε sum))
```

Some explanation is probably necessary for the last three concepts. The elements of "sum" are ordered triples of numerals such that the third is the sum of the first two in the usual sense. The Universe of Triples has three properties "first," "second" and "third." The description of "sum"

essentially says, "The tail of the third is the sum of the tails of the first and second. If there is no carry then the head of the third is the sum of the heads of the first and second. If there is a carry, then the heads of the first, second and third are related by ripple." The description of ripple is the same as the description of sum, except for the addition of a bit in the least significant digit which is allowed to "ripple through."

An example, representing the binary sum 1 + 11 = 100, will probably clarify matters further. The element of sum of concern here is

(first,(head,null;tail,1);second,(head,(head,null;tail,1);
tail,1);third,(head(head(head,null tail,1);tail,0);tail 0))

Initially, tail(first(IN)) = 1;tail(second(IN) = 1; and tail(third(IN)) = 0, satisfying the first conjunct of the first disjunct in the intention of the concept named "sum." Hence, since (first,1;second,1) is an element of "carry" the object (first,null;second,(head,null,tail,1);third,(head,(head, null;tail,1);tail,0)) is to be a member of "ripple" by the second disjunct of the second conjunct of the first disjunct in intention of "sum." Since "first(IN)" for this new object is "null" the third disjunct of "ripple" has to be satisfied, that is the object (first,(head,null;tail,1);second(head,null; tail,1);third,(head(head,null;tail,1);tail,0)) has to belong to "sum." Again, the tails satisfy the first conjunct in the first disjunct. Also, there is a carry so that the object

(first,null;second;null;third,(head,null;tail,1)) must be a member of ripple. Hence, again by the third disjunct of ripple (first,null;second,(head,null;tail,1);third(head,null; tail,1)) must belong to sum. By third disjunct of sum one must have ((head,null;tail,1) = (head,null;tail,1)) which is true.

One might object to the rather cumbersome nature of the concepts. However, any statement describing a complicated operation like arithmetic sum is bound to be somewhat cumbersome. The present statements are certainly less cumbersome than, say the Boolean expression describing a parallel thirty-six bit adder and yet is expressing operations on strings of arbitrary length.

However, the expression (head,(head,(head,null;tail,1); tail,1);tail,0) is certainly a more cumbersome expression than 100 or even (x = 1) & (y = 0) & (z = 0). Later on in this section methods will be considered which will reduce the unwieldyness of objects in the language and will also enable the attachment of names to objects. This way, it will be easier to express operations by means other then through relations.

The importance of the last sentence above becomes clear when one wants to express facts like 1+1+1=11 and 11+101=100+100. Unless some concept other than sum is to be introduced anew (a wasteful procedure), one has to introduce existential and universal quantifiers into the language so the above facts can be expressed respectively by saying "for any

z such that 1+1 = z it is true that z+1 = 11" and "for any z such that 11+101 = z it is true that 100+100 = z." This, of course, renders the recognition process of Figure 4.3 inadequate. Before these facts are discussed, one more example will be given which will bring out some further strengths and weaknesses of the language.

One can imagine classifying the residents of a street by their name, their house of residence, their age range (small, big) and sex. The house of residence may be described by their size, color and level of beauty (and perhaps even number, which would render the environment for houses non-full, which it is anyway). A typical person might be an object like (name, lucy; age, small; house, (size, small; color, white; look, pretty); sex, girl)

In such an Universe, a relation like fatherhood can be expressed as follows

IN∈father≡((house(first(IN)) = house(second(IN))) & (age(first (IN)) = big) & (sex(first(IN)) = man))

that is "of two people in the same house, the adult male is the other ones father." The description is certainly incomplete, but can be improved upon.

The difficulty in the way here again is ones inability to make such simple statements as "Harry is Susan's father." One can try to get around this by including the father's name in the object, but then one has to make a decision on whether to include the father's name only or to include the entire

object describing the father. The second gives rise to an infinite recursion: the first leads to the obvious problem of finding the father's father. Any cross-indexing needs the attachment of names to objects as are needed in the case of numerals.

These and related difficulties can be resolved (as far as the descriptive strength of the language is concerned) by introducing variables other than IN into the language and allowing logical quantifiers. Also, capabilities have to be established for naming objects by strings of symbols. However, strings of symbols, unlike symbols, should be processable. To enable this, a new syntactic entity will be introduced. In this new notation, the object (head,(head(head,null;tail,1); tail,0);tail,0) could have the representative STRING (1,0,0,numer) and if (first,α;second,β) was an element of "fatherhood," then α could be represented by STRING(father, of,β). Such naming processes, of course, should be describable within the language. Also, one should have the freedom of introducing axioms in D other than concepts. To do this, the symbol ≡, which so far had no logical significance, has to be a part of the theory. The concept of "proof" has to be introduced as in any logic. This renders recognition of objects as belonging to concepts more difficult. However, Milliken has shown that a suitable modification of the algorithm shown in Figure 4.3 can be made which enables recognition of some concepts even in this extended language.[43] Since the extended

language enables its own description and can describe integers, it is clear that a mechanical recognition procedure for all objects is impossible.[44]

In what follows, the extended language will be introduced and exemplified. A large part of the language will be similar to the one discussed before.

a) The Syntax

1. Any string of lower case latin letters and arabic numerals is a symbol. A symbol is a term. Any string of greek letters is a variable. A variable is a term.

2. If A and B are terms, then A,B is an ordered pair. An ordered pair is an ordered pair string. If A and B are ordered pair strings, then A;B is an ordered pair string. If A is an ordered pair string, then (A) is an object. An object is a term.

The important thing added to the syntax at this point is the variable. The discerning reader probably noticed before this that IN was playing a role similar to a variable in the previous discussion. However, a larger repertoire of variables are necessary for full flexibility of use. Going on with the syntax:

3. If A is a term and B is a term, then A(B) is a term.

The term

color(a)

stands for the English phrase "the color of a." Generally such a term is meaningful only when a stands for some object like

(color, red; size, big; number, 135);

in this case color(a) would stand for

red.

However, this interpretation, unlike the previous case, is part of our axiom set now.

4. If A and B are terms, then (A=B) and (A∈B) are <u>statements</u>. If A and B are statements, then (A∨B), (A&B), (A→B), (A≡B) and ~A are <u>statements</u>.

5. If A is a statement and B is a variable, then (∀B)A and (∃B)A are <u>statements</u>.

Rule 5, above, is one of the major reasons for introducing variables as parts of the syntax. Also, the use of ≡ now has a logical interpretation as a propositional connective, rather than merely as a cue for recognition as it had been in the previous discussion.

It has already been pointed out before that in this description language one has the freedom of giving names to sets of objects and using these names to define new sets of objects. However, these names are arbitrarily given symbols and did not have any syntactic relationship to the set of objects being defined. Hence, if one had to define a class of sets which had similar structures, this similarity would not be reflected in the given names. Thus, the set of all numbers

greater than 3 and the set of all numbers greater than 50 would be given two different names and the fact that each set has a lower bound would be lost. The reader is to recall that calling them things like "greater than 3" does not help, since the language deals with symbols as a single entity.

A part of what follows is directed towards giving a number of string processing abilities to any automaton using the language and for using these abilities to tie in the names of sets and objects with their structure. However, in line with previous procedures, the mapping which define the process will be included only in the axioms of the system. What follows then is only the syntactical part.

6. A symbol is a <u>train</u>. If A and B are trains, then A,B is a <u>train</u>. If A is a train, then STRING(A) is a <u>string</u>. A string is a <u>train</u>.
7. A term is an <u>operand</u> If A and B are operands, then A,B is an <u>operand</u>. If A is an operand, then TIE(A) is a <u>representative</u>. A string is a <u>representative</u>. A representative is an <u>operand</u>.
8. EMPTY is an <u>operand</u>. If A is a representative, then STRIP(A), END(A) and REST(A) are <u>operands</u>.
9. If A is a representative, then REPINV(A) is a <u>term</u>. If A is a term, then REP(A) is a <u>representative</u>.

Examples of strings and representatives are

STRING(2,0,1 num)

STRING(STRING(1,0,num), STRING(2,1,num),sum)

TIE(STRIP(REP(α)), end(β))

TIE(brother,of,name(first(α)))

10. If A and B are representatives, then (A=B) is a <u>statement</u>.

The following examples will indicate to the reader the usefullness of strings in obviating the difficulties mentioned earlier. Although the concept of "truth" has not been introduced in this formalism, the reader should be able to follow the examples from an intuitive understanding of the meaning of truth.

Let there be a concept in D as given before

αεfather⇒(age(first(α)) = big) & (sex(first(α))) = man)

& (house(first(α)) = house(second(α)))

A typical object in "father" might be

(first,(name,frank;sex,man;house,(size,small;

color,blue;look,pretty);age,big);second,(name

susan;sex,girl;house,(size,small;color,blue;

look,pretty);age,small)).

One can call Frank "Susan's father" - a rather generalized naming operation which was impossible in the language so far. For this, one can now define an axiom as follows

αεfather⇒(REP(first(α)) = TIE(father,of,name(second(α)))

which, by the rules described later, would yield the statement

(REP(name,frank;sex,man;house of residence

(size,small;color,blue;look,pretty);age,big)) =

STRING(father,of,susan)

The exact way in which the truth of this statement is derived in the language will be shown after the axiom system has been discussed.

b) The axiom schemata

1. A statement (A=B) is an <u>axiom</u> if and only if

(i) A and B are each the same (identical) term. Two objects are identical if every ordered pair appearing in A is identical to some ordered pair appearing in B and vice versa. Two ordered pairs are identical if their first elements are identical and their second elements are identical.

(ii) A and B are identical trains.

(iii) If A is a term of the form C(D) where D is an object, C the first element of some unique ordered pair of D and B is the second element of the same ordered pair.

(iv) If A is of the form TIE(C) or TIE(C,EMPTY) or TIE(EMPTY,C), C is a train and B is the string STRING(C).

(v) If A is of the form END(C) where C is a string of the form STRING(D,B) (alternatively STRING(D,A)) and B is either a symbol or a string.

(vi) If A is of the form REST(C) and C is a string of the form STRING(B,D) where D is a symbol or a string, or if C is a symbol and A is EMPTY"

(vii) If A is of the form STRIP(STRING(B)).

2. Every statement ~(A=B) is an <u>axiom</u> if and only if

(i) If A and B are symbols but not identical or if A (alternatively B) is a symbol and B (alternatively A) is an object.

(ii) If A and B are both objects which do not contain terms of the form C(D) and A and B are not identical.

(iii) If A and B are both trains and not identical.

3. Every statement ~(A∈B) is an <u>axiom</u> if B is an object.

4. If A and B are statements and X is a variable then the following are <u>axioms</u>

(i) $(A \to (B \to A))$

(ii) $((A \to (B \to C)) \to ((A \to B) \to (A \to C)))$

(iii) $((\sim A \to \sim B) \to (B \to A))$

(iv) $((\forall X)(A \to B) \to (A \to (\forall X)B))$

(v) $((\forall X)A \to S^{X}_{Y}A)$

In (iv) X must not occur free in A. In (v) Y is either a variable or a term; however, no free occurence of X in A must be in a sub-statement B of A in the form (∀Y)C where X is a free variable in C.

An occurence of a variable X in a statement A is said to be bound if it occurs in some sub-statement of A in the form (∀X)C. An occurence which is not bound is called free.

The symbol $S^{X}_{Y}A/$ stands for the statement A' obtained by replacing all free occurences of X in A by Y.

5. If A and B are statements and X is a variable then the following are axioms

(i) ((∃X)A ≡ ¬(∀X)¬A)

(ii) ((A∨B) ≡ ((A→B)→B))

(iii) ((A&B) ≡ ¬(A∨¬B))

(iv) ((A≡B) ≡ ((A→B) & (B→A)))

It will be noticed that some statements which were true in the previous system are now axioms. For instance, (color((size,big;color,red)) = red) is an axiom. However, since the recursive function "value" is not defined in the new system, some other statements, like (color(first((first,(color,red;size,big);second,hand))) = red) is not an axiom, because first(first(color red;size,big);second,hand) is not an object but a general term which is not covered by rule (bliii) above. The truth of the above statement will only follow from the definition of the rules of inference as given below. Meanwhile, it may be worthwhile to point out that although the statement above is not an axiom, its negation is not one either. Truth and falsity of statements are much more difficult to test in the new system: a price one pays for flexibility.

c) Rules of inference

Given a set D of statements and a statement A, we say A is derivable from D if there exists a sequence $S_1, S_2, \ldots, S_n$ of statements such that S_n is the same as A and for every i ($1 \leq i \leq n$) either S_i is an axiom, or S_i is a member of D or is inferred from previous statements $S_j, S_k (j,k < i)$ by one of the following 5 rules of inference:

(i) From A to infer $(\forall X)A$ where X is a variable.

(ii) From A and $(A \cdot B)$ to infer B.

(iii) From A and (X=Y), to infer A' where A' is obtained from A by replacing some occurences of X by Y and some occurences of Y by X.

(iv) From(REP(X)=Y) (where X is a term and Y is a representative), to infer (REPINV(Y)=X).

(v) From (A=B) to infer (B=A).

It may be worthwhile at this point to point out how some of the previously discussed statements are derivable from certain axioms.

The statement

(color(first((first,(color,red:size,big):second,hand))) = red)

is derivable ("true") since

(first((first,(color,red:size,big):second,hand)) = (color,red:size,big))

is an axiom by (biii) above.

Also, (color(color,red:size,big) = red) is an axiom.

Since both of the above are derivable, one can replace (accordind to rule (ciii) above) "(color,red;size,big)" in the second by "first{(first,(color,red;size,big);second,hand})," the L.H.S. of the first statement; deriving the initial statement.

Again, one can derive REP(name,frank;sex,man;house of residence,(size,small;color,blue;look,pretty);age,big) = STRING(father,of,susan) from the concept named "father" and the statement

(αεfather)→(REP(first(α)) = TIE(father,of,
name(second(α))))

in a similar manner, in view of some of the axioms discussed earlier.

Before closing this section it may be worthwhile to point out how the REP(representation) of the object named "frank" does not have to be unique. If locations of houses in cities and professions of people were included in the Universe, this object might also have STRING(frank,the,barber, of,seville) as a representation and one could have statements like

REPINV(STRING(father,of,susan))
=REPINV(STRING(frank,the,barber,of,seville)).

An example from the field of character recognition may motivate some readers more. Assume that the Universe consists of the different configurations of excitations on a square array of photo-cells and suppose one is interested in all configurations in which all excited photo-cells lie on a

straight line inclined to the horizontal edge of the photo-cell at -45°. Call it "negativediag."

In this Universe each photo-cell determines a property whose values are called 0 and 1. Each photo-cell is determined by its coordinates. Hence, the Universe of photo-cells has two properties corresponding to the X and Y coordinates, which will be called "first" and "second" here. The values of both these properties are integers, which have been discussed before in connection with the description language. The reader will verify that a typical configuration on a 2x2 array may be denoted by the object

```
((first,(head,null;tail,1);second,(head,null;
tail,1)),1;(first,(head,(head,null;tail,1);
tail,0);second,(head,null;tail,1)),0;(first,
(head,null;tail,1);second,(head,(head,null;
tail,1);tail,0)),1;(first,(head,(head,null;
tail,1);tail,0);second,(head,(head,null,tail,
1);tail,0)),0)
```

representing the configuration

1	0
1	0

One can now write a statement which defines the set "negativediag."

$$\alpha \epsilon \text{negativediag} \equiv (\exists\beta)(\exists\gamma)((\beta(\alpha) = 1) \;\&\; (\gamma(\alpha) = 1) \;\&\; \sim(\beta=\gamma))$$
$$\&\; (\exists\beta)(\forall\gamma)((\gamma(\alpha) = 1) \rightarrow ((\text{first},\text{first}(\gamma);$$
$$\text{second},\text{second}(\gamma);\text{third},\beta) \;\epsilon\; \text{sum}))$$

The language while describing concepts, has usefulness in other information retrieval systems. Its use in such systems has not been investigated but it may be safe to say that its capability, even if somewhat curtailed, may be greater than any conjunctive system of descriptors or association strength networks discussed in the field.[45]

7. Other Description Languages

Set theoretical descriptions have been used for concepts mostly by workers interested in simulating human cognitive activity. However, the entire basis of Pattern Recognition as a phenomenon is set theoretical or more precisely, logical. The motivation behind the different methods used in synthesizing concept learning algorithms often lie in fields like statistics,[36] or linear algebra[37]; however, in every case the final algorithm for recognizing an object as belonging to a concept or pattern (after the "learning phase") can be looked upon as using a compound statement as the description of the concept. This will be clear if one considers the case of a set of binary vectors whose components satisfy a linear inequality. The set {000,001,011,100,101,111} of binary vectors, for instance, can be represented by the linear inequality $\frac{1}{2} - y + z > 0$ or the Boolean Expression $(\sim y + z)$ or the statement $(y = 0) \vee (z = 1)$. Similar statements can be constructed for cases where the discriminating functions are non-linear or even when the components of the vectors come from a continuam. Each component of the vectors are properties whose values isolate subsets of the Universe. However, in this latter case [illegible] logical expressions representing these functions need quantifiers to take care of infinite set-theoretic connectives.

The modes of combination available to Pattern Recognition schemes based on statistics or linear algebra are richer than those available to Boolean Algebra. However, very

often the effectiveness of the various modes of combination dealt with in literature are strongly dependent on the initial measurements (i.e., the input properties or "features") - and dependent in an extremely ill-understood way; also, there is no uniform method for changing one set of algebraic operations into another to yield new "features" from old ones. It is to achieve such flexibility and to tie down the description with the basic set theoretical structure of the problem that the language of Section 5 was developed.

Very little can be said regarding the ultimate effectiveness of the various algebraic or statistically oriented languages available for description of patterns. Some of them (like linear separation) essentially restrict the capability of description for the sake of simplicity of description and "training." Others, like Braverman's potential functions,[38] are essentially "open ended" and can be used (like Boolean functions) to describe any concept whatever. However, these latter lead to problems of confidence limits and "generalization." It may be well to defer discussions of these to the next chapter, when learning is discussed.

Returning to the discussion of the use of simple Boolean expressions (or expressions in Propositional Calculus) as a description language, it has been shown in Section 5 how the class of describable concepts can be restricted for parsimony, yielding, say, the class of conjunctive and simple concepts. Although the class of simple concepts properly contains the

class of conjunctive concepts, all concepts are not simple and modes of description have to be available for describing every concept. The language of the property lists is not adequate for this. A suitable extension of it has been suggested which is capable of describing any concept. Conceptions can describe any concept also. The efficiency of both are severly restricted for a large class of concepts. However, the ultimate capability for description is not limited, as they are for perception-like devices, which use hyperplanes as discriminating surfaces.

Two other languages, The CLS by Hunt,[39] and EPAM by Feigenbaum,[40] are restricted in their ability to the same extent as the Conceptions. The relationship between the two have been discussed by Hunt. It is relevant to discuss here the salient points of difference between the Conception on the one hand and the CLS and the EPAM on the other.

The fact that the systems used by Hunt and Feigenbaum are binary trees while the Conception allows more than two branches to emenate from the nodes is not a crucial difference. All three are essentially tree structures - the fact that only one of the trees is non-binary is easily attributable to the strong influence that the word "bit" has had on psychologists since 1948.

There are two crucial differences between the CLS and EPAM tree and the Conception, however. One lies in the fact that the name of the concept described by the tree is placed at the root of the tree in the Conception while it is

placed in the leaves in the other two languages. This looks like an essentially wasteful feature of the Conception, since there has to be a different tree for every concept. However, there are some essential reasons for doing this, as an analysis of the basic sets involved will show.

For one thing, a property which is relevant to a concept A may not be relevant to a concept B. This fact can be used advantageously in the Conception. But when the EPAM processor, say, is testing an object for membership of B, it still has to go through a test for this non-relevant property, just because it was worth testing in testing for A.

There is another, much stronger reason for attaching concept names to roots of trees. Very often the same concept turns out to be sub-concepts of two different concepts, in the sense that there is a concept C, two property values $p \in P$ and $q \in Q$ and two concepts A and B such that $C = p \cap A = q \cap B$. Then, the name C can be placed on the conceptions of A and B instead of placing the entire C tree twice in the conceptions of A and B as would be necessary in the CLS or the EPAM. (These remarks do not pertain to CEPAM developed by Ernst and Sherman, which will be discussed later.)

These differences occur essentially since it is overlooked in the other languages that a specific object can be a member of more than one concept - hence, one has to attach more than one name to every leaf if the name of the concepts are to be attached to leaves. Also, some of the intermediate nodes of

a tree may contain enough information to identify an object in a concept while to recognize the same object in one of its sub-concepts would necessitate going deeper into the tree.

The need for attaching concept names to nodes becomes even more clear when one needs to define a new test in terms of old tests in the interest of simplicity of description. In the Conception, it is a matter of adding a new list into the Conception of the Universe - while it is impossible in the other two structures. No flexible language has appeared in the Pattern Recognition or cognitive process research as a counterpart of the description language discussed in Section 6. Various languages, like the languages developed by Narasimham,[47] and by Kirsch,[48] carries out operations on names of properties, but the procedures do not have the same flexibility and uniformity. However, the SIR System of Raphael has certain similarities with the language of Section 6 which ought to be discussed.[46]

One of the greatest similarities between SIR and the language described in Section 6 are the similarities in the structure of objects. The same property-value-pairs are stringed together, and values of properties may be objects themselves; it is not clear whether names of properties can be objects in SIR.

In SIR, a list of symbols are also allowed to names of values of properties. Thus, (name,harry;brothers,(tom,dick,don);age,15) would be a valid object. The advantage of this,

of course, is that the object, in a sense, has greater efficiency of processing. For instance, let the question be asked "Is Tom Harry's brother?" (i.e., let tom e brother (harry) be posed as a theorem). A special processor could answer this as "yes." However, if the question is asked "Does Harry have a brother aged 20?", the processor will have to know that there are objects whose property "name" have values "tom," "dick" and "don": a fact which is not clear from the format and a separate processor would be needed to incorporate such extra assumptions.

In fact, SIR as originally conceived and implemented consisted mostly as a series of processors capable of handling a special class of objects. The syntactic restrictions the objects were to satisfy were defined more in terms of the structures of the processors. As a result, certain facts about objects were easy to describe while others were impossible, unlike the language described in Section 6, where certain parts are easy to describe and others merely more difficult to describe. Also, in the language of Section 6, facts which were originally difficult to describe can be made easy to describe by adding new concepts. Most present day description languages lack this flexibility through expansion.

SIR I,[49] which was suggested by Raphael as the improved and flexible version of SIR, would in its basis be much more similar to the language discussed in Section 6.

In its generality, the language in Section 6 (as well

as SIR I) is a First Order Theory in the sense of Symbolic Logic. Hence, the testing of the truth of certain statements will, in most general cases, turn out to be a search for appropriate steps of the proof. This is at present extremely difficult.[41, 42] On the other hand, many theorems in the system can be proved by simple processes, as was shown in the first part of Section 6. Even the addition of certain flexibilities of the language appearing in the second part of Section 6 does not vitiate the facility.

The present chapter has discussed the role played by languages in the description of patterns. It has neglected the discussion of languages where the basic predicates involve arithmetic operations: although it is clear both from Metamathematics and from the discussion in Section 6, that such operations can be included in the languages discussed in this chapter. However, these arithmetic languages (and "statistical" languages) are best discussed in terms of their "generalizing" ability. This will be done in the next chapter.

CHAPTER V - LEARNING AND GENERALIZATION

1. Introduction

In Chapter IV, the major concern was to develop languages in which sets of objects could be described in such a way that a specific object could be tested for membership in a set in terms of the object's properties. Associated with any technique of concept learning - whether it be by discriminant functions,[38] probability estimation,[50] and such like - has to have a language in which expressions can be written to define a set. The major points of difference between the ones described in Chapter IV and the more popular ones in the field lie in the following way. Initially, the languages described in Chapter IV are essentially non-numerical, so that the objects do not have to be pre-processed to yield numerical values of the properties. In the field of Pattern Recognition, this pre-processing is essentially a phenomenon left out of the learning and recognition techniques analysed. It is the belief of the author that this separation of pre-processing from recognition introduces great difficulties in the way of answering the most important question in the field to-day, "How does one determine the most effective pre-processors in a Pattern Recognition problem?" It is to be noted that in the languages described in Chapter IV, the

non-input properties stand for the results of pre-processing the input properties. Thus the pre-processing is described in the same format as the patterns are described. Thus, the "effectiveness" of pre-processors can be discussed in an uniform way. It was indicated while discussing the language of Section 4.6 that this does not necessarily remove the advantages inherent in numerical processing.

This chapter will describe certain algorithms for pattern learning using two of the languages described in the previous chapter. It will be indicated how the resulting descriptions are "succinct" when the pattern being learned fulfills certain conditions. Although one of the algorithms indicated will be strong enough to learn any concept (rather than only those which satisfy the given conditions) the description learned in the general case ceases to be succinct. It will be indicated how this lack of succinctness affects the statistical degree of confidence in the learned description.

The arguments used in these discussions will be shown to have direct analogs to cases where the description is written in terms of discriminant function or maximum likelihood ratios, as is often done in the literature. It will be indicated how these arguments point towards the great need for a meaningful discussion of feature extraction or "concept formation."

Since the flexible languages discussed in Chapter IV can describe pre-processing in the same format as descriptions,

one can discuss the modification of P in an environment to render descriptions succinct.

2. Learning Conjunctive Concepts

The algorithm described in this section was developed by Pennypacker for developing conceptions (see Chapter IV, Section 3) for patterns on the basis of examples of objects belonging to the patterns.[32] Unlike most experiments conducted in the field (except those conducted with psychological interest by Bruner and his followers,[19, 30]), the algorithm will be given the freedom of choosing examples on the basis of past examples shown by the trainer; also, it will be given the freedom of asking two other questions, "Is the pattern described by the following conception completely contained in the pattern under consideration?", and "Is this a correct conception for the pattern under consideration?" Both these questions can be replaced by statistical tests and this latter may be necessary in real circumstances. However, the purpose of the investigation was to establish the logical structure of the algorithm.

The basis of the algorithm lies in the isolation of a set of property values $G = \{p_{1i_1}, p_{2i_2}, \ldots, p_{si_s}\}$ such that $p_{1i_1} \cap p_{2i_2} \cap \ldots \cap p_{si_s} \subseteq X$ where X is the pattern being learned and such that there is no subset of G whose intersection is contained in X. If X is a conjunctive pattern, this basic algorithm converges to yield a short conception for X. Otherwise it yields a conjunctive pattern which is a proper subset of X. Examples outside the pattern and inside X are then interrogated to yield other conjunctive patterns. The

process continues till a set of conjunctive patterns are obtained whose union covers X.

The operation of the algorithm needs the ability to obtain conceptions of Boolean functions of patterns having known conceptions and testing for identity of concepts and containment of one concept in another. It also needs the capability of evaluating properties of objects, given the values of its input properties. Algorithms for doing these have been developed by Pennypacker: some of these have been briefly discussed in the previous chapter.

The operation of the algorithm depends on the following lemmata.

<u>Lemma 5.1:</u> Let $\langle U, \mathcal{P} \rangle$ be an environment and let

$$p_{1i_1} \cap p_{2i_2} \cap \ldots \cap p_{ni_n} \subseteq p_{k_1j_1} \cap p_{k_2j_2} \cap \ldots \cap p_{k_sj_s}$$

where for all $t(1 \leq t \leq n)$ $p_{ti_t} \in P_t \in \mathcal{P}$

and for all $r(1 \leq r \leq s)$ $p_{k_rj_r} \in P_{k_r} \in \mathcal{P}$

Also, let $p_{2i_2} \cap \ldots \cap p_{ni_n} \neq p_{1i_1} \cap \ldots \cap p_{ni_n}$

and $p_{2i_2} \cap \ldots \cap p_{ni_n} \subseteq p_{k_1j_1} \cap \ldots \cap p_{k_sj_s}$

Then for no $r(1 \leq r \leq s)$ $p_{k_rj_r} = p_{1i_1}$

<u>Proof:</u> Assume to the contrary; then

$$p_{2i_2} \cap \ldots \cap p_{ni_n} \subseteq p_{k_1j_1} \cap \ldots \cap p_{k_sj_s} \subseteq p_{k_rj_r} = p_{1i_1}$$

whence,

$$p_{2i_2} \cap \cdots \cap p_{ni_n} = p_{1i_1} \cap p_{2i_2} \cap \cdots \cap p_{ni_n}$$

contrary to hypothesis.

<u>Lemma 5.2:</u> Let $\langle U, \mathcal{P} \rangle$ be an environment and $\{P_1, P_2, \ldots, P_n\} \subseteq \mathcal{P}$. Let

$$\emptyset \neq p_{1i_1} \cap p_{2i_2} \cap \cdots \cap p_{ni_n} \subseteq p_{k_1 j_1} \cap p_{k_2 j_2} \cap \cdots \cap p_{k_s j_s}$$

where for each $r (1 \leq r \leq s)$, $1 \leq k_r \leq n$ and for all m, $p_{mi_m} \in P_m$ $(1 \leq m \leq n)$. Then for each $r (1 \leq r \leq s)$

$$j_r = i_{k_r}$$

<u>Proof:</u> Let $k_r = t$. Then

$$p_{1i_1} \cap p_{2i_2} \cap \cdots \cap p_{ni_n} \subseteq p_{ti_t} = p_{k_r i_{k_r}}$$

and

$$p_{k_1 j_1} \cap p_{k_2 j_2} \cap \cdots \cap p_{k_s j_s} \subseteq p_{k_r j_r}.$$

The theorem follows since P_{k_r} is a partition and

$$p \quad \cap p_{k_r i_{k_r}} \neq \emptyset.$$

<u>Lemma 5.3:</u> Under the hypothesis of Lemma 5.2 if

$$p_{2i_2} \cap \cdots \cap p_{ni_n} \not\subseteq p_{k_1 j_1} \cap \cdots \cap p_{k_s j_s}$$

then for some $r (1 \leq r \leq s)$

$$k_r = 1$$

<u>Proof:</u>

$1 \leq k_r \leq n$ for all $r (1 \leq r \leq s)$

and

$$k_r \neq 1 \text{ for all } r(1 \leq r \leq s)$$

indicates

$$2 \leq k_p \leq n \text{ for all } p$$

contradicting hypothesis

<u>Lemma 5.4:</u> Under the hypothesis of Lemma 5.2 let

$$p_{1i_1} \cap \cdots \cap p_{ni_n} \neq p_{k_1 j_1} \cap \cdots \cap p_{k_s j_s}$$

and let $\{p_{1i_1}, p_{2i_2} \cdots p_{ti_t}\}$ be the set of <u>all</u> <u>terms</u> on the left hand side such that $p_{mi_m} \neq p_{k_r j_r}$ for any $r(1 \leq r \leq s)$.

Then,

$$p_{(t+i)i_{t+1}} \cap \cdots \cap p_{ni_n} \neq p_{1i_1} \cap \cdots \cap p_{ni_n}.$$

<u>Proof:</u> By construction of the set $\{p_{1i_1}, \ldots, p_{ti_t}\}$, the left hand side is equal to $p_{k_1 j_1} \cap \cdots \cap p_{k_s j_s}$, which directly contradicts the hypothesis.

The algorithm can now be justified rigorously. Any object is an intersection of a set $\{p_{ij_i}\}$ of property values. If this object is properly contained in a conjunctive concept which is to be learned then the hypothesis of Lemma 5.2 is fulfilled. If now one removes from the set $\{p_{ij_i}\}$ one value $p_{i_o j_o}$ then one of four things may occur

(i) The new concept obtained by intersecting the

elements of $\{p_{ij_i}\} - p_{i_oj_o}$ is the same as the object.

(ii) The new concept as obtained above contains the object properly and is properly contained in the conjunctive concept being learned. This fulfills the condition of Lemma 5.1 and hence, $p_{i_oj_o}$ does not occur in the expression for the conjunctive concept to be learned and hence, can be removed from the set $\{p_{ij_i}\}$ without violating the hypothesis of Lemma 5.2.

(iii) The new concept as obtained in (i) above coincides with the concept to be learned. This terminates the learning process.

(iv) The new concept is not contained in the concept being learned. Then Lemma 5.3 holds and $p_{i_oj_o}$ occurs in the expression for the conjunctive concept being learned.

If case (ii) holds, one can start the process over again by removing a new property value $p_{i'_oj'_o}$ from the set $\{p_{ij_i}\} - p_{i_oj_o}$. If (iv) holds, then $p_{i_oj_o}$ is left in the set $\{p_{ij_i}\}$ and never removed again. In either case, the test to be performed on the set $\{p_{ij_i}\}$ is constrained to be performed on

a smaller set. Hence, if on each removal of a $p_{i_o j_o}$ only (ii), (iii) or (iv) holds the successive removal comes to an end. If it does not come to an end at (iii), then the concept to be learned is not conjunctive.

It is to be noted that the above discussion provides the rationale for Bruner's conservative focussing strategy. Case (i) never occurs in his experiments since his input properties are all the properties in the environment and the input properties form a full fine structure family (see Chapter IV). In the present case, where the environment contains many non-input properties so that the entire property set is not full, the algorithm needs the modification discussed below.

If (i) above is the case, then other property values have to be removed from the set $\{p_{ij_i}\}$. If (i) holds for all values so removed, then combination of two values are removed from $\{p_{ij_i}\}$ and the process is repeated. At this point fulfillment of (iv) does not yield any result, since the condition of Lemma 5.2 is not necessarily fulfilled any more. However, the fulfillment of condition (ii) is still significant, since the hypothesis of Lemma 5.2 was not involved in the proof of Lemma 5.1.

By Lemma 5.4, as long as there is any property value left in $\{p_{ij_i}\}$, there will be some combination of property

values whose removal will result in fulfillment of condition (ii). This way all property values not occuring in the expression of the conjunctive concept is removed in a finite number of operations and condition (iii) occurs. If, however the concept being learned is not conjunctive then condition (iv) occurs on the removal of any property value. In this case a new object is chosen which is contained in the concept being learned but not in the conjunctive concepts learned so far and the process is repeated. Since in a finite environment any concept is the union of a finite number of conjunctive concepts (see discussion following Theorem 4.12), the process terminates with the recognition of the concept.

The above discussion constitutes an informal proof of the following theorem.

<u>Theorem 5.5:</u> In a finite environment the algorithm shown in flow chart of Figure 5.1 terminates in a finite number of steps with the recognition of a concept.

It may be pointed out that the finiteness of the algorithm does not in any way assure a short description of the concept learned, because a short description may not exist at all. Also, the process, though finite, may be inordinately long. This also has some extremely adverse effect on the "generalization" of the concept. This latter will be discussed in a later section. In the next section an algorithm will be discussed which learns simple concepts in a finite number of steps. It utilizes the language discussed in Section 5, and

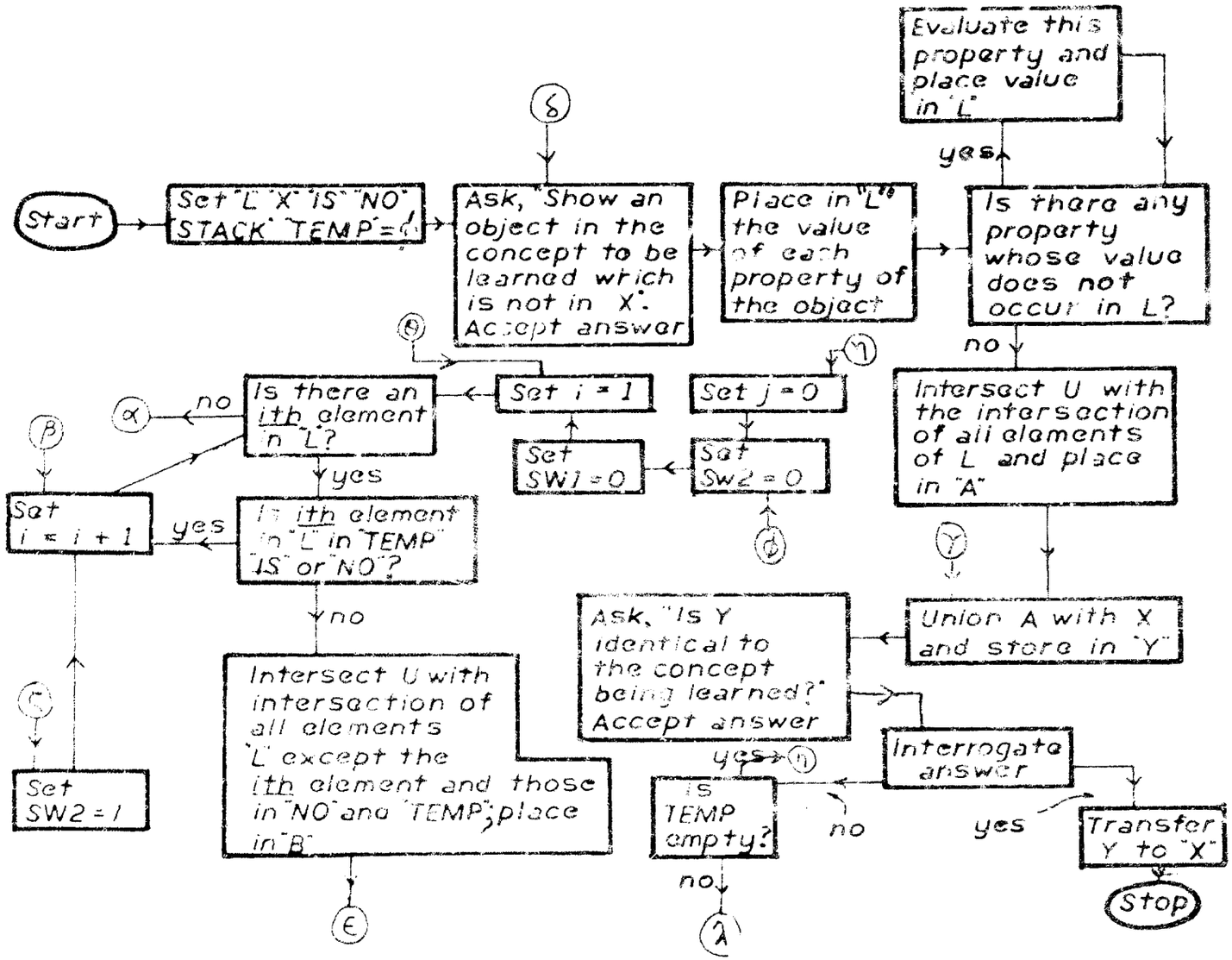


Figure 5.1
(part 1)

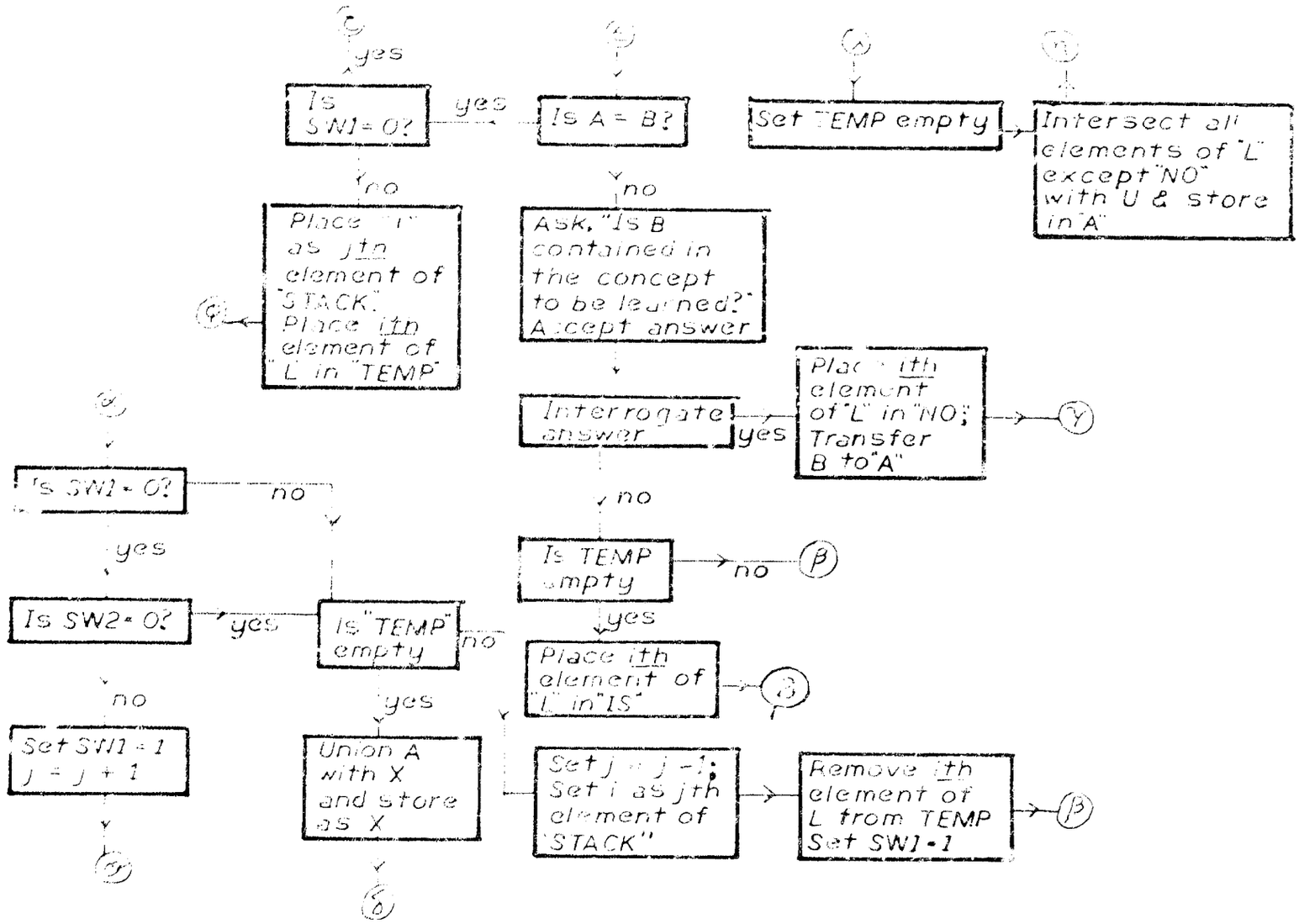


Figure 5.1
(part 2)

hence, can construct simple descriptions for a much richer class of concepts than the conjunctive. However, in many realistic cases (for instance, in environments where properties are two-valued) even this excludes many concepts.

3. Learning Simple Concepts

This section will exhibit an algorithm which, given a set of objects $\{X_1, X_2, \ldots, X_p\}$ generates a subset T of K (see Chapter IV, Section 5), such that $H(T) = \{X_1 \cup X_2 \cup \ldots \cup X_p\}^S$. This algorithm will then be used as a basic component of a method for generalization and learning.

Given a concept X, one defines a set G_X of property values ($G_X \subset K$)

$$G_X = \{p_{ij} : p_{ij} \in P_i \cap P_i \in \quad \text{and} \quad p_{ij} \in \overline{X}\}.$$

From Theorem 4.11 and the fact that $p_{ij} \in \overline{X}$ if and only if $p_{ij} \cap X = \emptyset$ one sees right away that an alternative method for writing an expression for X^S is

$$X^S = \overline{\bigcup_{p_{ij} \in G_X} p_{ij}} = H(G_X)$$

Also

<u>Lemma 5.6:</u> If $H(T) = X^S$, then $T \subset G_X$

<u>Proof:</u> Since $\overline{X^S} = \overline{X}$

$$\overline{H(T)} = \overline{\bigcup_{p_{ij} \in T} p_{ij}} = \overline{T} = \overline{X^S} = \overline{X}$$

Hence, $p_{ij} \in T$ implies $p_{ij} \in \overline{X}$ or $p_{ij} \in G_X$.

This Lemma indicates that G_X is the largest subset of K such that $H(G_X) = X^S$. That is, G_X is identical to M(X) as defined in Chapter IV.

Let now $\{X_1, X_2, \ldots, X_p\}$ be a sequence of objects. Associate with this sequence a sequence of subsets $\{K_0, K_1, \ldots, K_p\}$ of K as follows

$$K_o = K$$

If $K_i = \{t_i, t_2, \ldots, t_m\}$ and $X_i = p_{1i_1} \cap p_{2i_2} \cap \ldots \cap p_{ni_n}$ then $K_{i+1} = \{t | t = t_i$ from some $i (1 \le i \le m)$ and $t \notin p_{ki_k}$ for any $k (1 \le k \le n)\}$.

The following lemma is important.

<u>Lemma 5.7:</u> $K_p = M(X_1 \cup \ldots \cup X_p)$.

<u>Proof:</u> Let $p_{ms} \notin K_p$. Then there is some $X_t (1 \le t \le p)$ such that $X_t \subseteq p_{ms}$.

But $p_{ms} \in M(X_1 \cup \ldots \cup X_p)$ implies $p_{ms} \subseteq \overline{X}_1 \cap \overline{X}_2 \cap \ldots \overline{X}_p \subseteq \overline{X}_t$ which leads to a contradiction. Hence, $p_{ms} \notin M(X_1 \cup \ldots \cup X_p)$. Hence, $\overline{K_p} \subseteq \overline{M(X_1 \cup \ldots X_p)}$ or $K_p \supseteq M(X_1 \cup \ldots X_p)$.

Now let $X_t = p_{1i_1} \cap \ldots \cap p_{ni_n}$. Then $p_{mi_m} \notin K_p$ for any $m (1 \le m \le n)$. Hence, $X_t \subseteq p_{mn}$ implies $p_{mn} \notin K_p$ or

$$K_p \subseteq \{p_{mn} | X_t \not\subseteq p_{mn}\} = \{p_{mn} | X_t \cap p_{mn} = \emptyset\} = \{p_{mn} | p_{mn} \subseteq \overline{X}_t\} = M(X_t)$$

(Since X_t, being an object is either wholly contained in or disjoint from any property value: see Theorem 4.1).

So $K_p \subseteq M(X_t)$ for all $t (1 \le t \le p)$ or $K_p \subseteq \bigcap_{t=1}^{p} M(X_t) = M(X_1 \cup \ldots \cup X_t)$ (Theorem 4.14).

This with the previous inequality yields the lemma.

One can thus construct the following algorithm for

learning a concept. The algorithm starts with two copies of K (K_0^1 and K_0^2) in memory. Every time an object $p_{1i_1} \cap \ldots \cap p_{ni_n}$ is presented as belonging to a concept X, the values p_{mi_n} $(1 \le m \le n)$ are removed from K_i^1 to yield K_{i+1}^1. Similarly, any time the object is presented as belonging to $\overline{X}$, K_i^2 is similarly modified to K_{i+1}^2. At any stage of learning, if m positive and n negative instances are presented, then $H(K_m^1) \subseteq X^s$ and $H(K_n^2) \subseteq \overline{X}^s$.

If X and $\overline{X}$ are both simple concepts then the algorithm converges at some value of m and n such that $H(K_m^1) = X$ and $H(K_n^2) = \overline{X}$. However, if either X or $\overline{X}$ is not simple, then one has to remain satisfied with the approximation to X given by X^s and $\overline{\overline{X}^s}$.

A test has been developed by Windeknecht and Snediker to find out if a concept and its complement are both simple.[33, 34] The test depends on the following lemma.

<u>Lemma 5.8:</u> $x^s = x$ and $\overline{x}^s = \overline{x}$ if and only if $x^s \cap \overline{x}^s = \emptyset$

<u>Proof:</u> The "only if" part is immediate. For the "if" part one notes that $x \cup \overline{x} \subseteq x^s \cup \overline{x}^s$. But $x \cup \overline{x} = U$. Hence, $x^s \cup \overline{x}^s \supseteq U$, yielding $x^s \cup \overline{x}^s = U$. Hence, $\overline{x}^s = \overline{x^s}$ from hypothesis or $x^s = \overline{\overline{x}^s}$. But by Theorem 4.10, $x \supseteq \overline{\overline{x}^s} = x^s$ and also, $x \subseteq x^s$ whence $x = x^s$. $\overline{x} = \overline{x}^s$ follows similarly.

The application of this test depends on a test for

$H(K_m^1) \cap H(K_m^2)$ being empty or, according to Theorem 4.14 for $H(K_m^1 \cup K_m^2) = \emptyset$. The test for this is not straight forward; given a subset $T \subseteq K$, it may not be easy to find out if $H(T) = \emptyset$. Clearly, for any $P \in \mathcal{P}$, $H(P) = \emptyset$. Also, if $T' \supseteq P$ then $H(T^1) \subseteq H(P) = \emptyset$. However, there may be some T which does not contain any $P \in \mathcal{P}$ as a subset and yet $H(T) = \emptyset$. The set $\{R_2, R_3, R_4, T_3\}$ in the example of Chapter IV is an example.

A rather involved procedure has been developed by Snediker for the test. Rather than describing the test in detail here, it may be more worthwhile to discuss the effectiveness of the procedures discussed here and in the previous sections in realistic situations and compare them with other well known Pattern Recognition techniques.

4. Problems of Learning and Feature Extraction

A study of either of the methods of learning in the two previous sections is very illuminating in that it brings to the attention of the reader some of the major difficulties in the way of pattern learning and points out some of the important requirements for an effective pattern learning technique.

It will be noticed in the case of both the methods that they are most effective in learning patterns (or concepts) in certain classes. The Pennypacker technique, although an effective procedure for all concepts, ensures rapid convergence only for conjunctive concepts. The Windeknecht-Snediker technique converges to the corract concept only if the concept is simple. In the Pennypacker technique, however, there is a technique for finding out when a concept is not conjunctive and modifying the algorithm to take account of this fact. In the Windeknecht-Snediker Algorithm, the corresponding test indicates, not whether the concept being learned is simple, but whether both it and its complement is simple. No algorithm has been developed which would learn any concept as an union of simple concept in a way analogous to the Pennypacker algorithm.

However, the Pennypacker algorithm takes certain liberties which are not used by any other algorithm known to us. It asks the experimenter questions about the inclusion relationship between the concept being learned and concepts described by the algorithm. The Bruner conservative focussing

strategy on which the Pennypacker algorithm is based, did not allow these liberties, although it did envisage questions from the subject regarding memberships of specific objects in the concept being learned. The extra liberties were necessitated by the fact that unlike in Bruner's case (and the case of most psychological work after him), it was not assumed that the input properties of the real environment is full. This invalidates some of the methods used in the psychological experiments for obtaining new objects from a focus object.

It is difficult to say how many of the advantages of the Pennypacker algorithm over the Windeknecht-Snediker algorithm would remain if the extra liberties were taken away. Just as one needs to develop methods for asking Bruner-type questions ("membership rather than inclusion") in the environment envisaged by Pennypacker, methods need to be developed also for modifying the Windeknecht-Snediker algorithm to the cases where the concepts learned are non-simple. However, in any case, since there are more simple concepts in an environment than there are conjunctive ones, the Windeknecht-Snediker method ought to be more effective in general. However, in environments where all properties have only two values, all simple concepts are conjunctive. In this case the relative advantages disappear.

The weaknesses and strong points of the learning methods discussed in this chapter may be used to develop a set of criteria for the evaluation of concept learning methods in

general. In the previous paragraphs the methods of this chapter have been discussed on the basis of the following questions.

1. How rich is the class of concepts any one of whose members can be learned by this method?
2. How rich is the class of concepts any one of whose members can be learned efficiently by this method?
3. How rich is the class of concepts whose descriptions are succinct when expressed in the language envisaged by the algorithm?
4. Given the interpretation of an environment as a real pattern learning situation, how many patterns to be learned can be expected to be members of the class described in questions 1, 2 and 3 above?

It can be seen that the class described in question 1 above contains the class described in question 2. (Nothing can be learned efficiently unless it is learned!) In the case of many methods, this latter class coincides with the class described in question 3. However, this may not be true for all methods and languages. A mathematical study of this point can not be attempted unless precise and acceptable definitions of the words, "succinct" and "efficient" be given. This will not be attempted.

Question 4, perhaps, needs some clarification, since

it refers, not to an abstract entity called the "environment" in the discussion, but to the unformalizable thing called "real life," and the way one abstracts it to an "environment" in the technical sense. In question 4, the words, "patterns to be learned" refers to "real life," as for example, the class of "all roman letters projected on a grid of photo-cells" while the "classes described in questions 1, 2 and 3" refer to the describable classes after a class of properties have been abstracted and used in a mathematical system. If in "real life" one was called upon to learn all concepts possible (for instance, if learning the class containing "all lower case "a"s, upper case "Q"s and all symmetrical figures" was as necessary as learning the class of all upper case "B"s) the answers to question 4 would coincide with the answers to questions 1, 2 and 3 respectively. However, this is often not true.

It has already been seen that in the case of the Pennypacker technique the class described by question 1 is, "the class of all concepts." The class described by questions 2 and 3 is, "the class of all conjunctive concepts." Of course, how rich this latter class is depends on the richness of the family of properties in the environment. This latter point will be discussed presently. Meanwhile, it may be worth pointing out that if one restricts oneself to a family of input properties (like, say, "the excitation level of each photo-cell in the grid") the class of conjunctive concepts is not very rich, especially with respect to "real life."

It ought to be pointed out, however, that the flexibility of the languages described in Chapter IV is such that the definition of new properties are extremely easy to incorporate into the language. This can be done, moreover, with respect to the class of patterns described in question 4. No efficient algorithm exists for introducing such new properties, but certain heuristics can be considered. This is done below with respect to the Universe exemplified in Chapter IV, Section .

If the environment consists of the two properties p and q then the only conjunctive concepts are the ten values of p and q and the twenty-five objects. Let us now assume that the concept $A = \{3,4,13,14,15,16,17,18,1,2,11,12\}$ has to be learned. The only way the Pennypacker algorithm could learn the concept would be as an union of exemplars. This would render the learning process extremely inefficient and also the conception of the pattern learned would be unwieldy. The same would be the case with respect to learning the concepts $B = \{5,6,7,8,9,10,19,20\}$, $C = \{21,22,26,27\}$ and $D = \{23,24,25,28,29,30\}$. However, at this point it could be realized (if we had an algorithm strong enough to do it) that $A \cup B$ (or T_1 so far unknown), $C \cup D$ (or T_2) and $U - T_1 - T_2$ could be used as a property of the environment and so could $A \cup C = R_2$, $B = R_4$ and $U - R_2 - R_4$. This would yield $A = R_2 \cap T_1$ and $C = R_2 \cap T_2$ at a considerable increase in succinctness of description. However, this succinctness would

be purchased at the expence of storing T_1, T_2, R_2, R_4, T_3 and $R_3 \cup R_1$ in the description list of the Universe. Such property generation, then, can only be justified if they yield shorter descriptions of many concepts. This leads to efficient learning of concepts encountered later. Also, it has profound significance with respect to the "generalizing ability" of the learning algorithm, as will be shown in the next section.

The reader will note that the concepts T_1, T_2, R_2 and R_4 as defined above led to new descriptions of concepts which would not have been learned from examples. They would have been internally generated to facilitate the storage of concepts which have been learned from examples. Such concepts are generally called "features" in the literature and process which isolates them are called "feature extraction." In this book, the term, "concept formation" has also been reserved for this phenomenon.

It will be worthwhile at this point to consider some of the other learning algorithms in literature and how they stand with respect to the presently described methods.

Because of the similarity of the description languages, the first methods that come to mind are the EPAM,[31] and the CSL-I.[30] The comparative advantages of the languages have been discussed beforehand. The major point to be made about EPAM is really in connection with the limitations of the EPAM tree. Because the name of the concept occurs at the root of the tree, two non-disjoint concepts can not be very well described in

an EPAM tree. This puts essential restrictions on EPAM as a learning algorithm; however, a recognizer using the EPAM tree as the description makes an extremely efficient property evaluator.

Some of the drawbacks of the EPAM net have been removed by Ernst and Sherman.[51] It has enabled the building of a description language incorporating some of the highly desirable characteristics of the predicate calculus language described in Section 6 of the previous chapter. Since the exact form of the Ernst and Sherman language is not completely formalized, a detailed description of the language will not be given here. However, it may be worthwhile to point out a few important characteristics.

The Universe of Discourse of the language has two properties, "Ex" and "Na." The values of "Ex" (acronym for "exemplar") are objects in the sense discussed in Section 6 of Chapter IV. The values of "Na" are concept names. The EPAM-like tree describes a single concept consisting of exemplar-name pairs such that each exemplar belongs to the concept having the paired name. Thus, if an object X belongs to both the concepts A and B then in the Ernst-Sherman language the objects (Ex, X; Na, A) and (Ex, X; Na, B) would both belong to the concept described by the tree.

The other major departure of the Ernst-Sherman tree from EPAM lies in the fact that the test nodes of the tree may contain statements of the form "term < term" as well as

"term = term" while the conventional EPAM test nodes consist of the latter types only. This enables the language to have some of the advantages of the language of Chapter IV, Section 6. As a result, the Ernst-Sherman learning algorithm can make use of previously learned concepts in describing new concepts and thus, shows an important aspect of truly adaptive behavior.

Like most learning techniques based on Boolean Algebraic methods (the Windeknecht-Snediker technique being a notable exception), the learning algorithm is most effective in learning conjunctive concepts; however, like the Pennypacker algorithm, it can learn any concept. Moreover, it can learn concepts whose descriptions involve statements of the form "term ϵ term."

The CSL-I technique of Hunt, learning with the description-tree developed by him, has one capability which the Pennypacker algorithm lacks: it can learn succinct descriptions of concepts whose complements are conjunctive. To do this, it has to store in memory all the objects shown in the concept and its complement, instead of modifying the description with each new presentation of an exemplar as the Pennypacker algorithm and its parent, Bruner's conservative focussing strategy, does. Unlike the Pennypacker method, there is no method in CSL-I for using non-input properties in the description. As a matter of fact, the advantages of having non-input properties (an advantage which is used by all human beings) seems to have been completely neglected in all

psychologically oriented structures of Pattern Recognition that the present author has come across.

On a superficial study, it might appear that the numerical techniques of Pattern Recognition discussed often in literature are far stronger than the ones discussed here. As a matter of fact, there is a tendency to include in the field of Pattern Recognition only techniques based on the theory of vector spaces and probability. The importance of the study and development of flexible description languages as done here often seems to be outside the pale of the field of Pattern Recognition. This is extremely hard to understand in view of the constant bemoanings in the Pattern Recognition field regarding the elusive nature of the "feature extraction" problem, which is intimately associated with the basic predicates of description languages.

In what follows, a short discussion will be given of the present author's interpretation of the methods and results in the field of numerical Pattern Recognition.

As has been pointed out before for these methods to be effective, one has to have properties whose values can be represented as real numbers. Thus, in any environment with a finite number of input properties (and no distinction has yet been attempted between input and non-input properties), each object (many authors prefer to call the objects "patterns" but it will be safer here to hold to an uniform terminology) is represented by a vector in the space of n-tuples of reals.

The learning algorithms, on the basis of a list of objects, tagged by their membership in a given concept C, constructs a real function f of n variables (the "discriminant function") such that for a large number of objects x encountered or expected to be encountered f(x) would be positive if and only if the object belonged to C. In symbols

$$f(x) > 0 \equiv x \in C$$

Just as in the case of the algorithms described previously, the form of the function f is restricted to a class, at least by the efficiency of recognition. That is, for some of the algorithms the class of concepts described in question 1 is restricted and in others this specific class is unrestricted while the class described in questions 2 and 3 are restricted and, in most cases, identical. The class described in Section 4 can only be considered on the basis of experimentation and the results of the experiments. Different methods have varied, both with respect to their quality of the results and conclusiveness of the experiments.

Comparisons of these different learning techniques are generally made on the basis of the operational mode of the learning algorithm. One criterion for this is whether a technique is adaptive, i.e., whether the algorithm stores all the tagged objects and constructs the function f on the basis of the entire set of tagged objects or whether the function f, starting from an arbitrary initial value, is modified by each tagged object in succession, so they do not have to be stored for processing 'en masse.'

Another important criterion for distinction may be on the basis of "motivation," i.e., the basis of choosing the class of functions to be constructed. In some cases this function is generated on the basis of the estimation of the parameters in a set of probability distributions.[52, 53] That is, it is assumed that the concept C and its complement $\overline{C}$ are such that there exists two distributions p and q, characterized by a vector of parameters $\overline{\theta}$ such that the function f has the form

$$f(x) = \frac{p(x;\overline{\theta})}{q(x;\overline{\theta})} - 1$$

for some parameter-vector $\overline{\theta}$. The class f is determined by the forms of p and q and the allowed range of choice of the vector θ. The forms for p and q are chosen either on the basis of the workers belief (on the basis of empirical data, hopefully) that the distributions p and q are adequate or by the fact that the estimation of the parameters is computationally feasible for a large set of tagged objects if p and q are assumed to have some given form. Unfortunately, reality does not often conform to the conveniences or limitations of the theoretician.

Another basis for the choice of the class of functions f may be dictated by certain "distance functions."[37] That is, one starts with the axiom that there is a metric ρ on the space of n-tuples such that if A is the set of all objects tagged as belonging to C and B the set of all objects tagged as belonging to $\overline{C}$, then f is often defined by

$$f(x) = \min_{y \in A} \{\rho(x,\bar{y})\} - \min_{y \in B} \{\rho(x,\bar{y})\}$$

or

$$f(x) = \bar{\rho}(x,\bar{y}) \Big|_{\bar{y} \in A} - \bar{\rho}(x,\bar{y}) \Big|_{\bar{y} \in B}$$

$\bar{\rho}$ denoting average value. Again, the class f is determined on the basis of the investigator's choice of ρ - hopefully on some rational or empirical basis. Often the class of f chosen by different methods turn out to be the same.

A large number of authors restrict the class f directly without reference to any statistical or metric criteria - which to the present author's mind is no less justifiable than choosing the class on the basis of the faiths discussed above. The most popular form, of course, is the linear one where

$$f(x) = \bar{a} \cdot x + b$$

where $\bar{a}$ is a vector and b a real number.[54] It appears that although the class of concepts describable by these linear functions (the "linearly separable patterns") is much richer then the class of functions discussed in the previous sections, it is still a very small fraction of the class of all possible concepts - even where the vector space under consideration is the finite space of all possible binary sequences. What is worse, even the class of concepts described by question 4 turns out to be inadequate on the basis of experimental evidence, unless the set of properties defined by the components of the vector is "adequately chosen" - and there is no uniform way of choosing the "adequate" representation.

The class f has been enriched by many workers by including non-linear functions - especially polynomials of large degree. The only major difficulty with respect to such choice lies with the very large number of coefficients needed for an adequate description of the concepts. This difficulty is analogous to the cases where the Pennypacker technique learns a concept as the union of an inordinately large number of conjunctive concepts. Another analogous case arises where the description is taken to be "piecewise linear," i.e., where the description of C takes the form

$$x \in C \equiv B((f_1(x) > o) , (f_2(x) > o), \ldots, (f_p(x) > o))$$

where B is a logical combination of the statements $\{(f_i(x) > o | 1 \leq i \leq p\}$ and $f_i(x)$ is a linear function. A subclass of the class of functions so describable are those describable by the so called "two layer nets," i.e., where B yields a linearly separable function, so that the statement above can be rewritten

$$x \in C \equiv \sum_{i=1}^{p} a_i \operatorname{sgn}[f_i(x)] + a_o > o$$

where $\operatorname{sgn}[t] = 1$ if $t > o$ and o otherwise. "Multi-layer nets" can be similarly constructed, yielding richer classes of describable concepts.

When one considers adaptive techniques for the evaluation of coefficients appearing in the representations in any of the schemes described above an important question arises regarding the "convergence" of the training scheme. Convergence

proofs are known only for some of the algorithms based on statistical estimation,[53] as also algorithms on the basis of linear separability.[55] The earlier algorithms were proven to converge only in cases where the tagged objects were from a linearly separable concept and its compliment. Algorithms have been suggested recently where the algorithms also indicate failure when the concept is not linearly separable.[56] In many cases algorithms are introduced on the basis of empirical evidence that they converge "in many cases." Nothing is known regarding the convergence of algorithms for "multi-layer nets," although empirical algorithms have been used for designing these in the literature, both for Pattern Recognition and game-playing.

As has been indicated above, another major difficulty with non-linear or piecewise linear discriminant functions (and to the present author's mind these are the only functions which have any promise of success) lies with the extremely large number of coefficients to be stored. An equally important consideration closely connected with this is the "generalizing ability" of the discriminant functions formed from these coefficients. As was said in the beginning of Chapter IV, our discussion has been limited to learning and describing concepts without any reference to the phenomenon of generalization. Some attempts will be made towards discussing generalization in the next section. The discussion will attempt to bring out the importance of appropriate description languages and "features."

5. Generalization - "Concept Formation" and Languages

It has been seen in the previous section that concept formation or feature extraction plays a very important role in simplifying the expressions which describe a concept. There has also been a belief that, somehow, extracting the "correct" features makes subsequent learning easier. Also, that once one is in possession of a correct set of features (i.e., has formed the right concepts) one can generalize from the encountered tagged objects well enough to recognize latter objects with a high degree of confidence based on the descriptions formed by a learning program. In the absence of good features, "rote learning" seems to be the only possible learning method and one cannot generalize well from a description learned "by rote."

In what follows, a preliminary effort will be made to give a rough mathematical framework to give meaning to the terms used above and justification for the beliefs indicated.

The very existence of a possibility of generalization indicates that the class of all concepts to be recognized (the class described by question 4 in the previous section) is restricted to a subset of the class of all concepts. To make this point clear, it may not be necessary to take the most general environment. It will suffice, as an example, to take an Universe having a full fine structure family of input properties having n properties in the family and m values of each property. The main concern here will be the richness of

the class of concepts rather than the simplicity of their description: hence, non-input properties need not be considered.

There are m^n objects in this environment and hence, 2^{m^n} possible concepts. Any time a specific object, $p_{1i_1} \cap \ldots \cap p_{ni_n} = X$, is known to belong to a specific (unknown) concept, those concepts to which X does not belong are eliminated from those under consideration, and the concept to be learned is known to belong to one of 2^{n^m-1} possible concepts. In general, when k_1 objects are presented to a learning algorithm as belonging to a concept and k_2 objects are presented as belonging to the complement of a concept, then there are $2^{n^m-k_1-k_2}$ possible choices for the concept. This number, it will be noticed, does not reduce to 1 (to "correct" learning) till $k_1 + k_2 = n^m$, (i.e., till every object in the Universe has been presented).

A restriction of the class of concepts to be learned, then, seems essential. Such restriction leads to extremely fast convergence (exemplified, for instance, by the Pennypacker Algorithm in the case of conjunctive concepts). However, results of experiments (on the perceptron, for example, or even what can foresee in the future for the CSL or the Pennypacker and Snediker Algorithms, indiscriminately applied) indicate that ad hoc restrictions stand very little chance of "standing up to reality." The restriction has to be learned, just as the concepts themselves have to be learned.

The last sentence has to be persued with some care. It will be noticed that the learning of a concept consists of the learning of the union of a class of objects. The learning of a restriction, on the other hand, consists of the learning of a class of concepts. Although the language of Section 6, Chapter IV, is adequate for describing both sets and classes of sets (so glibly, in fact, that unless $\alpha \notin \alpha$ is introduced as an axiom, contradiction will result!) it is probably premature to suggest that both the learnings go on in the same language! Much better understanding of the "second level" learning (learning of classes) will be needed before that.

For the present, it will be assumed that there exists certain concepts which (even though one is not called upon to learn them through tagged objects) may be used in constructing simple descriptions of concepts that are learned by tagged objects. Second level learning (or "feature extraction" or "concept formation") can thought of as consisting in the recognition of the former concepts. In Section 4, and example was given to indicate how this phenomenon may possibly be made into an algorithm. Very little research has gone in this direction (extraction of masks, as done by Uhr,[57] and Niellson,[58] are efforts in this direction, although they are strongly biased in the direction of character recognition and can fail (see, for instance, BOGART[59]), when tried on more ambitious projects.

It may be somewhat easier to express the thoughts and

to reduce the chance of misunderstanding if the above paragraph is interpreted formally. This will be done in the next few paragraphs. The reader is warned that the only reason for the formalism here is precision. No deeper insight results from it immediately.

Let $\langle U, P, \rho \rangle$ be a real environment, where ρ is the entire class of input properties. Let there also be given a class F of specific modes of combination of sets to obtain new sets (which modes may be operations like union and complementation or may be linear or non-linear threshold schemes). One now defines an ordering $O(\rho, F)$ on the class C_ρ of concepts. If the concept C_1 is lower than concept C_2 in this order, then C_1 has a simpler description than C_2.

Given a class of concepts $C \subseteq C_\rho$, ρ will be called <u>satisfactory</u> for C, if every element of C ranks low in the order $O(\rho, F)$. If ρ is not satisfactory, then a set of concepts C_o will be said to be concepts "formed in view of C" if $\rho \cup \{(c, \bar{c}) | c \in C_o\}$ is satisfactory for C.

Evidently, the class C is "formed in view of C" in a very trivial way. A good concept former would be expected to form a class C_o in some "optimal" way which has not been defined yet.

The restricted class of concepts to be considered for avoiding the difficulty of generalization will be the class which are easy to describe in terms of the language available after concept formation. The major point that ought to be

emphasized in this section is that restrictions based on the assumption of simplicity of description has an extremely strong repurcussion on what one understands by the "generalizing ability" of a learning algorithm.

"Generalizing ability" in the sense discussed before, can at present be identified most closely with the term, "confidence level" as used in the field of statistical hypothesis testing. Also, the slight confusion with respect to the acceptable definition of "concept formation" is reflected remarkably well in the slight confusion that occurs in the use of the word "degrees of freedom" in that field (the present author is thankful to Professor Herbert Simon of Carnegie Tech. for pointing this out).

This last fact can probably be brought out by an example. Let the experiment consist of exhibiting two-digit decimal representations of the first 99 positive integers, tagging some of the representations with 1 and the others with zero. Let all elements of the set {1,11,13,15,27,33,35,47,49,59} be tagged with 1 and let the elements of {4,14,16,18,24,32,38,42, 56,66} be tagged with 0. If the values of the first and second digits define the only two properties of the environment, one can obtain the following contingency table for a Chi-Square test

		Tagged with 0	Tagged with 1
Ending with	1	0	2
	2	2	0
	3	0	2
	4	3	0
	5	0	2
	6	3	0
	7	0	2
	8	2	0
	9	0	2
	0	0	0

The contingency table yields a value of 27 for the Chi-Square. This has a significance level (with the attendant error arising from the small size of the sample and with a degree of freedom 15) of 0.025 which may not be considered significant. On the other hand, if evenness of a numeral is considered to be a property of the environment one could get the following contingency table

		Tagged with 0	Tagged with 1
second digit	even	10	0
	odd	0	10

The χ^2 this time, for the same hypothesis of uniform distribution with the degree of freedom 4 is 20. This value (quite accurately this time) is significant to the level of less than .001.

There is little in the theory of sampling itself to indicate which of the two contingency tables should actually be used for testing significance. It appears that to interpret the phenomenon indicated above inside statistics, the latter may have to be enriched by considerations of the available description language. The suggestion made some paragraphs back regarding restricting the generalizable concepts to easily describable concepts was based on the observation that in any contingency table, the cells are always chosen to be the ones most easily describable.

The number of rows in the contingency table is large if the concept being tested involves the union of a large number of simply describable concepts. This will explain the statement in Chapter IV, Section 1 that the "size" of the connective "or" seems to be larger than that of other usual connectives.

It must be pointed out that even when a concept is described as an union of simply describable concepts, generalization is not impossible. In the first contingency table above, for instance, if the entries in each cell were doubled, one could consider the table as a significance indicator for the hypothesis, "All numerals ending in 2, 4, 6 and 8 are tagged with 1." Only more observations would be needed. If it turns out that the structure of the environment and of the language are such that too many observations are not possible in each cell, generalization is impossible without a change in language. One can think, for instance, of the concept of "even integers,"

described as a piecewise linearly separable concept where the only known feature of an integer is its value. Generalization would be impossible on the basis of observing the two tagged sets mentioned above. On the other hand, if one used the digits in the binary representation of the numeral as features, the concept of even numbers would be linearly separable and, hence, easily generalizable.

A learning and concept forming algorithm, starting from a fine structure family of properties, could learn concepts by some technique in which the class of learnable concepts is not restricted. If the concepts learned by it are not simply describable (and if the extremely unfortunate situation discussed in the previous paragraph does not occur), then the learned concepts need a large number of tagged objects to establish their significance. Once these are established, concepts may be formed to simplify the description of every concept learned. Attempts are then made to learn later concepts within the restriction imposed by the newly formed concepts. In the environment which follows a restriction dictated by these newly formed concepts, generalization of the learned concepts will be easier: otherwise the class of formed concepts would be modified.

It is the author's belief that the development of this kind of algorithms is essential if Pattern Recognition is to become a viable branch of Artificial Intelligence - indeed, if Artificial Intelligence ever has to become a viable field.

So far the discussions have been carried out with

the background of non-numerical description languages and recognition techniques. In the case of techniques based on the assumption that the objects are vectors of n real numbers, analogous discussions remain valid. It may be worthwhile to limit discussions to polynomial discriminant functions. (It may be repeated here that initial restriction of learnable classes to "linearly separable" or "distributed normally" impose conditions on the initial measurements which are extremely ill-understood). That is, let C be such that $x \in C \equiv o \leq P(x)$ where

$$P(x) = \alpha_o + \sum_{i=1}^{n} \alpha_i x_i + \sum\sum_{i,j=1}^{n} \alpha_{ij} x_i x_j$$

$$+ \ldots\ldots + \Sigma\Sigma\Sigma \ldots \Sigma\, \alpha_{ijk \ldots t}\, x_i x_j x_k \ldots x_t$$

$$+ \ldots + \ldots \alpha_{12\ldots n}\, x_1 x_2 \ldots x_n.$$

It will be assumed that if an element is chosen from the concept C, the probability that a specific x is chosen is $f(x,c)$; similarly the probability of x to be chosen when an element is chosen from $\bar{c}$ is given as $f(x,\bar{c})$. It will be noted that in Baysian techniques of Pattern Recognition, it is the parameters of f that are estimated. However, since the "a"s of the above polynomial are functions of these parameters (note that $f(x,c) = o$ if $P(x) > o$) it will be assumed that the estimation of the "a"s is the matter in issue.

Let now k_1 vectors $(\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_{k_1})$ be presented to

the learning algorithm tagged with 1 and k_2 vectors $(z_1, z_2, \ldots, z_{k_2})$ are presented tagged with 0. On the basis of these vectors the algorithm estimates coefficients b_o, $\{b_i\}$, $\{b_{ij}\}$ ..... $\{b_{123\ldots n}\}$. (Of course, in any practical case, many of the higher degree coefficients will be zero.) The "b"s are functions of the variables $\{\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_{k_1}\}$ and $\{\bar{z}_1, \ldots, \bar{z}_{k_2}\}$, which have probability distributions $f(\bar{y}, c)$ and $f(\bar{z}, \bar{c})$ respectively. Hence, each b will have a probability distribution and need not be equal to the "a"s unless k_1 and k_2 be extremely large. However, if the learning procedure is any good, the distribution of each b will be centered around the corresponding a.

In the language of the theory of small samples, each coefficient b is an estimator of the corresponding a. While in the past workers in the field have been generally satisfied if the estimates are unbiased, for discussion of generalization, the efficiency of these estimators must be known. The efficiency will be given by the estimation procedure and the distribution $f(\bar{x}, c)$. However, in the general case, the following discussion is germaine.

In any good estimation technique each "b" will have the corresponding "a" as mean and will have some variance δ which will decrease with increasing k_1 and k_2. However, the rate of convergence can be seen to be seriously restricted by the number of "b"s being estimated.

The number of degrees of freedom is not $k_1 + k_2$ but is reduced by the number of parameters being estimated. As a result, the number of observations needed for generalization become larger, the larger the number of parameters estiamted (i.e., the more complex the discriminant function is). The reduction of the number of parameters is only possible by using carefully chosen measurements for the components of the vector. Even though the input properties look "naturally numerical" that is no indication that the natural choice is reflected in any way on the restriction to the concepts being learned.

An example may make the point clear. Consider the concept {4,5,8} in the environment indicated in Figure 1.1. Denoting the "natural" property "number of borders" by x and "number of figures" by y, this concept is represented by the set of vectors {(2,1), (2,2), (3,2)}. The reader may convince himself that this set is not linearly separable. However, if new features z and w are defined as follows

$$z = 2 \text{ if } x = 3,\ y = 1$$

$$z = 1 \text{ if } x = 2,\ y = 2$$

$$z = y \text{ otherwise}$$

$$w = 2 \text{ if } x = 3,\ y = 1$$

$$w = 3 \text{ if } x = 2,\ y = 2$$

$$w = x \text{ otherwise}$$

the concept appears as the set of vectors {(2,1), (3,1), (3,2)} which is separated by the linear polynomial $w - z - \frac{1}{2}$. A very

unnatural numerical measure turns out to be the useful one as far as simplicity of expression is concerned.

An alternative mode of feature extraction may be indicated by pointing out that the "features" z and w may well have been looked upon as functions of x and y which rendered the concept separable. The form of the function $z(x,y)$ is seen to be quite complicated. Hence, if a non-linear discriminating function has to be constructed by replacing z and w by complex non-linear functions of x and y in $w - z - \frac{1}{2}$, all semblance of simplicity would be lost. However, if one is faced with a large number of highly complex discriminants for a large number of concepts in an environment of n dimensional vectors $(x_1, x_2, \ldots, x_n)$ and discovers a set of transformations

$$y_i = y_i\ (x_1, \ldots, x_n)(1 \leq i \leq n)$$

such that the original discriminants are simple functions of the y_i, one may consider the y's as the significant features of the environment and use them for subsequent learning and generalization.

In the absence of good measurements (good "features" in the general case) concept formation is an essential adjunct to Pattern Recognition, no matter how sophisticated may be the modes of combination of the basic predicates. As has been said before, past experience has shown that threshold gates are in no way more effective than Boolean gates, if the features are not good. On the other hand, recent work has shown that with

good features, quite economical switching circuits (with a very manageable number of gates) suffice for recognition.[60]

6. Learning Games by Generalization - Importance of Description Languages

It was shown in Section 9 of Chapter III that the sets $\{W_i'\}$ acted as adequate approximations to the evaluations of any Tic-Tac-Toe-like game. It might be noticed that all the basic predicates involved in the description of the sets $\{W_i'\}$ are the same as those involved in the description of Tic-Tac-Toe-like games. Hence, the Universes and fine structure families of properties needed for describing the rules of the games are adequate for the description of the $\{W_i'\}$. However, if in addition to the basic predicates one also used the derived predicates $\#_s(A) = i$ and $(\exists n)((n,A) \in C_s \ \& \ (n,B) \in C_s)$, then the description of the $\{W_i'\}$ becomes much simpler.

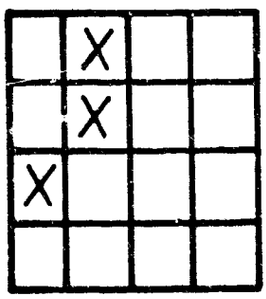
However, one important point was not made adequately in the previous discussion: that learning descriptions of the $\{W_i'\}$ as combinations of predicates of the above form leads to correct generalization with very little data.

The point is probably best illustrated by an example. Consider any plane (horizontal, vertical or diagonal) in the cubic board with three cells assigned to X and the rest by $\wedge$, as shown in Figure 5.2(a). For convenience, the cell assigned to $\wedge$ have been shown empty. That any configuration in qubic which contains a plane like this and it is the players move is in W_6' can be easily seen by the persual of Figure 5.2(b). Here each X_i represents the $i^{\underline{th}}$ move of the player and each Y_i that of the opponent. That this is also a member of K_6 is seen by

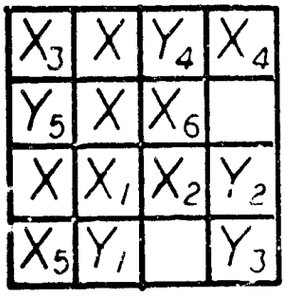
a persual of Figure 5.2(c) and the accompanying intersection matrix (which can be seen to be merely an alternative representation of a weighted graph). The intersection nodes here have been numbered to bring out the reason for K_i and W_i' being the same set.

It will be noted that the intersection matrix of Figure 5.2(c) also describes other members of K_6. Some of these are shown in Figure 5.3. Also any position equivalent to one of these under the multifarious symmetries of the qubic board,[61] would have the same matrix. Also, a plane with some extra Y's (for instance, with one between Y_2 and X_4 in Figure 5.2) would have identical descriptions. So would a configuration which have the same intersection matrix but with some of the lines in a different plane (and all their symmetrical equivalents). This mode of description (the intersection matrix is merely a convenient representation of the statement forms discussed before) is thus, more powerful than storing specific positions and considering the symmetries of the board. This latter method has been a favorite in the field and both Citrenbaum and Koffman have been erroneously criticized for not using this less efficient method which is applicable only to Qubic. The number of symmetries in games like Bridg-it and Go-Moku are far fewer but the description shown here remains equally general.

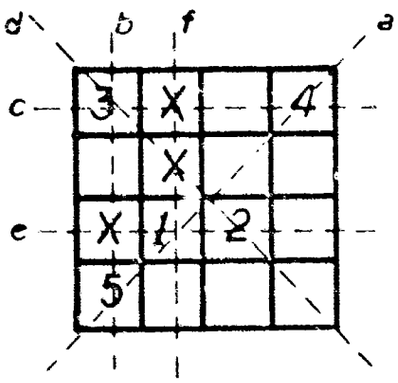
This is also a convenient place to point out that if in Figure 5.2 there was an extra Y between Y_1 and Y_3, the resulting position could have the same connection matrix but



(a)



(b)



intersects with

#	line	a	b	c	d	e	f
4	a						
3	b	5					
3	c	4	3				
3	d		3	3			
3	e	1			2		
2	f	1				1	

(c)

Figure 5.2

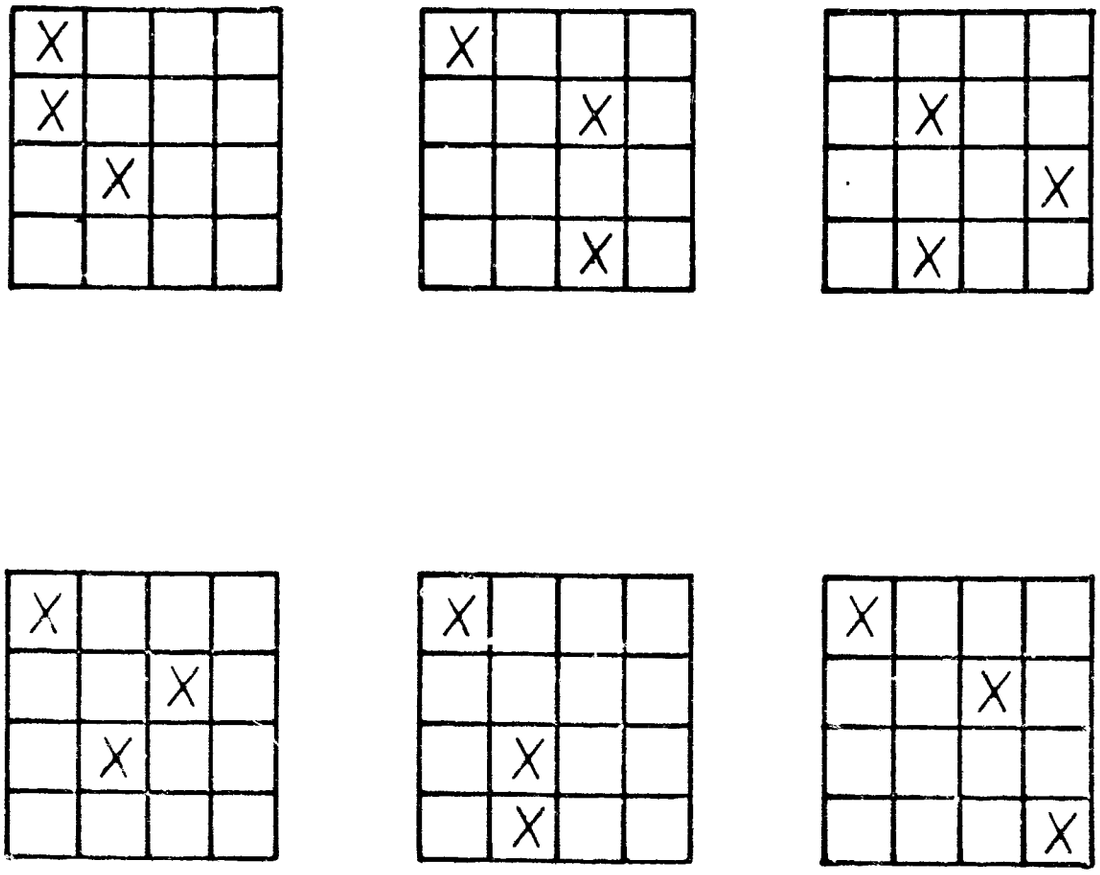


Figure 5.3

would not be a member of W_6. This indicates how some members of $\bigcup W_i'$ are <u>not</u> members of $\bigcup W_i$.

Koffman's learning program is designed to learn descriptions of $\{W_i'\}$ when their elements occur in the course of a play of the game. Here generalization is very much facilitated by an analysis of the actual course of a game. The program carries out the analysis as follows.

The first game is played at random by the program till it is defeated (or accidentally wins). The winning move is now removed and the file uncovered by the process is stored in the trivial matrix as a description of W_1'. From then on, no win against the program is possible with a single threat. When the machine is defeated (or accidentally wins) by a "fork," removal of the winning move reveals a position whose description was already available in memory. At this point the previous move is removed, and the line uncovered together with its intersection with the lines satisfying the previous description is stored as a new matrix. Now the program blocks all forks and initiates its own forks when possible. Learning continues in subsequent defeats and accidental wins by deeper forks in a similar fashion. Successive previous moves are removed till no match is found with previous descriptions. The last move removed is then analysed to reveal the alternative threat that was blocked. The lines of this threat and that of the previous one, together with their intersection pattern, is then stored in a new matrix.

It ought to be pointed out that in the absence of this kind of analysis descriptions of the $\{W_i'\}$ could be learned as conjunctions of statements from a large number of examples by some algorithm analogous to Pennypacker's. However, the above analysis leads to a more rapid learning. As a result, Koffman's program needs to play only about 12 games before it defeats its opponent 50% of the time in Qubic and Go-Moku and wins in Bridg-it every time it plays first.

REFERENCES

1. E. A. Feigenbaum, The Simulation of Verbal Learning Behavior, Proceedings of the Western Joint Computer Conference, 19(1961), 121.

2. F. M. Tonge, Summary of a Heuristic Line Balancing Procedure, Management Science, 7(1960), 21.

3. L. R. Marino, Winning and Non-Losing Stretegies in Games and Control, Report No. SRC 91-A-66-36, Case Institute of Technology, Cleveland, Ohio, 1966.

4. R. E. Bellman and S. E. Dreyfus, Applied Dynamic Programming, Princeton University Press, Princeton, N.J., 1962.

5. M. D. Mesarovic, Toward A Formal Theory of Problem Solving, Computer Augmentation of Human Reasoning (Sass and Wilkinson, Ed.), Spartan Books, Washington, D.C., 1965, 37.

6. T. G. Windeknecht, Problem Solving and Finite Automata, A Status Report on Research in Artificial Intelligence and Linguistics, Case Institute of Technology, Cleveland, Ohio, 1964.

7. G. W. Ernst and A. Newell, Some Issues of Representation in a General Problem Solver, AFIPS Conference Proceedings, 30(1967), 583.

8. J. Von Neumann and O. Morgenstern, Theory of Games and Economic Behavior, Princeton University Press, Princeton, N.J., 1947.

9. A. L. Samuel, Some Studies in Machine Learning Using the Game of Checkers, IBM Journal of Research and Development, 3(1959), 210.

10. A. Newell and H. A. Simon, The Logic Theory Machine, IRE Transactions on Information Theory, IT-2(1956), 61.

11. W. W. R. Ball, Mathematical Recreations and Essays, McMillan and Co., Ltd., London, 1940, 303.

12. A. Hormann, Programs for Machine Learning, Part II, Information and Control, 7(1964), 55.

13. P. M. Cohn, Universal Algebra, Harper & Row, New York, 1965.

14. A. Newell, J. C. Shaw and H. Simon, Report on a General Problem Solving Program, Proceedings of the International Conference on Information Processing, UNESCO, Paris, 1959, 256.

15. S. Amarel, On Machine Representation of Problems and Reasoning About Actions - The Missionaries & Cannibal Problem, RCA Laboratories, Princeton, N.J., 1966.

16. A. Church, Introduction to Mathematical Logic, Princeton University Press, Princeton, N.J., 1956.

17. S. T. Hu, Threshold Logic, University of California Press, Berkeley, 1965.

18. R. L. Ashenhurst, The Decomposition of Switching Functions, Proceedings of the International Symposium on the Theory of Switching, Harvard University Press, Cambridge, Massachusetts, 1959.

19. J. S. Bruner, J. J. Goodnow and G. A. Austin, A Study of Thinking, John Wiley & Sons, New York, 1956.

20. I. H. Sublette, Recognition of Class Membership by Means of Weak, Statistically Dependent Features, Radio Corporation of America, Princeton, N.J., 1966.

21. A. Lehman, A Solution for the Shannon Switching Game, U.S. Army Mathematical Research Center Tech. Summary Report 308; July 1962.

22. R. G. Busacker and T. L. Saasy, Finite Graphs and Networks, McGraw Hill Book Co., New York, 1965.

23. C. Berge, The Theory of Graphs and its Applications, John Wiley & Sons, New York, 1962.

24. J. Hartmanis and R. E. Stearns, Algebraic Structure Theory of Sequential Machines, Prentice Hall, Englewood Cliffs, N.J., 1966.

25. F. Rosenblatt, Perceptron Experiments, Proceedings of the IRE, 48(1960), 301.

26. S. Ginsburg, The Mathematical Theory of Context-Free Languages, McGraw Hill, New York, 1966.

27. R. B. Banerji, An Information Processing Program for Object Recognition, General Systems, 5(1960), 117.

28. R. B. Banerji, Computer Programs for the Generation of New Concepts from Old Ones, Neure Ergebrisse der Kybernetik, Oldenbourg Verlog, Munich, 1964.

29. E. B. Hunt and C. I. Hovland, Programming a Model of Human Concept Formulation, Proceedings of the Western Joint Computer Conference, Los Angeles, 1961.

30. E. B. Hunt, J. Marin and P. J. Stone, Experiments in Induction, Academic Press, New York, 1966.

31. E. A. Feigenbaum, The Simulation of Verbal Learning Behavior, Proceedings of the Western Joint Computer Conference, Los Angeles, 1961.

32. J. C. Pennypacker, An Elementary Information Processor for Object Recognition, Report No. SRC 30-I-63-1, Case Institute of Technology, Cleveland, Ohio, 1963.

33. T. G. Windeknecht, A Theory of Simple Concepts with Applications, Report No. SRC 53-A-64-19, Case Institute of Technology, Cleveland, Ohio, 1964.

34. J. Snediker, A Generalized Program for Information Retrieval, Report No. SRC 77-A-65-29, Case Institute of Technology, Cleveland, Ohio, 1965.

35. P. Z. Ingerman, A Syntax Oriented Translator, Academic Press, New York, 1966.

36. G. S. Sebestyen, Pattern Recognition by an Adaptive Process of Sample-Set Construction, IRE Transactions on Information Theory, IT-8(1962), S-82.

37. W. H. Highleyman, Linear Decision Functions, with Application to Pattern Recognition, Proceedings of the IRE, 50(1962), 1501.

38. M. A. Aizerman, Lernvorgänge, bei der Erkennung von Zeichenklassen, Neure Ergebrisse der Kybernetik, Oldenbourg Verlog, Munich, 1964.

39. E. B. Hunt, J. Marin and P. J. Stone, Experiments in Induction, Academic Press, New York, 1966.

40. E. Feigenbaum, An Information Processing Theory of Verbal Learning, Report No P-1817, October, The RAND Corporation, Santa Monica, California, 1959.

41. J. A. Robinson, A Machine Oriented Logic Based on the Resolution Principle, Journal of the Association for Computing Machinery, 12(1965), 23.

42. J. A. Robinson, The Mechanisation of Theorem Proving, Systems and Computer Science (Hart & Takasu, Ed.), University of Toronto Press, 1967.

43. E. C. Milliken, A Language for Class Description and its Processor, National Conference of the Association for Computing Machinery, Cleveland, Ohio, 1965.

44. E. Mendelson, Introduction to Mathematical Logic, Van Nostrand, New York, 1964.

45. J. L. Kuhns, An Application of Logical Porbability to Problems in Automatic Abstracting and Information Retrieval, First Congress on the Information System Sciences, Hot Springs, Virginia, 1962.

46. B. Raphael, SIR: A Computer Program that Understands, Fall Joint Computer Conference, 1964.

47. R. Narasimham, Syntactic Descriptions of Pictures and Gestalt Phenomena of Visual Perception, Report dated July 25, 1963 from Digital Computer Laboratory, University of Illinois, Urbana, Illinois.

48. R. A. Kirsch, Computer Interpretation of English Text and Picture Patterns, Transactions of the IEEE, EC-13(1964), 363.

49. B. Raphael, SIR: A Computer Program for Information Retrieval, Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1964.

50. G. S. Sebestyen, Decision Making Processes in Pattern Recognition, MacMillan & Co., New York, 1962.

51. G. W. Ernst and R. Sherman, Learning Concepts in Terms of Other Concepts, Conference of the IFIPS (communicated), 1968.

52. D. F. Specht, Generation of Polynomial Discriminant Functions for Pattern Recognition, IEEE Transactions on Electronic Computers, EC-16(1967), 308.

53. Y. C. Ho and R. L. Kashyap, An Algorithm for Linear Inequalities and its Applications, IEEE Transactions on Electronic Computers, EC-14(1965), 683.

54. B. Widrow and F. W. Smith, Pattern Recognizing Control Systems, Computer and Information Sciences (Tou & Wilcox, Ed.), Spartan Books, Washington, D.C., 1964.

55. A. Novikoff, On Convergence Proofs for Perceptrons, Proceedings of the Symposium on Mathematical Theory of Automata, Brooklyn Polytechnic Institute, 1963.

56. L. C. Barbosa and E. Wang, On a Class of Iterative Algorithms for Linear Inequalities with Application to Pattern Classification, Princeton Conference on Information Sciences and Systems, 1967.

57. L. Uhr and C. Vossler, A Pattern Recognition Program that Generates, Evaluates and Adjusts its Own Operators, Western Joint Computer Conference, Los Angeles, 1961.

58. H. D. Block, N. J. Nilsson and R. O. Duda, Determination and Direction of Features in Patterns, Computer and Information Sciences (Tou & Wilcox, Ed.) Spartan Books, Washington, D.C., 1964.

59. C. Newman and L. Uhr, BOGART: A Discovery and Induction Program for Games, 20th National Conference of the Association for Computing Machinery, 1965.

60. J. G. Simek and C. J. Tunis, Handprinting Input Device for Computer Systems, IEEE Spectrum, 4(1967), 72.

61. R. Silver, The Group of Automorphisms of the Game of 3-dimensional Tic-Tac-Toe, American Mathematical Monthly, 74(1967), 247.

62. R. L. Citrenbaum, The Concept of Strategy and its Application to 3-Dimension Tic-Tac-Toe, Report No. SRC 72-A-65-26, Case Institute of Technology, Cleveland, Ohio, 1965.

63. S. Amarel, On the Automatic Formation of a Computer Program which Represents a Theory, Self Organizing System, (Yovits & Cameron, Ed.), Spartan Books, Washington, D.C, 1962.

64. S. Amarel, Problem Solving Procedures for Efficient Syntactic Analysis, 20th National Conference of the ACM, Cleveland, Ohio, 1965.

65. S. Amarel, An Approach to Heuristic Problem Solving and Theorem Proving in the Propositional Calculus, Systems and Computer Science (Hart & Takasu, Ed.), University of Toronto Press, 1967.

66. R. Bellman and P. Brock, On the Concept of a Problem and Problem Solving, American Mathematical Monthly, 67(1960), 119.

67. R. Bellman, Dynamic Programming, Intelligent Machines and Self Organizing Systems, Mathematical Theory of Automata, Polytechnic Institute of Brooklyn, New York, 1963.

68. R. Bellman, Mathematical Model Making A an Adaptive Process, Report No. P-2400, The RAND Corporation, 1961.

69. E. G. Koffman, Learning Through Pattern Recognition Applied to a Class of Games, Report No. SRC 107-A-67-45, Case Western Reserve University, Cleveland, Ohio, 1967.

APPENDIX

9. Approximation to Strategies in Tic-Tac-Toe-Like Games

Tic-Tac-Toe-like games have already been discussed in Section 5. In the present section certain subsets $\{W_i'\}$ of S (the set of situations) will be discussed which contain the sets $\{W_i\}$ although they do not coincide with the sets $\{W_i\}$. In what follows, the definitions for W_i' will be introduced. It will be shown in Chapter V that descriptions of $\{W_i'\}$ are much easier to learn than those of $\{W_i\}$. The significance of this learning will be clarified in Chapter V.

It will be recalled that a Tic-Tac-Toe-like game is completely specified by a set N of cells and two subsets, $\mathcal{G}$ and $\mathcal{B}$, of 2^N, called the winning and losing files. Given a game $\langle N,\mathcal{G},\mathcal{B}\rangle$ one can define a reduced game $\langle N,\mathcal{G},\emptyset\rangle$, with the same set of cells and winning files, but no losing files. The evaluations of $\langle N,\mathcal{G},\mathcal{B}\rangle$ will be denoted by $\{W_i\}$ and those of $\langle N,\mathcal{G},\emptyset\rangle$ by $\{W_i'\}$. Similarly, the situations to which (n,X) or (n,Y) are applicable will be denoted by $S_{(n,X)}$ and $S_{(n,Y)}$ as before for $\langle N,\mathcal{G},\mathcal{B}\rangle$ and by $S'_{(n,X)}$ and $S'_{(n,Y)}$ for $\langle N,\mathcal{G},\emptyset\rangle$.

The following theorem indicates how the sets $\{W_i'\}$ act as approximations to W_i.

<u>Theorem 3.22:</u> If $s \in W_i$ then $s \in \bigcup_j^i W_j'$.

<u>Proof:</u> If $s \in W_i$ then there is an $n \in N$ such that $s \in S_{(n,X)}$ and $s_1 = (n,X)(s) \in W$. However, $s \in S_{(n,X)}$ implies that $s \in S-L$, $|s^{-1}(X)| = |s^{-1}(Y)|$, and $s(n) = \Lambda$. Since L is empty in $\langle N,\mathcal{G},\emptyset\rangle$, this also implies that $s \in S'_{(n,X)}$. Also,

$s_1 = (n,X)(s) \in W_1$ implies that $s_1^{-1}(X) = A$ for some $A \in \mathcal{G}$ and if no $B \in \mathcal{B}$, $s_1^{-1}(Y) \supseteq \mathcal{B}$. Again, since $\mathcal{B}$ is empty in $\langle N, \mathcal{G}, \emptyset \rangle$, this implies $(n,X)(s) \in W$ for $\langle N, \mathcal{G}, \emptyset \rangle$ also. Hence, $W_1 \subseteq W_1'$. The theorem is thus true for $i = 1$.

Let the theorem be true for $i = k$. Let $s \in W_{k+1}$. If $s \in \bigcup_{j=1}^{k} W_j'$, there is nothing to prove. Otherwise recall that there exists an $n \in N$ such that $s \in S_{(n,X)}$ and for each n' such that $(n,X)(s) \in S_{(n',Y)}$, $(n',Y)((n,X)(s)) \in \bigcup_{j=1}^{k} W_j \subseteq \bigcup_{j=1}^{k} W_j'$. Since $S'_{(n,X)} \supseteq S_{(n,X)}$ as proved before and since $S'_{(n',Y)} \supseteq S_{(n',Y)}$ can be proved similarly, this implies that $s \in W_{k+1}'$. This proves the theorem.

In what follows, elements of $\bigcup W_i'$ will be given an alternative description which will be easier to test than the exhaustive trial indicated by the definitions in Section 6 used so far. For this, the following ideas will have to be introduced.

Let α and β be two arbitrary sets, let $C \subseteq \alpha \times \beta$ and let $\#$ be a function mapping the range of the relation C into integers. Then the pair $\langle C, \# \rangle$ will be called a <u>weighted graph</u> on α and β.

Given a situation s in a Tic-Tac-Toe-like game, let $\langle C_s, \#_s \rangle$ be the weighted graph on N and $\mathcal{G}$ defined as follows:

(i) $(n,A) \in C_s$ if and only if $s(n) = A$, $n \in A$ and $s^{-1}(Y) \cap A = \emptyset$

(ii) $\#_s(A) = |A \cap s^{-1}(A)|$

The ideas involved here may perhaps be illustrated by reference to Figure 3.6, showing some situations in a 3 x 3 Tic-Tac-Toe game. If the cells are called 1 to 9 in the usual order then the set of all files are (1,2,3), (4,5,6), (7,8,9), (1,4,7), (2,5,8), (3,6,9), (1,5,9) and (3,5,7). Calling these a to h respectively, the weighted graph of the board shown in Figure 3.6(a) is

$$C = \{(7,c),(7,h),(2,e),(3,h),(3,g),(6,f),(8,e),(8,c)\}$$

$$\#(c) = 2;\ \#(e) = 2;\ \#(f) = 2;\ \#(h) = 2.$$

C and # are represented in Figure 3.6(b) in a graphical form.

For an understanding of what follows it would be worthwhile to indicate what happens to the graph $\langle C_s, \#_s \rangle$ as the situation changes as a result of applying controls and distrubances ("moves" and "countermoves"). Two steps of change are indicated in Figures 3.6(c), (d), (e) and (f). The effects indicated in these pictures can be formalised as follows:

For each element n of α, let X_n and Y_n be two functions from weighted graphs to weighted graphs defined as follows:

$$X_n\ (\langle C, \# \rangle) = \langle C', \#' \rangle$$

where $\quad C' = [(\alpha - \{n\}) \times \beta] \cap C$

and $\quad \#'(A) = \#(A)$ if $(n,A) \notin C$

$\qquad\qquad\quad \#(A) - 1$ if $(n,A) \in C$

$$Y_n\ (\langle C, \# \rangle) = \langle C', \#' \rangle$$

where $\quad C' = [(\alpha - \{n\}) \times (\beta - C(n))] \cap C$

$\#'(A) = \#(A)$ for all elements of the range of C'.

<u>Theorem 3.23:</u> In any Tic-Tac-Toe-like game and any situation s

$$\langle C_{(n,X)(s)}, \#_{(n,X)(s)}\rangle = X_n(\langle C_s, \#_s\rangle)$$

$$\langle C_{(n,Y)(s)}, \#_{(n,Y)(s)}\rangle = Y_n(\langle C_s, \#_s\rangle)$$

whenever the left-hand sides are defined.

<u>Proof</u> Let $(n,X)(s)$ be defined, i.e., let $s \in S_{(n,X)}$. Then $s(n) = \Lambda$. If $(n,X)(s) = s_1$, then $s_1(n) = X$ and for all $m \in N$, $m \neq n$ implies $s_1(n) = s(n)$. Hence, $(m,A) \in C_{s_1}$ if and only if $(m,A) \in C_s$ and $m \neq n$. Also, for any A in the domain of $C_{s'}$, $|A \cap s_1^{-1}(\Lambda)| = |A \cap s^{-1}(\Lambda)|$ unless $n \in A$, i.e., if $(n,A) \in C$, in which case $|A \cap s_1^{-1}(\Lambda)| = |A \cap S^{-1}(\Lambda)| - 1$. This proves the first part of the theorem. The proof of the second part is left to the reader.

For an alternative description of the sets $\{W_i'\}$ one has to define the following class $\{J_i\}$ of sets of weighted graphs.

A weighted graph $\langle C,\#\rangle$ belongs to J_1 if and only if there is an A in the range of C such that $\#(A) = 1$.

K_i is defined for $i > 1$ as follows:

Let $\langle C',\#'\rangle$ be any graph which is a member of $\bigcup_{j=1}^{i} J_j$ and such that for all n it is true that $Y_n(C',\#') \in \bigcup_{j=1}^{i} J_j$.

Let $\{C_1, C_2, \ldots, C_n\}$ be the set of all subgraphs of C' such

that for each $p(1 \le p \le n > (C_p, \#_p)$ is a member of $\bigcup_{j=1}^{i} J_j$ ($\#_p$ is the restriction $\#'$ to the range of C_p). Let $(A_1, A_2, \ldots, A_m)$ be a set of elements in the range of C' such that there is at least one $A_q (1 \le q \le m)$ in the range of each $C_p (1 \le p \le i)$. Let n be any element of α not in the domain of C'. Let <C",#"> be constructed as follows:

$$C'' = C' \cup \{(n,A_1),(n,A_2),\ldots(n,A_m)\}$$

$$\#''(A) = \#'(A) + 1 \text{ if } A = A_q \ (1 \le q \le m)$$

$$\#'(A) \text{ otherwise.}$$

A weighted graph belongs to J_{i+1} if and only if it does not belong to $\bigcup_{j=1}^{i} J_j$ but has <C",#"> above as a subgraph.

<u>Theorem 3.24:</u> If a graph <C,#> belongs to J_{i+1} there exists an n in the domain of C such that for all n',

$$Y_{n'} (X_n(\langle C,\#\rangle)) \in \bigcup_{j=1}^{i} J_j$$

<u>Proof:</u> By definition of J_{i+1}, <C,#> has no subgraph belonging to $\bigcup_{j=1}^{i} J_i$. Also, there is a graph <C',#'> and a subgraph <C",#"> of <C,#> such that <C",#"> is constructed from <C',#'> as described in the definition of J_{i+1}. Let n be a member of α which occurs in C" but not in C'. From construction of <C",#"> it is evident that

$$X_n(C'',\#'') = \langle C',\#'\rangle$$

Since (C",#") is a subgraph of <C,#>, <C',#'> is a subgraph

of $X_n(\langle C,\#\rangle)$. Since for all n', $Y_{n'}(\langle C',\#'\rangle) \in \bigcup_{j=1}^{i} J_i$,

and $\langle C',\#'\rangle$ is a subgraph of $X_n(\langle C,\#\rangle)$, for all n',

$Y_{n'}(X_n\langle C,\#\rangle) \in \bigcup_{j=1}^{i} J_j$

<u>Theorem 3.25:</u> For any Tic-Tac-Toe-like game, $s \in W_k'$ if and only if $|s^{-1}(X)| = |s^{-1}(Y)|$, $s^{-1}(X) \not\supset A$ for any $A \in G$ and $\langle C_s, \#_s\rangle \in J_k$.

<u>Proof:</u> Let k = 1.

If $\langle C_s, \#_s\rangle \in J_1$, then there exists an $A \in G$ such that $|s^{-1}(\wedge) \cap A| = 1$ and $s^{-1}(Y) \cap A = \emptyset$, i.e., for all cells in A except one $s(m) = X$, and for one cell $n \in A$, $s(n) = \wedge$. Since $|s^{-1}(X)| = |s^{-1}(Y)|$ and $s^{-1}(X) \not\supset A'$ for any $A' \in G$, $s \in S_{(n,X)}$. Also, if $(n,X)(s) = s_1$, then $s_1(n) = X$ and $s_1(m) = s(m) = X$ for all cells of A. Thus $s_1^{-1}(X) \supset A$ and $(n,X)(s) \in W$. Hence, $s \in W_1'$.

Let now $s \in W_1'$, so that there exists an $n \in N$ such that $(n,X)(s) \in W$. Let $(n,X)(s) = s_1$. Now $s_1^{-1}(X) = s^{-1}(X) \cup \{n\}$. Since $s_1 \in W$, there is an $A \in G$ such that $s^{-1}(X) \cup \{n\} \supset A$. However, since $s \notin W$, $s^{-1}(X) \not\supset A$. Hence, $n \in A$ and for all $m \in A$ such that $m \neq n$, $s(m) = X$. Hence, $\#_s(A) = 1$ and $s \in J_1$.

Let now the theorem be true for $k \leq i$. Let $\langle C_s, \#_s\rangle \in J_{i+1}$. Then by Theorem 3.24 there exists an n such that for all n'

$$Y_{n'}(X_n(\langle C_s, \#_s\rangle)) \in \bigcup_{j=1}^{i} J_j$$

However, since $s \notin W$ and $|s^{-1}(X)| = |s^{-1}(Y)|$, and from the proof of Theorem 3.25 of $\langle C_s, \#_s \rangle$ as a member of J_{i+1}, there is an $n \in N$ such that $s(n) = \Lambda$, $s \in S_{(n,X)}$ and hence, $(n,X)(s)$ is defined. Also, whenever $(n',Y)((n,X)(s))$ is defined, one has by Theorem 3.24

$$\langle C_{(n',Y)((n,X)(s))}, \#_{(n',Y)((n,X)(s))} \rangle$$

$$= Y_{n'}(X_n(C_S, \#_S)) \in \bigcup_{j=1}^{i} J_j$$

hence, there exists an n such that $s \in S_{(n,X)}$ and for all n' such that $(n,X)(s) \in S_{(n',Y)}$,

$$(n',Y)((n,X)(s)) \in \bigcup_{j=1}^{i} W_j$$

Hence,

$$s \in W_{i+1}.$$

Let now $s \in W_{i+1}$. Then, there exists an n such that $s \in S_{(n,X)}$ and for all n' such that $(n,X)(s) \in S_{(n',Y)}$

$$(n',Y)((n,X)(s)) \in \bigcup_{j=1}^{i} W_i$$

and hence,

$$Y_{n'}(X_n(\langle C_s, \#_s \rangle)) \in \bigcup_{j=1}^{i} K_i$$

Since $Y_{n'}(X_n(\langle C_S, \#_S \rangle))$ is a subgraph of $Y_n(\langle C_S, \#_S \rangle)$ by definition of $Y_{n'}$, $X_n(\langle C_S, \#_S \rangle)$ has subgraphs $\{\langle C_k, \#_k \rangle\}$ $\in \bigcup_{j=1}^{i} K_i$. Also, none of $\{\langle C_k, \#_k \rangle\}$ are subgraphs of

$\langle C_s, \#_s \rangle$, since $\langle C_S, \#_S \rangle \notin \bigcup_{j=1}^{i} K_i$ by induction hypotheses. Hence, n must occur in the domain of each of these subgraphs. So $\langle C_s, \#_s \rangle$ has a subgraph which is obtained from $\{\langle C_k, \#_k \rangle\}$ by the construction shown in the definition of K_{i+1}. Hence, $s \in K_{i+1}$.

The reason for introducing the above theorems is the fact that the only predicates needed for the recognition of members of $\bigcup K_i$ are the values of $\#_s$ for the different files and (in view of the construction of K_{i+1} from $\bigcup_{j=1}^{i} K_j$) the fact that $s(n) = \wedge$ for some cell n common to a number of files. Given a situation $s \in \{X, Y, \wedge\}^N$, i.e., an assignment of X, Y and $\wedge$ on the cells, the search for files with given values of $\#_s$ and having certain cells in $s^{-1}(\wedge)$ in common between files is much more directed than the exhaustive mini-max searches indicated by the definition of $\{W_i\}$.

The difficulty with the description of W_i through the K_i, however, lies in the fact that the $\bigcup W'_i$ contains $\bigcup W_i$, but does not coincide with it. Hence, the K_i are only approximations to W_i. The reason for this is that the converse of Theorem 3.22 is not true. One reason for this, in turn, is that elements W'_i may be elements of L in $\langle N, \mathcal{G}, \mathcal{S} \rangle$, and is a member of $\bigcup_{n \in N} S_{(n,X)}$ only because L' is empty in $\langle N, \ldots \rangle$. An example of this will be given in Chapter V, Section 6.

However, because the difference between $\{W_i\}$ and

$\{W_i'\}$ lies mainly because of the emptyness of L', a state in $\bigcup W_i'$ can be tested for membership in $\bigcup W_i$ by a somewhat well-directed search also. A method for doing this has been pointed out by Citrenbaum.[62]

Another very important reason for using the $\{K_i\}$ as approximations to the $\{W_i\}$ is that the $\{K_i\}$, being obtainable from a specific mode of combination of statements of the form $*_s(A) = i$ and $(\exists n)((n,A) \in C_S \ \& \ (n,B) \in C_s)$, leads to easy generalizations from examples. A learning program based on such generalization was developed by Koffman,[69] and will be discussed in Section 6 of Chapter V. The descriptions learned by this program is utilized by a game-playing program to make very deep forcing moves during the play of any Tic-Tac-Toe-like game. In this sense, the program is game-independent within this class of games. Given any game $\langle N, G, \beta \rangle$ it plays the game legally and, on the basis of its experience, improves its game - often to defeat its opponent.

CAPTIONS TO FIGURES

Figure 1.1	A Simple Universe of Discourse
Figure 2.1	The Tower of Hanoi
Figure 3.1	An Illustration of a Pathological Game
Figure 3.2	The 4^3 Tic-Tac-Toe Game
Figure 3.3	The Shannon Game for Bridg-it
Figure 3.4	A Winning Situation in 7x7 Hex
Figure 3.5	Grundy-Function of the 7-stick NIM
Figure 3.6	Playing a 4-pile NIM Game
Figure 3.7	Some Weighted Graphs for Tic-Tac-Toe
Figure 4.1	Illustration of Non-input Properties
Figure 4.2	Detailed Structure of "$X \subseteq C$?" (two sheets)
Figure 4.2(a)	Detailed Algorithm for "Is S D-true?" (two sheets)
Figure 4.3(b)	Skeleton of "Evaluate Term m"
Figure 4.3(c)	Details of "Evaluate A"
Figure 5.1	Pennypacker's Learning Algorithm (two sheets)
Figure 5.2	A Member of K_6 in 4^3 Tic-Tac-Toe and its Description
Figure 5.3	Some More Members of K_6 with same Description as Figure 5.2

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Systems Research Center Case Western Reserve University Cleveland, Ohio 44106	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

SOME RESULTS IN A THEORY OF PROBLEM SOLVING (PART II)

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
scientific; interim

5. AUTHOR(S) (First name, middle initial, last name)

R. B. Banerji

6. REPORT DATE November 1967	7a. TOTAL NO. OF PAGES 156	7b. NO. OF REFS 69
8a. CONTRACT OR GRANT NO. AF-AFOSR-67-125A b. PROJECT NO. 9769-05 c. 61102F d. 681304	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) AFOSR 68-2083	

10. DISTRIBUTION STATEMENT

1. This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES TECH OTHER	12. SPONSORING MILITARY ACTIVITY Air Force Office of Scientific Research Directorate of Information Sciences Arlington, Virginia 22209

13. ABSTRACT

This report considers the effect that specific language descriptions have on the efficiency of pattern recognition and problem solving methods. The efficiency of a language for the description of a given set is viewed in terms of the size, in some sense, of the shortest expression that denotes the set. Central to the discussion are questions of how the description of a concept should be stored to use the smallest amount of memory, and how the description should be stored and processed so that, given an object and a concept, an efficient determination can be made on containment of the object in the concept. The languages are essentially non-numeric, enabling pre-processing to be described in the same format as that used for pattern description. Algorithms for learning and generalization are presented that use two of the languages. The property of succinctness is considered for the algorithms, and the effect of lack of succinctness on the statistical degree of confidence in the learned description is indicated. Analogies are made to descriptions in terms of discriminant functions and maximum likelihood ratios.

DD FORM 1473