Workshop on Current Developments
in Machine Learning, CMU, Pittsburgh,
July 16-18, 1980

80-7

INDUCTIVE LEARNING AS RULE-GUIDED GENERALIZATION AND CONCEPTUAL SIMPLIFICATION OF SYMBOLIC DESCRIPTIONS: Unifying principles and a methodology

R. S. Michalski
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

ABSTRACT

A theoretical framework is presented which treats inductive learning as a process of generalizing and simplifying symbolic descriptions, under a guidance of *generalization rules* (representing inference processes which generalize descriptions) and *problem environment rules* (representing problem dependent knowledge). This approach unifies various types of inductive learning, such as learning from examples (determination of characteristic or descriminant descriptions, and sequence prediction), and learning from observation (revealing a conceptual structure underlying an arbitrary collection of entities).

A brief description is given of two inductive learning programs INDUCE 2 --- for learning characteristic or discriminant structural descriptions, and CLUSTER/PAF --- for learning from observation ('conceptual clustering'). The latter program determines a taxonomic description, which partitions a given collection of entities into clusters, such that each cluster is described by a single conjunction of relational statements and the obtained assembly of clusters satisfies an assumed criterion of preference.

The presented methodology can be useful for automated 'conceptual' analysis of experimental data, for searching for patterns and abstracting the contents of databases, and also for aiding the knowledge acquisition processes in the development of expert systems.

INDUCTIVE LEARNING AS RULE-GUIDED GENERALIZATION AND CONCEPTUAL SIMPLIFICATION OF SYMBOLIC DESCRIPTIONS
Unifying Principles and a methodology

R. S. Michalski
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

INTRODUCTION

Our understanding of inductive learning processes remains very limited despite considerable progress in recent years. Making progress in this area is particularly difficult, not only because of the intrinsic complexity of these problems, but also because of their open-endedness. This open-endedness implies that when we make inductive assertions about some piece of reality, there is no natural limit to the level of detail of descriptions of this reality, to the scope of concepts and operators used in the expression of these assertions, or to the richness of their forms. Consequently, in order to achieve non-trivial general solutions, one has to circumscribe carefully the nature and goals of the research. This includes defining the language in which descriptions may be written and the modes of inference which will be used. Careful definitions will avoid the main difficulty of most current research: attacking problems which are too general with techniques which are too limited.

Recently there has been a growing need for practical solutions in the area of inductive learning. For example, the development of knowledge-based expert systems requires efficient methods for acquiring and refining knowledge. Currently, the only method of knowledge acquisition is the handcrafting of an expert's knowledge in some knowledge representation system, e.g., production rules (Shortliffe [1], Davis [2]) or a semantic net (Brachman [3]). Progress in the theory of induction and the development of efficient inductive programs can provide valuable assistance and an alternative method in this area. For example, inductive programs could be useful for filling in gaps and testing the consistency and completeness of expert-derived decision rules, for removing redundancies, or for incremental improvement of the rules through the analysis of their performance. They could provide a means for detecting regularities in data bases and knowledge bases. Also, for appropriately selected problems, the programs could determine the decision rules directly from examples of expert decisions, which would greatly facilitate the transfer of knowledge from experts into machines. Experiments on the acquisition of rules for the diagnosis of soybean diseases (Michalski and Chilausky [4]) have indicated that rule-learning from examples is not only feasible, but in certain aspects it may be even preferable.

Another potential for applying inductive programs is in various areas of science, e.g., biology, microbiology, and genetics. Here they could assist a scientist in revealing structure or detecting interesting conceptual patterns in collections of observations or results of experiments. The traditional mathematical techniques of regression analysis, numerical taxonomy, factor analysis, and distance-based clustering techniques are not sufficiently adequate for this task. Methods of conceptual data analysis are needed, whose results are not mathematical formulas but conceptual descriptions of data, involving both qualitative and quantitative relationships.

Similar in general framework, but different in objectives is research in a sub-area of computer inductive learning such as automatic programming (e.g., Shaw, Swartout and Green [5], Jouannaud and Kodratoff [6], Burstall and Darlington [7], Biermann [8], Smith [9], Pettorossi [10]). Here, the objective is to synthesize a program from I/O pairs or computational traces, or to improve its computational efficiency by application of correctness-preserving transformation rules. The final result of learning is thus a program, in a given programming language, with its inherent sequential structure, destined for machine rather than human "consumption" (or, in other words, a description in "computer terms" rather than in "human terms"). Here, the *postulate* of *human comprehensibility* (mentioned below) is not too relevant. Quite similar to research on automatic programming is research on grammatical inference (e.g., Bierman and Feldman [11], Yau and Fu [12]) where the objective of learning is a formal grammar.

This paper is concerned with computer inductive inference, which could be called a "conceptual" induction. The final result of learning is a symbolic description of a class (or classes) of entities (which typically are not computational processes) which is in a form of a logical-type expression. Such an expression is expected to be relatively "close" to a natural language description of the same class(es) of entities, specifically it should satisfy what we call the *comprehensibility postulate:*

The results of computer induction should be conceptual descriptions of data, similar to the descriptions a human expert might produce observing the same data. They should be comprehensible by humans as single 'chunks' of information, directly interpretable in natural language, and can involve both quantitative and qualitative information.

This postulate implies that descriptions should avoid more than one level of bracketing, more than one implication or exception symbol, avoid recursion, avoid including more than 3-4 conditions in a conjunction and more than 2-3 conjunctions in a disjunction, not include more than two quantifiers, etc. (the exact numbers can be disputed, but the principle is clear). This postulate can be used to decide when to assign a name to a specific formula and use that name inside another formula. This postulate stems from the motivation of this research to provide new methods for knowledge acquisition and techniques for conceptual data analysis.

It is also well confirmed by the new role for research in artificial intelligence, as envisaged by (Michie [13]), which is to develop techniques for *conceptual interface* and *knowledge refinement*.

In this paper we will consider two basic types of inductive inference: learning from examples and learning from observation (specifically, the so called "conceptual clustering").

2. COMPUTER INDUCTION AS GENERALIZATION AND SIMPLIFICATION OF SYMBOLIC DESCRIPTIONS

2.1 Inductive Paradigm

The process of induction can be characterized as the search for an economical and correct expression of a function which is only partially known. In other words, its goal is the determination and validation of plausible general descriptions (inductive assertions or hypotheses) which explain a given body of data, and are able to predict new data. Between the two aspects of induction -- the generation of plausible inductive assertions and their validation -- only the first is the subject of our study. We feel that the subject of hypotheses generation, in particular the problems of generalization and simplification of symbolic descriptions by a computer, is a quite unexplored and very important direction of research. The problems of hypothesis confirmation, in the Carnapian (Carnap 14) or similar sense, are considered to be beyond the scope of this work. In our approach, inductive assertions are judged by a human expert interacting with the computer, and/or tested by standard statistical techniques. The research is concentrated on the following inductive paradigm:

Given is:

(a) a set of *data rules (input rules)*, which consist of *data descriptions*, $\{C_{ij}\}$, specifying initial knowledge about some entities (objects, situations, processes, etc.), and the *generalization* *class*, K_i, associated with each C_{ij} (the association is denoted by ::>):

$$\begin{array}{llll}
C_{11} ::> K_1, & C_{12} ::> K_1 & \ldots\ldots & C_{1t1} :: K_1 \\
C_{21} ::> K_2, & C_{22} ::> K_2 & \ldots\ldots & C_{2t2} :: K_2 \\
\vdots & & & \\
C_{m1} ::> K_m, & C_{m2} ::> K_m & \ldots\ldots & C_{mtm} ::> K_m
\end{array} \qquad (1)$$

Descriptions C_{ij} can be symbolic specifications of conditions which given situations satisfy, production rules, sequences of attribute-value pairs representing observations or results of experiments, etc. The descriptions are assumed to be expressions in a certain logical calculus, e.g., propositional calculus, a decision tree structure, predicate calculus, or calculi specially developed for inductive inference, such as variable valued logic systems VL_1 (Michalski [15]) or VL_2 (Michalski [16]).

(b) a set of rules which define a *problem environment*, i.e., represent knowledge about the induction problem under consideration. This includes definitions of value sets of all descriptors* used in the data rules, the properties of descriptors and their interrelationships and any "world knowledge" characteristic to the problem at hand.

(c) a *preference* or (*optimality*) *criterion*, which for any two symbolic descriptions of an assumed form, and of the same generalization class, specifies which one is more preferable, or states that they are equally preferable.

**Descriptors* are variables, relations and functions which are used in symbolic descriptions of objects or situations.

The problem is to determine a set of *inductive assertions (output descriptions)*:

$$\begin{array}{llll} C'_{11} ::> K_1, & C'_{12} ::> K_1, & \ldots & C'_{1r1} ::> K_1 \\ C'_{21} ::> K_2, & C'_{22} ::> K_2, & \ldots & C'_{2r2} ::> K_2 \\ \vdots & & & \\ C'_{m1} ::> K_m, & C'_{m2} ::> K_m, & \ldots & C'_{mrm} ::> K_m \end{array} \tag{2}$$

which are *most preferable* among all sets of rules in an assumed format, that do not contradict the *problem environment* rules, and which are, with regard to the data rules, *consistent* and *complete*.

A set of inductive assertions is *consistent* with regard to data rules, if any situation which satisfies a data rule of some generalization class either satisfies an assertion of the same class, or does not satisfy any assertion.

A set of assertions is *complete* with regard to input rules, if any situation which satisfies some data rules also satisfies some assertion in the set.

It is easy to see that if a set of assertions is consistent and complete with regard to the data rules, then it is semantically equivalent to or more general than the data rules (i.e., there may exist situations which satisfy an assertion but do not satisfy any data rules).

From a given set of data rules it is usually possible to derive many different sets of hypotheses which are consistent and complete, and which satisfy the problem environment rules. The role of the preference criterion is to select one (or a few alternatives) which is (are) most desirable in the given application. The *preference criterion* may refer to the simplicity of hypotheses (defined in some way), their generality, the cost of measuring the information needed for their evaluation, their degree of approximation to the given facts, etc. (Michalski [16]).

We will distinguish following special types of induction (this is not an exhaustive classification):

I. Learning from examples

Within this type three subclasses of problems were studied most:

a. concept acquisition, or learning a *characteristic description* of a class of entities (representing a concept).

b. classification learning, or learning *discriminant descriptions* of related classes of objects.

c. sequence prediction, or discovery of a rule which generates a given sequence of entities.

II. Learning from observation

It is a process of "conceptual clustering," which reveals a conceptual structure underlying an arbitrary collection of entities. It produces a *taxonomic description* of

Most of the research on computer induction has dealt with a special subproblem of type Ia, namely learning a conjunctive concept (description) characterizing a given class of entities. Here the data rules involve only one generalization class (which represents a certain concept), or two generalization classes; the second class being the set of "negative examples" (e.g., Winston [17], Vere [18], Hayes-Roth [19]). Where there is only one generalization class (the so-called *uniclass generalization*) there is no natural limit for generalizing the given set of descriptions. In such case the limit can be imposed, e.g., by the form of expression of the inductive assertion (e. , that it should be a most specific conjunctive generalization wi in the given notational framework, as in (Hayes-Roth [19]) and (Vere [18]), or the assumed *degree of generality* (Stepp [20]). When there are negative examples the concept of *near miss* (Winston [17]) can be used to effectively determine the limit of generalization .

A general problem of type Ia is to learn a *characteristic description* (it can be, e.g., a disjunctive description, grammar, or an algorithm) which characterizes all entities of a given class, and does not characterize any entity which is not in this class.

Problems of type Ib are typical pattern classification problems. Data rules involve many generalization classes; each generalization class represents a single pattern recognition class. In this case, the individual descriptions C_{ij} are generalized so long as it leads to their simpliciation and preserves the condition of consistency (e.g., Michalski [21]). Obtained inductive assertions are *discriminant descriptions*, which permit one to distinguish one recognition class from all other classes. A descriminant description of a class is a special case of characteristic description, where any object which is not in the class is in one of the finite (usually quite limited) number of other classes. Of special interest are discriminant descriptions which have minimal cost (e.g., the minimal computational complexity, or minimal number of descriptors involved).

Problems of type Ic are concerned with the discovery of a rule governing th generation of an ordered sequence of entities. The rule may be deterministic (as in letter sequence prediction (e.g., Simon & Lea [22]), or nondeterministic, as in the card game EULESIS (Dietterich [23]). Data rules involve here only one generalization class, or two generalization classes, where the second class represents "negative examples."

Problems of type II (learning from observation) are concerned with determining a structure underlying a collection of entities. In particular, such a structure can be a partition of the collection into clusters of entities representing certain single concepts ("conceptual clustering," Michalski [24]). Data descriptions in (I) represent in this case individual entities, and they all belong to the same generalization class (i.e., data descriptions consist of a single row of data rules in e.g. (1)).

Methods of induction can be characterized by the type of language used for expressing initial descriptions C_{ij} and final inductive assertions C'_{ij}. Many authors use a restricted form of predicate calculus (usually quantifier-free) of, or some equivalent notation (e.g., Morgan [25], Fikes, Hunt and Nilsson [26], Banerji [27], Cohen [38], Hayes-Roth and McDermott [29], Vere [18]).

In our earlier work we used a special propositional calculus with multiple-valued variables, called variable-valued logic system VL_1. Later on we developed an extension of the first order predicate calculus, called VL_{21} (Michalski [16]). It is a much richer language than VL_1, which includes several novel operators not present in predicate calculus, e.g., the *internal conjunction*, *internal disjunction*, the *exception*, the *selector*. We found these operators very useful for describing and implementing generalization processes; they also directly correspond to linguistic constructions used in human descriptions. VL_{21} also provides a unifying formal framework for adequately handling descriptors measured on different scales. (The orientation toward descriptions with descriptors of different types is one of the unique aspects of our approach to induction.)

2.2 Relevancy of Descriptors in Data Descriptions

A fundamental question underlying any machine induction problem is that of what information the machine is given as input data, and what information the machine is supposed to produce. An important specific question here concerns data relevancy, i.e., how relevant to the problem under consideration must be the variables (in general, descriptors) in the input data, and what is the relationship between variables in the output descriptions and the initial variables?

It is useful to distinguish three cases:

1. The input data consists of descriptions of objects in terms of variables which are relevant to the problem, and the machine is supposed to determine a logical or mathematical formula of an assumed form involving the given variables (e.g., a disjunctive normal expression, a regression polynomial, etc.).

2. The input data consists of descriptions of objects as in case 1, but the descriptions may involve, in addition to relevant variables, a relatively large number of irrelevant variables. The machine is to determine a solution description involving only relevant variables.

3. This case is like case 2, except that the initial descriptions may not include the relevant variables at all. They must include, however, among irrelevant variables, also variables whose certain functions (e.g., represented by mathematical expressions or intermediate logical formulas) are relevant variables. The final formula is then formulated in terms of the derived variables.

The above cases represent problem statements which put progressively less demand on the content of the input data (i.e., on the human defining the problem) and more demand on the machine.

The early work on concept formation and the traditional methods of data analysis represent case 1. Most of the recent research deals with case 2. In this case, the method of induction has to include efficient mechanisms of determining irrelelvant variables. The logic provides such mechanisms, and this is one of the advantages of logical type solutions. Case 3 represents the subject of what we call *constructive induction*.

Our research on induction using system VL_1 and initial work using VL_{21} has dealt basically with case 2. Later on we realized how to approach constructive induction, and formulated the first constructive generalization rules. We have incorporated them in our inductive program INDUCE 1 (Larson and Michalski [30], Larson [31]) and in the newer improved version INDUCE-1.1 (Dietterich [32]).

The need for introducing the concept of constructive induction may not be obvious. The concept has basically a pragmatic value. To explain this, assume first that the output assertions involve derived descriptors, which stand for certain expressions in the same formal language. Suppose that these expressions involve, in turn, descriptors which stand for some other expressions, and so on, until the final expressions involve only initial descriptors. In this case the constructive induction simply means that the output descriptions are multi-level or recursive.

But this is not the only interesting case. Derived descriptors in the output assertions may be any arbitrary, fixed (i.e., not learned) transformations of the input descriptors, specified by a mathematical formula, a computer program, or, even implemented in hardware (e.g., the hardware implementation of fast Fourier transform). Their specification may require language quite different from the accepted formal descriptive language. To determine these descriptors by learning, in the same fashion as the output descriptions, may be a formidable task. They can be determined, e.g., through suggestions of possibly useful transformations provided by an expert, or as a result of some generate-and-test search procedure. In our approach, the derived descriptors are determined by *constructive induction rules*, which represent segments of problem-oriented knowledge of experts.

2.3 Problem Specification and the Form of Inductive Assertions

The induction process starts with the *problem specification* and ends with a set of alternative *inductive assertions*. The *problem specification* consists of a) *data rules*, b) *specification of the problem environment* and c) the *preference criterion*. We will briefly discuss each of these topics.

2.3.1 Form of data rules and inductive assertions

In program INDUCE 2, the data descriptions, C_{ij}, and inductive assertions, C'_{ij}, are *c-formulas* (or VL_{21} *terms*), defined as products of VL_{21} selectors, with zero or more quantifiers in front. For example, a C'_{ij} can be:

$$\exists(2)P1,P2\ [\text{color}(P1) = \text{red},\text{blue}][\text{weight}(P1) > \text{weight}(P2)]$$
$$[\text{length}(P2) = 3..8][\text{ontop}(P1,P2)]\wedge$$
$$[\text{shape}(P1) \cdot \text{shape}(P2) = \text{box}]$$

Paraphrasing the rule in English:

There are two (and only two) parts P1, P2 such that color of P1 is red or blue, weight of P1 is greater than of P2, length of P2 is between 3 and 8 inclusively, P1 is on top of P2, shape of P1, and of P2 is 'box'.

For the description of the language see Michalski [21]. The concept of numerical quantifier $\exists(k)P_1, P_2, \ldots$, is explained in sec. 3 (rule (vii)). Since selectors can include internal disjunction and involve concepts of different levels of generality (as defined by the generalization tree; see next section), the c-formulas are more general concepts than conjunctive statements of predicates.

Other desirable forms of C_{ij} are:

- Assertions with the exception operator

$$(T_1 \vee T_2 \vee \ldots) \veebar T \qquad (3)$$

where T, T_1, T_2, ... are c-formulas, and $\veebar$ is the exception operator (see Appendix 1).

The motivation for this form comes from the observation that a description can be simpler in some cases, if it states an overgeneralized rule and specifies the exceptions. We have introduced this concept in the past (Michalski 74), but have not made much progress with it. Recently Vere (1978) proposed an algorithm for handling such assertions in the framework of conventional conjunctive statements. He allows several levels of exception, which we consider undesirable because of the postulate of comprehensibility.

- Implicative assertions

$$T(T_1 \rightarrow T_2) \tag{4}$$

Production rules used in knowledge-based inference systems are a special case of (4), when T is omitted and there is no internal disjunction. Among interesting inductive problems regarding this case are:

1. developing algorithms for exposing contradictions in a set of implicative assertions
2. deriving simpler assertions from a set of assertions
3. generalizing assertions so that they may answer a wider class of questions while being consistent.

Various aspects of the last problem within a less general framework were studied, e.g., by Hedrick [34].

- Case assertions

$$([f = R_1] \rightarrow T_1) \vee ([f = R_2] \rightarrow T_2) \vee \ldots \tag{5}$$

where R_1, R_2..are pairwise disjoint sets.

This form occurs when a description is split into individual cases characterized by different values of a certain descriptor.

3.2.2 Specification of the problem environment

The problem environment is defined by the specification of the types of the descriptors, their values sets and their interrelationships.

- Types of descriptors

The process of generalizing a description depends on the type of descriptors used in the description. The type of a descriptor depends on the structure of the value set of the descriptor. We distinguish among three different structures of a value set:

1. *Unordered*

   Elements of the domain are considered to be independent entities, no structure is assumed to relate them. A variable or function symbol with this domain is called *nominal* (e.g., variable 'blood type', or relation *contains*(A, B1, B2) (meaning: A contains B1 and B2).

2. *Linearly Ordered*

   The domain is a linearly ordered set. A variable or function symbol with this domain is called *linear* (e.g., military rank, temperature, weight). Variables measured on ordinal, interval, ratio and absolute scales are special cases of a linear descriptor.

3. *Tree Ordered*

   Elements of the domain are ordered into a tree structure, called a *generalization tree*. A predecessor node in the tree represents a concept which is more general than the concepts represented by the dependent nodes (e.g., the predecessor of nodes 'triangle, rectangle, pentagon, etc.,' may be a 'polygon'). A variable or function symbol with such a domain is called *structured*.

Each descriptor (a variable or function symbol) is assigned its type in the specification of the problem. In the case of structured descriptors, the structure of the value set is defined by inference rules (see descriptor shape(B_1) in example in sec. 3).

- Relationships among descriptors

In addition to assigning a domain to each variable and function symbol, one defines properties of variables and atomic functions characteristic for the given problem. They are represented in the form of inference rules. Here are a few examples of such properties.

1. Restrictions on Variables

   Suppose that we want to represent a restriction on the event space saying that if a value of variable x_1 is 0 ('a person does not smoke'), then the variable x_3 is 'not applicable' (x_3 - kind of cigarettes the person smokes). This is represented by a rule:

$$[x_1 = 0] \Rightarrow [x_3 = \text{NA}]$$

NA = not applicable

2. Relationships Between Atomic Functions

   For example, suppose that for any situation in a given problem, the atomic function $f(x_1, x_2)$ is always greater than the atomic function $g(x_1, x_2)$. We represent this:

$$T \Rightarrow \forall x_1, x_2\ [f(x_1, x_2) > g(x_1, x_2)]$$

3. Properties of Predicate Functions

   For example, suppose that a predicate function is transitive. We represent this:

$$T \doteqdot \forall x_{\cdot} \quad , x_3([\mathrm{left}(x_1,x_2)][\mathrm{left}(x_2,x_3)] \rightarrow [\mathrm{left}(x_1,x_3)])$$

Oth[illegible]ypes of relationships characteristic for the problem en[illegible]ronment can be represented similarly.

The rationale behind the inclusion of the *problem environment description* reflects our position that the guidance of the process of induction by the knowledge pertinent to the problem is necessary for nontrivial inductive problems.

2.3.3 The preference criterion

The preference criterion specifies the desired properties of the solution to the problem, i.e., the properties of hypotheses being sought. There are many dimensions, independent and interdependent, on which the hypotheses can be evaluated. The weight given to each dimension depends on the ultimate use of the hypothesis (e.g., the number of operators in it, the quantity of information required to encode the hypothesis using operators from an a priori defined set (Coulon and Kayser [33]), the scope of the hypothesis relating the events predicted by the hypothesis to the events actually observed (some form of measure of degree of generalization), the cost of measuring the descriptors in the hypothesis, etc. Therefore, instead of defining a specific criterion, we specify only a general form of the criterion. The form, called a 'lexicographic functional' consists of an ordered list of criteria measuring hypothesis quality and a list of 'tolerances' for these criteria (Michalski [15]).

An important and somewhat surprising property of such an approach is that by properly defining the preference criterion, the same computer program can produce either the *characteristic* or *discriminant* descriptions of object classes.

3. GENERALIZATION RULES

The transformation from data descriptions (eg. (1)) to inductive assertions (eq. (2)) can be viewed (at least conceptually) as an application of certain *generalization rules* to the data descriptions or intermediate descriptions.

A *generalization rule* is defined as a rule which transforms one or more symbolic descriptions (data rules) in the same generalization class into a new description (inductive assertion) of the same class which is *equivalent* or *more general* than the set of initial descriptions.

A description

$$V :: > K \tag{6}$$

is *equivalent* to a set of

$$\{V_i :: > K\},\ i = 1, 2, \ldots \tag{7}$$

if any *event* (a description of an object or situation) which satisfies at least one of the V_i, $i = 1, 2, \ldots$, satisfies also V, and conversely. If the converse is not required, the rule (6) is said to be *more general than* (7).

The generalization rules are applied to data rules under the condition of preserving consistency and completeness, and achieving optimality according to the preference criterion. A basic property of a generalization transformation is that the resulting rule has UNKNOWN truth-status; being a hypothesis, its truth-status must be tested on new data. Generalization rules do not guarantee that the inductive assertions are useful or plausible.

We have formalized several generalization rules, both for non-constructive and constructive induction. (The notation $D_1 \mid\!< D_2$ specifies that D_2 is more general than D_1).

<u>Non-constructive rules:</u>

(i) the *extending reference* rule

$$V[L = R_1] :: > K \; | \! < \; V[L = R_2] :: > K$$

where L - is an atomic function

$R_2 \supseteq R_1$, and R_1, R_2 are subsets of the value set, D(L), of descriptor L.

V - an arbitrary description (a *context formula*)

This is a generally applicable rule; the type of descriptor L does not matter.

(ii) The *dropping selector* (or *dropping condition*) rule

$$V[L = R] :: > K \; | \! < \; V :: > K$$

This rule is also generally applicable. It is one of the most commonly used rules for generalizing information. It can be derived from rule (i), by assuming that R_2 in (i) is equal the value set D(L). In this case the selector $[L = R_2]$ always has truth-status TRUE, and as such can be removed.

(iii) The *closing interval* rule

$$\left.\begin{array}{l} V[L = a] :: > K \\ V[L = b] :: > K \end{array}\right| < \; V[L = a..b] :: > K$$

This rule is applicable only when L is a linear descriptor.

To illustrate rule (iii), consider as objects two states of a machine, and as a generalization class, a characterization of the states as *normal*. The rule says that if the states differ only in that the machine has two different temperatures, say, a and b, then the hypothesis is made that all states in which the temperature is in the interval $[a,b]$ are also *normal*.

(iv) The *climbing generalization* tree rule

$$\begin{array}{l} \text{one or} \\ \text{more} \\ \text{rules} \end{array} \left\{\begin{array}{l} V[L = a] :: > K \\ V[L = b] :: > K \\ \quad \vdots \\ V[L = i] :: > K \end{array}\right| < \; V[L = s] :: > K$$

where L is a structured descriptor

s - represents the lowest parent node whose descendents include nodes a, b, ... and i, in the generalization tree for L.

The rule is applicable only to selectors involving structured descriptors. This rule has been used, e.g., in (Winston [17], Hedrick [34], Lenat [35]).

Example:

$$\left.\begin{array}{l} V[\text{shape}(p) = \text{triangle}] :: > K \\ V[\text{shape}(p) = \text{rectangle}] :: > K \end{array}\right| \;\mathrel{|}\!\!< V[\text{shape}(p) = \text{polygon}] :: > K$$

(v) The *extension against* rule

$$\left.\begin{array}{l} V_1[L = R_1] :: > K \\ V_2[L = R_2] :: > \neg K \end{array}\right| \;\mathrel{|}\!\!< [L \neq R_2] :: > K$$

where $R_1 \cap R_2 = \emptyset$

V_1 and V_2 - arbitrary descriptions.

This rule is generally applicable. It is used to take into consideration 'negative examples', or, in general, to maintain consistency. It is a basic rule for determining discriminant class descriptions.

(vi) The '*turning constants into variables*' rule

$$\text{one or more rules} \left\{\begin{array}{l} V[p(a,Y)] :: > K \\ V[p(b,Y)] :: > K \\ \cdot \\ \cdot \\ \cdot \\ V[p(i,Y)] :: > K \end{array}\right| \;\mathrel{|}\!\!< \exists x,\ V[p(x,Y)] :: > K$$

where Y stands for one or more arguments of atomic function p.

x is a variable whose value set includes a, b, ..., i.

It can be proven that this rule is a special case of the *extending reference* rule (i). This is a rule of general applicability. It is the basic rule used in works on induction employing predicate calculus.

(vii) *Extending the quantification domain rule*

In the simplest case, the rule changes the universal quantifier into the existential quantifier:

$$\forall\, P,\ V(P) \quad |\!< \quad \exists\, P,\ V(P)$$

where V(P) stands for a formula containing the quantified variable P.

Using the concept of a *numerical quantifier,* the rule can be defined in a more general way.

Let the expression

$$\exists(S)P,\ V(P)$$

stand for a statement, which is true, if the formula V(P) is true for at least one number of occurrences of P, specified in the set of integers S. (S is called the *quantification domain*). For example,

$$\exists(2..8)P,\ V(P)$$

states that there are 2 to 8 P-s for which V(P) is true. Thus, $\exists P,\ V(P)$ is equivalent to $\exists(\geq 1)P,\ V(P)$

and

$\forall P,\ V(P)$ is equivalent to $\exists(k)P,\ V(P)$, where k is the cardinality of the value set of P.

The general form of the rule is:

$$\exists(S_1)P,\ V(P) \quad |\!< \quad \exists(S_2)P,\ V(P)$$

where $S_1 \subseteq S_2$.

If in a quantifier expression (k)P, ..., where k is an integer, one wants to distinguish between different P-s, then one writes (k) P_1, P_2, ..., P_k, ...

Constructive Rules:

Constructive generalization rules generalize descriptions by involving new (*derived*) descriptors, which are functions of initial or other derived descriptors. Thus, these rules evoke *constructive induction procedures*, which generate new descriptors.

These rules may represent knowledge or heuristics which are of general applicability (e.g., capturing semantical dependencies of natural language), or applicable only in the specific problem domain. There is no limit for such rules.

Here are a few examples of more general rules:

(c-i) *Counting rules:*

- CQ rule (count quantified arguments)

Given an expression with a quantifier form $\exists$(k) $P_1, P_2, \ldots, P_k$, the rule generates descriptors '#P-COND', which measure the number of P_i-s satisfying certain condition COND (since there may be many conditions formulated, the rule can potentially generate a large number of such descriptors). If COND is not specified, the descriptor simply counts quantified arguments.

For example, if the COND is 'the arguments P_i such that [*attribute*$_1$(P_i) = R]', then the generated descriptor will be '# -*attribute*$_1$-R'. If the *attribute*$_1$ is, e.g., length, and R is ..4], then the derived descriptor is '#P_i-length-2..4' (the number of P_i-s, whose length is between 2 and 4, inclusively).

- CA-rule (count arguments)

If an initial descriptor is a relation with variable or fixed number of arguments, $REL(P_1,P_2,\ldots)$, the rule generates descriptors '#P-COND' , which measure the number of arguments in REL which satisfy condition COND.

Similarly to the above, there may be many such descriptors generated (each with different COND).

For example, if relation is *contains*$(A,B_1,B_2,\ldots)$, i.e., A contains objects $B_1,B_2,\ldots$, and COND is 'B_i-s which are large and red', then the derived descriptor '#B-large-red-A-contains' measures the number of B_i-s which are large and red, and contained in A.

(c-iii) The *generating chain properties* rule

If the arguments of different occurrences of the same transitive relation (e.g., relation 'above', 'left-of', 'next-to', 'in-front-of', etc.) form a chain, i.e., form a consecutive sequence of objects ordered by the relation, the rule generates descriptors relating to some specific objects in the chain, such as:

LST-object -	the 'least object', i.e., the object at the beginning of the chain (e.g., the bottom object in the case of relation 'above')
MST-object -	the object at the end of the chain (e.g., the top object)
MID-object -	the objects in the middle of the chain
Nth-object -	the object at the Nth position (from LST-object) in the chain.

After identifying these objects, the rule investigates all known properties of them (as specified in the data rules) for determining potentially relevant new descriptors.

The rule also generates a descriptor characterizing the chain itself:

- REL-chain-length - the length of the chain defined by relation REL. For example, if the relation is ON-TOP, then the REL-chain-length would specify the height of a stack of objects.

(c-iv) The *descriptor association* detection rule:

Suppose that in an event, existentially quantified variables $P_1, P_2, \ldots, P_m$ satisfy a condition $COND(P_1, P_2, \ldots, P_m)$, and the values of two descriptors $x(P_i)$, and $y(P_i)$ can be ordered into strictly ascending sequences:

$$<x(P_i)> \quad \text{and} \quad <y(P_i)>$$

where $i \in \{1, 2, \ldots, m\}$.

If the order of P_i-s in both sequences is identical, then a two-argument predicate descriptor is generated

$$\uparrow(x,y)$$

which states that descriptors x and y are related by a monotonically growing function (if x grows then y grows), for P_i-s satisfying $COND(P_1, P_2, \ldots, P_m)$.

If the order of P_i-s in the second sequence is opposite to the order of P_i-s in the first sequence, then another descriptor is generated:

$$\downarrow(x,y)$$

stating that 'if x grows then y decreases', for P_i-s satisfying the $COND(P_1, P_2, \ldots, P_m)$.

The above 'monotonic' definition of derived descriptor $\uparrow(x,y)$ (or $\downarrow(x,y)$) can be generalized by not requiring that the order of P_i-s in the corresponding sequences is identical (or opposite), but 'sufficiently similar' (or 'sufficiently dissimilar'), where similarity is measured by, e.g., the coefficient of statistical correlation.

Concluding this subject, we will note that the concept of generalization rules is very useful for understanding and classifying different methods of inductive learning (Dietterich and Michalski [36]).

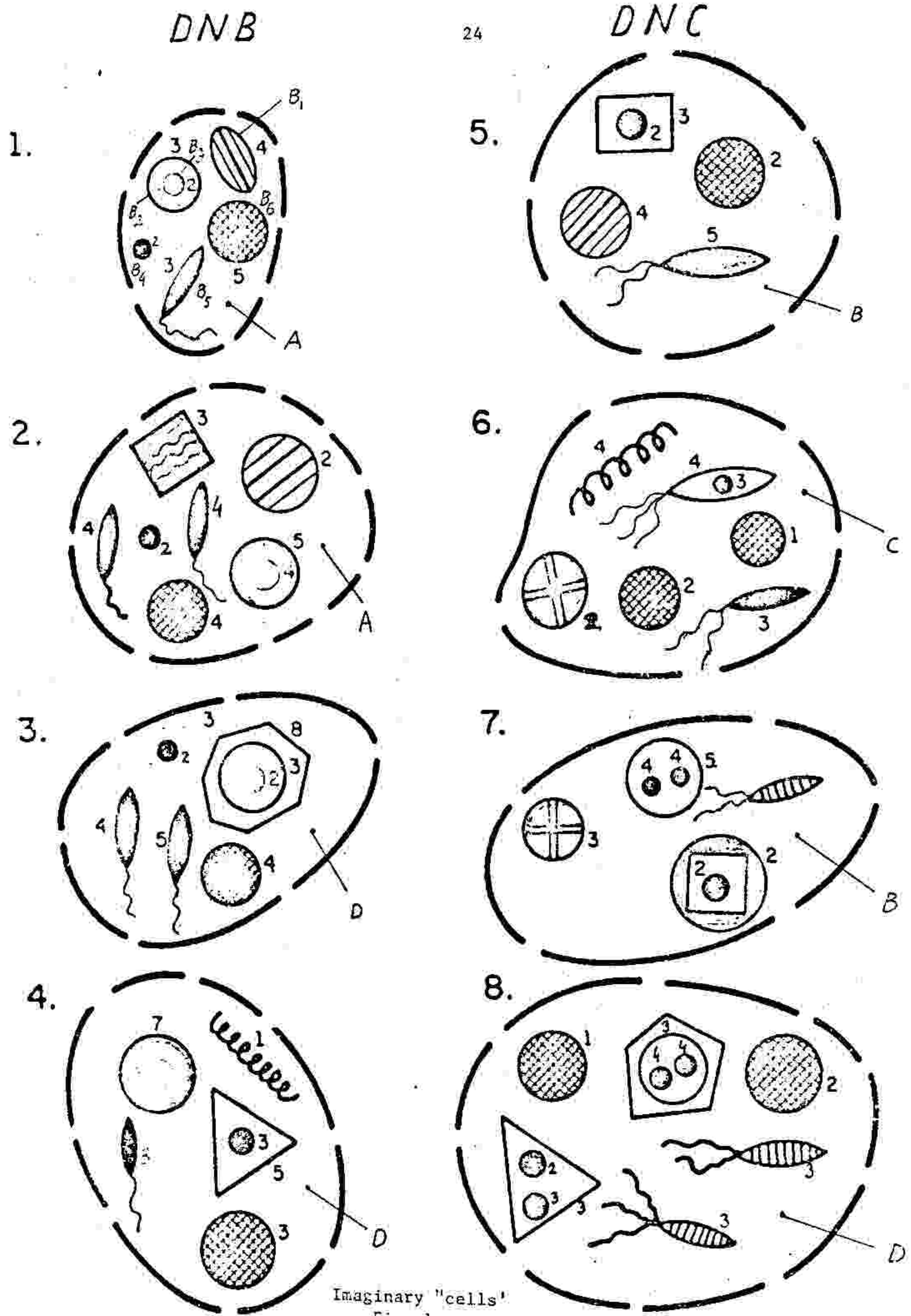
3. LEARNING FROM EXAMPLES

We will discuss here briefly the subject of learning from examples, considering it from the viewpoint of developing methods for 'conceptual' data analysis . Such methods are intended to discover conceptual (i.e., logical functional, or causal) relationships in data, rather than statistical, which are the subject of conventional methods of data analysis.

For concreteness, let us consider a simple data analysis problem, involving imaginary 'cells' (fig. 1). Suppose that cells DNB represent a sample of cancerous cells, and cells DNE -- a sample of normal cells. Suppose that a researcher wants to determine:

- all important common properties of cancerous cells, and of normal cells (i.e., to determine characteristic descriptions of each class)
- properties differentiating between the two classes of cells (i.e., discriminant descriptions of each class).

An assumption is made that the properties to be discovered may involve both the quantitative information about the cells and their components, as well as the qualitative information, which includes nominal variables and



Imaginary "cells"

relationships existing among the components. Using this example we will briefly describe the methodology underlying program INDUCE 2 (a successor of the earlier program INDUCE 1.1 (Larson and Michalski [30], Larson [31], Dietterich [32]).

The solution to the posed problem (or similar problems) can be obtained by a successive repetition of a 'focus attention-hypothesize-test' cycle.

The 'focus attention' phase is concerned with defining the scope of the problem under consideration. This includes selecting descriptors appearing to be relevant, specifying underlying assumptions and formulating the relevant problem knowledge. This phase is performed by a researcher; it involves his/her technical knowledge and informal intuitions. The 'test' phase examines the hypotheses and tests them on new data. This phase may require collecting new samples, performing laboratory experiments, and/or critically analyzing the hypotheses, involving knowledge inaccessible to any currently feasible computer program.

It is the 'hypothesize' phase in which the program INDUCE 2 may play a useful role, that of an assistant in conducting search for the most plausible and/or most interesting hypotheses.

This search may be a formidable combinatorial task for a researcher, if the size of the sample data is large, and each item of the data (in this case, a cell) is described by many variables and/or relations.

The methodology underlying INDUCE 2 requires a collaboration between a user and the program, in which each party does what it can do best. The major steps are as follows:

1. The user formulates the initial space of descriptors, and specifies the type, domain and any special properties of each descriptor (e.g., the transitivity of a relation).

In the case of the structured descriptors, the user also specifies the structure of the domain.

Suppose that for our simple example problem, the following descriptors are selected:

I Global (descriptors characterizing a whole cell)

- circ - the number of segments in the circumference of the cell

  Type: linear

  Domain: {1..10}

- sol - the type of the phyciological solution in the cell (marked in fig. 1 by capital letters)

  Type: nominal

  Domain: {A,B,C,D}

II Local (descriptors characterizing cell bodies and their relationship

- shape (B_i) - the shape of body B_i

  Type: structured

  Leaves of the domain: {circle, ellipse, heptagon, triangle, square, boat, spring}

  Higher nodes are defined by rules:

  [shape = circle, ellipse] ⇒ [shape = oval]

  [shape = triangle, square, heptagon] ⇒ [shape = polygon]

  [shape = oval, polygon] ⇒ [shape = regular]

  [shape = spring, boat] ⇒ [shape = irregular]

- texture (B_i) - the texture of B_i

  Type: nominal

  Domain: {blank, grid, solid-black, solid-grey, stripes, cross, wavy}

- weight (B_i) - the weight of body B_i

  Type: linear

  Domain: [1..5]

- orient (B_i) - the orientation of B_i

  Type: linear - cyclic (the last element is followed by the first)

  Domain: {N, NE, E, SE, S, SW, W, NW}

- contains ($C,B_1,B_2,\ldots$) - C contains bodies $B_1,B_2,\ldots$

  Type: nominal

  Domain: {true,false}

  Properties: transitive relation

- hastails ($B,L_1,L_2,\ldots$) - B has tails $L_1,L_2,\ldots$

  Descriptor applicable only if [shape(C_i) = boat]

  Type: nominal

  Domain: {true,false}

Note that descriptors 'contains' and 'hastails' are predicates with variable number of arguments. Descriptor 'contains' is characterized as the transitive relation. Descriptors 'hastails' and 'orient' are applicable only under a certain condition.

2. The user formulates data rules, which describe cells in terms of selected descriptors and specify the generalization class associated with each cell. For example, following is a data rule for the DNB cell 1:

$\exists(1)\mathrm{CELL}_1\ \exists(6)B_1,B_2,\ldots,B_6$ [contains($\mathrm{CELL}_1,B_1,B_2,\ldots,B_6$)]

[circ(CELL_1)=8][sol(CELL_1)=A][shape(B_1)=ellipse]

[texture(B_1)=stripes][weight(B_1)=4][orient(B_1)=NW][contains(B_2,B_3)]

[shape(B_2)=circle][texture(B_2)=solid-grey]...

[shape(B_6)=circle][texture(B_6)=grid][weight(B_6)=5]

::> [class=DNB]

3. The user indicates which general rules of constructive induction are applicable, and also formulates any problem-specific rules.

   For example, the counting rule CQ (sec. 3) will generate, among others, a descriptor:

- '#B-black-boat' - the number of bodies whose shape is 'boat' and texture is 'solid-black' (i.e., using COND [shape(B)=boat][texture(B)=solid-black])

   (For simplicity of notation, the name of this descriptor, as well as other descriptors below, has been abbreviated, so it does not follow strictly the naming convention described in sec. 3.) The counting rule CA will generate such descriptors as:

- total-B - the total number of bodies in a cell (if condition COND is NULL)
- indep-B - the number of independent bodies in a cell (assuming COND 'bodies not contained in another body')
- #contains-B - the number of bodies contained in another body B
- #tails-boat-B - the number of tails in a body B, whose shape is 'boat'.

Program INDUCE 2 also allows a user to formulate arbitrary arithmetic expressions, as suggestions of possibly relevant descriptors. For example, the user may suggest a descriptor:

$$\text{weight(CELL)} = \sum_i \text{weight}(B_i),$$

where B_i, $i = 1,2,\ldots$ are quantified variables.

The program also has knowledge of certain concepts, such as *even-odd* number, *area* and *perimter* a triangle or rectangle.

4. Finally, the user specifies the type of description sought and the criterion of preference. For this example, we will assume that both characteristic and discriminant descriptions are sought, and that the criterion of preference for characteristic descriptions is 'to maximize the length of the output c-formula', and for discriminant descriptions--'to minimize the length of the output c-formula'.

For illustration, we will give here samples of characteristic and discriminant descriptions obtained for the DNB 'cells' in fig. 1:

Characteristic descriptions of DNB cells:

- $\exists(1)$ B [weight(B)=5]

  ('in every cell there is one (and only one) body with weight 5')

- $\exists(2)\, B_1, B_2$ [contains (B_1, B_2)][shape$(B_1)\cdot$shape(B_2)=circle]$\wedge$

  [texture(B_1)=solid-grey][weight(B_1)=even]$\wedge$

  [texture(B_2)=solid-black][weight(B_2)=odd]$\wedge$

  [#<u>contains</u>-B_1= 1] (. denotes the *internal conjunction*)

  ('in every cell there are two bodies of circle shape, one contained in another, the outside circle is solid-grey, has 'even' weight, the inside circle is solid-black and has 'odd' weight. The outside circle contains only one body.')

- $\exists(1)$ B, [shape(B)=circle][texture(B)=grid]

  [weight(B) $\geq$ 3]

  ('every cell contains a circle with 'grid' texture, whose weight is at least 3') (also discriminant)

- [circ=even]

  ('the number of segments in the circumference of every cell is even') (also discriminant)

• $\exists$(> 1) B [shape(B)=boat][orient(B)=N,NE]

[$\underline{\text{\#tails-boat}}$(B)=1]

('every cell has at least one body of 'boat' shape, which has one tail with N or NE orientation')

• $\exists$(2) B [shape(B)=circle][texture(B)=sold-black]

alternatively

[$\underline{\text{\#B-circle-solid-black}}$=2]

(each cell has exactly two bodies, which are circles and have solid black texture) (also discriminant)

Discriminant descriptions of DNB cells:

In addition to characteristic descriptions which are also discriminant as marked above), here are some discriminant descriptions:

• $\exists$(1) B [texture(B)=grid][weight(B) $\geq$ 3]

('every cell DNB, as opposed to DNC, has exactly one body with 'grid' texture and weight greater or equal 3')

• $\exists$($\geq$ 1) B [shape(B)=boat][orient(B)=N,NE]

('....at least one 'boat' shape body with orientation N or NE')

• $\exists$($\geq$ 1) B [$\underline{\text{\#tails}}$-boat-B=1]

('...at least one body with number of tails 1')

• $\exists$(1) B [shape(B)=circle][$\underline{\text{\#contains}}$-B=1]

'......a circle containing a single object'

Note, that each description involves the minimum number of conditions necessary to distinguish any DNB cell from any DNC cell. Underscored descriptors are derived descriptors obtained through constructive induction.

The above example is too simple that really unexpected patterns can be discovered. But it illustrates well the potential of a learning program as a tool for searching for patterns in complex data, especially when the relevant properties may involve numerical and relational information at the same time. The program therefore offers a new tool for data analysis.

How does the program work? Earlier implementations of the program are described in (Larson [31], Michalski [16], Dietterich [32]). The new version, INDUCE 2, is under completion and will be described in a separate paper. An outline of the main algorithm is given in Appendix 1. Here we will give a summary of the main ideas, their limitations, and describe some problems for future research.

The work of the program can be viewed essentially as the process of applying generalization rules, inference rules (describing the problem environment) and constructive induction rules to the data rules, in order to determine inductive assertions which are consistent and complete. User_selected preference criteria are used to select the most preferable assertions as the final solution.

The process of generating inductive assertions is inherently combinatorially explosive, so the major question is how to guide this process in order to detect quickly the most preferable assertions.

As described in Appendix 1, the first part of the program generates (by putting together step by step the 'most relevant' selectors) a set of consistent *c-formulas*.

The *relevancy test* for the selectors is a function of the number of data rules covered in the given generalization class, and, in the case of discriminant descriptions, also rules covered in other generalization classes.

C-formulas are represented as labelled graphs, and testing them for consistency (i.e., the null intersection with descriptions of objects in generalization classes other than the class under consideration) or for the *degree of coverage* of the given class is done by testing for subgraph isomorphism. By taking advantage of the labels on nodes and arcs, this operation was greatly simplified. However, it is nevertheless quite time and space consumming.

In the second part, the program transforms the consistent c-formulas into VL_1 events (i.e., sequences of values of certain many-valued variables [Michalski 15], and further generalization is done using AQVAL/1 generalization procedure (Michalski and Larson [37]). During this process, the *extension against, closing the interval* and *climbing generalization tree* generalization rules are applied. The VL_1 events are represented as binary strings, and most of the operations done during this process are logical operations on binary strings. Consequently, this part of the algorithm is very fast and efficient. Thus, the high efficiency of the program is due to the change of the data structures representing the rules into more efficient form, once a relevant set of selectors have been found (by determining consistent generalizations).

A disadvantage of this algorithm is that the extension of references of selectors, achieved by the application of the *extension against*, the *closing interval* and *climbing* generalization rules, is done after a (supposedly) relevant set of selectors have been determined. It is possible, however, that a selector from the initial data rules, or generated by constructive generalization rules which did not pass the 'relevance test', could turn out to be very relevant if its reference was appropriately generalized. On the other hand, applying the above generalization rules to each selector represented as a graph structure (i.e., before the AQVAL procedure takes over) could be computationally very costly. This problem will be aggravated when the number of rule generating derived descriptors will be increased.

We plan to seek solutions to this problem by designing a better *descriptor relevancy test,* determining more adequate data structures for representing selectors and testing intersections with descriptions, and by applying problem knowledge.

Another type of learning from examples is 'sequence prediction', i.e. learning from the data which have a strict linear order. Such a problem occurs when given is a sequence of entities (e.g., letters, or, in more general case, structured objects) and the problem is to discover a rule which might have generated the sequence. The program here has to take into consideration the order, and consequently new form of rules and also derived descriptors can be involved. For more details we will refer the reader to Dietterich [23,40].

4. LEARNING FROM OBSERVATION

The major difference between problems of learning a characteristic description from examples (type IA), and problems of learning from observation (type II) is that in the later problem the input is usually an arbitrary collection of entities, rather than a collection of examples representing a single predetermined conceptual class; and that the goal is to determine a partition of the collection into categories (in general, to determine a structure within the collection), such that each category represents a certain concept.

Problems of this type have been intensively studied in the area of cluster analysis and pattern recognition (as 'learning without teacher'). The methods which have been developed in these areas partition the entities into clusters,such that the entities within each cluster have a high 'degree of similarity', and entities of different clusters have a low 'degree of similarity'.

The degree of similarity between two entities is typically a function (usually a reciprocal of a distance function), which takes into consideration only properties of these entities and not their relation to other entities, or to some predefined concepts. Consequently, clusters obtained this way rarely have any simple conceptual interpretation.

In this section we will briefly describe an approach to clustering which we call *conceptual clustering*. In this approach, entities are assembled into a single cluster, if together they represent some concept from a predefined set of concepts.

For example, consider the set of points shown in Fig. 2.

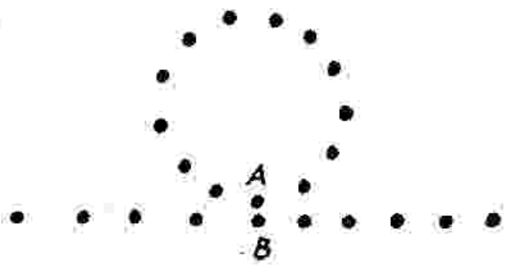


Fig. 2

A typical description of this set by a human is something like 'acircle on a straight line'. Thus, the points A and B, although closer to each other than to any other points, will be put into different clusters, because they are parts of different concepts.

Since the points in Fig. 2 do not fill up completely the circle and the straight line, the obtained conceptual clusters represent generalizations of the initial data points. Consequently, conceptual clustering can be viewed as a form of generalization of symbolic descriptions, similarly to problems of learning from examples. The input rules are symbolic descriptions of the entities in the collection (to interpret this problem as a special case of the paradigm in sec. 2.1, consider the collection as a single generalization class).

If the concepts into which the collection is to be partitioned are defined as C-formulas, then the generalization rules discussed before would apply also here (within the restriction that the resulting formulas cannot intersect). Similarly, constructive induction rules apply.

We will outline here an algorithm for such a clustering, assuming that the concepts are simpler constructs than C-formulas, namely, non-quantified C-formulas with *unary selectors*, i.e., logical products of such selectors. Unary selectors are relational statements:

$$[x_i \# R_i]$$

where:

x_i is one of n predefined variables $(i=1,2,\ldots,n)$

$\#$ is one of the relational operators $= \neq > \geq < \leq$

R_i is a subset of the value set of x_i.

A selector is *satisfied* by a value of x_i, if this value is in relation # with some value from R_i. Such restricted C-formulas are called VL_1 *complexes* or, briefly, *complexes* (Michalski [24]).

Individual entities are assumed to be described by *events*, which are sequences of values of variables x_i:

$$(a_1, a_2, \ldots, a_n)$$

where $a_i \epsilon D(x_i)$, and $D(x_i)$ is the value set of x_i, $i=1, 2, \ldots, n$. An event e is said to *satisfy* a complex, if values of x_i in e satisfy all selectors.

Suppose E is a set of events, each of which satisfies a complex C. If there exist events satisfying C which are not in E, then they are called *unobserved events*. The number of unobserved events in a complex is called the *sparseness* of the complex. We will consider the following problem. Given is an event set E and an integer k. Determine k pairwise disjoint complexes such that:

1. they represent a partition of E into k subsets (a k-partition)
2. the total sparseness of the complexes is minimum.

The theoretical basis and an algorithm for a solution of this problem (in somewhat more general formulation, where the clustering criterion is not limited to sparseness) is described in Michalski [24]. The algorithm is interactive, and its general structure is based on the dynamic clustering method (Diday and Simon [39]). Each step starts with k specially selected data events, called *seeds*. The seeds are treated as representives of k classes, and this way the problem is reduced to essentially a classification problem (type 1b). The step ends with a determination of a set of k complexes defining a partition of E. From such complex a new seed is selected, and the obtained set of k seeds is the input to the next iteration. The algorithm terminates with a k-partition of E, defined by k complexes, which has the 'local' minimum of the total sparseness, or, in general, of the assumed cost criterion. (The algorithm does not guarantee the global minimum.)

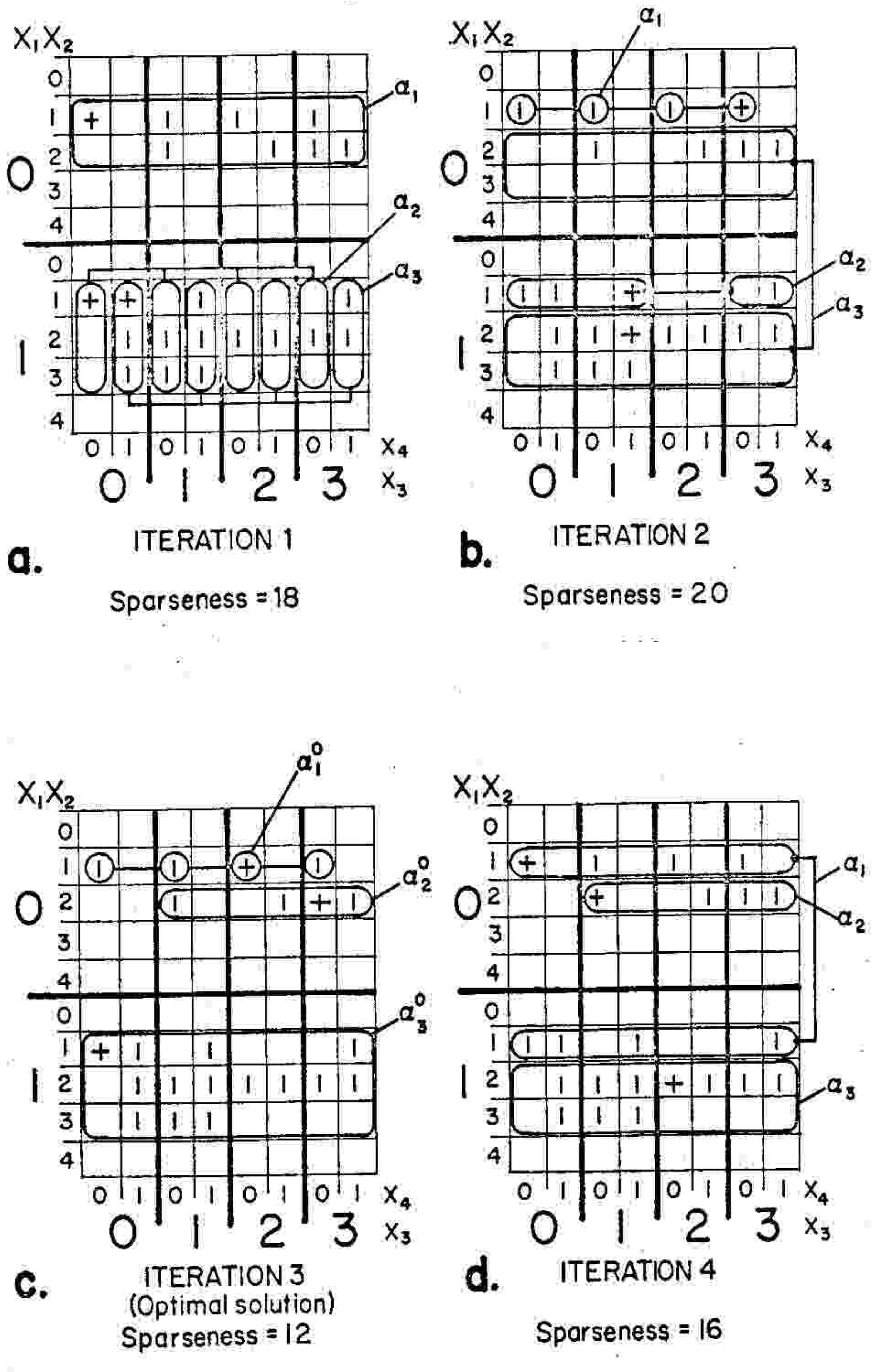
Figure 3 (on the next page) presents an example illustrating this process. The space of all events is defined by variables x_1, x_2, x_3 and x_4, with value set sizes of 2, 5, 4 and 2, respectively. The space is represented as a diagram, where each cell represents a possible event. Cells marked by 1 represent data events, the remaining cells represent unobserved events. Figure 3a shows complexes obtained in the first iteration. The remaining figures show results from the consecutive iterations. Cells representing seed events in each iteration are marked by +.

The solution with the minimum sparseness is shown in Figure 3c. The partition is specified by complexes:

$$\alpha_1^o = [x_1 = 0][x_2 = 1][x_4 = 0]$$

$$\alpha_2^o = [x_1 = 0][x_2 = 2][x_3 = 1..3]$$

$$\alpha_3^o = [x_1 = 1][x_2 = 1..3]$$



An example of the iterative determination of conceptual clusters

Fig. 3

This result was obtained by program CLUSTER/PAF implementing the algorithm.

Another experiment with the program involved clustering 47 cases of soybean diseases. These cases represented four different diseases, as determined by plant pathologists (the program was not, of course, given this information). Each case was represented by an event of 35 many-valued variables. With k=4, the program partitioned all cases into four categories. These four categories turned out to be precisely the categories corresponding to individual diseases. The complexes defining the categories involved known characteristic symptoms of the corresponding diseases. This, and another experiment (involving conceptual clustering of 100 Spanish songs) is described in more detail by Stepp [41].

5. SUMMARY

Inductive learning is described as a process of generalizing and conceptually simplifying symbolic descriptions of given collections of entities. It is shown that this process can be viewed as an application of *generalization rules* and *problem environment rules* to initial and intermediate descriptions. Two types of inductive learning are distinguished: learning from examples and learning from observation. The most studied categories of learning from examples include determination of characteristic and discriminant descriptions, and sequence prediction. Learning form observation ('conceptual clustering') produces a taxonomic description of a collection of entities, which reveals the conceptual structure underlying the collection.

The presented theoretical framework unifies the above types of learning, and was used to develop two learning programs: INDUCE 2 --- for determining characteristic or discriminant structural descriptions, and CLUSTER/PAF --- for conceptual clustering of arbitrary collections of entities. The described methodology is viewed as especially promising for applications such as automated 'conceptual' analysis of experimental data, searching for patterns and abstracting the contents of databases, or aiding knowledge acquisition processes in the development of knowledge-based systems.

ACKNOWLEDGMENT

A partial support of this research was provided by the National Science Foundation under grant MCS-79-06614. Many ideas presented here have been developed over the years in collaboration with numerous graduate students and collaborators in USA and abroad. Without their colaboration this research would have not been possible. To them the author expresses his deepest gratitude.

REFERENCES

[1] Shortliffe, E. G., "MYCIN: A Rule Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection," Ph.D. Thesis, Computer Science Department, Stanford University, October 1974.

[2] Davis, R., "Applications of Meta-level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases," Report No. 552, Computer Science Department, Stanford University, July 1976.

[3] Brachman, R. T., On the Epistomological Status of Semantic Networks, Report No. 3807, Bolt, Beranek and Newman, April 1978.

[4] Michalski, R. S. and Chilausky, R. L., Learning By Being Told and Learning From Examples, Policy Analysis and Information Systems, Special Issue on Knowledge Acquisition and Induction, No. 2, 1980.

[5] Shaw, D.E., Swartout, W. R. and Green, C. C., Inferring Lisp programs from examples, Proceedings of the 4th International Joint Conference on Artificial Intelligence, Vol. I, pp. 351-356, Tbilisi, September 1975.

[6] Jouannaud, J. P., and Kodratoff, Y. (1979), Characterization of a Class of Functions Synthesized from Examples by a Summers-like Method using a "B.M.W." Matching Technique, Sixth International Joint Conf. on Art. Intelligence-Tokyo 1979, pp. 440-447.

[7] Burstall, R. M. and Darlington, J. (1977) A transformation sytem for developing recursive programs. JACM, Vol. 24, No. 1, 44-67.

[8] Biermann, A. W., (1978), The Inference of Regular LISP Programs from Examples, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-8(8) August 1978, pp. 585-600.

[9] Smith, D. R., A Survey of the Synthesis of LISP Programs from Exampsles, International Workshop on Program Construction, Bonas, Sept. 8-12, 1980.

[10] Pettorossi, A., An Algorthm for Reducing Memory Requirements in Recursive Programs Using Annotations, IBID.

[11] Biermann, A. W., and Feldman, J. (1972), Survey of Results in Grammatical Inference, in Frontiers of Pattern Recognition, Academic Press Inc., New York, 1972, pp. 31-54.

[12] Yau, K. C. and Fu, K. S., Syntatic shape recognition using attributed grammars, Proceedings of the 8th Annual EIA Symposium on Automatic Imagery Pattern Recognition, 1978.

[13] Michie, D., "New Face of Artificial Intelligence," Information 3, pp. 5-11, 1977.

[14] Carnap, R. "The Aim of Inductive Logic," in E. Nagel, P. Suppes, and A. Tarski, eds., Logic, Methodology and Philosophy of Science, Stanford, California: Stanford University Press, 1962, pp. 303-318.

[15] Michalski, R. S., "AQVAL/1--Computer Implementation of a Variable-valued Logic System and the Application to Pattern Recognition," Proceedings of the First International Joint Conference on Pattern Recognition, Washington, D.C., October 30-November 1, 1973.

[16] Michalski, R. S., "Pattern Recognition As Knowledge-Guided Computer Induction," Report No. 78-927, Department of Computer Science, University of Illinois, Urbana, Illinois, June 1978.

[17] Winston, P. H., "Learning Structural Descriptions from Examples," Technical Report AI TR-231, MIT AI Lab, Cambridge, Massachusetts, 1970.

[18] Vere, S. A., "Induction of Concepts in the Predicate Calculus," Advance Papers of the 4th International Joint Conference on Artificial Intelligence, Vol. I, pp. 351-356, Tbilisi, Georgia, September 1975.

[19] Hayes-Roth, F., "Patterns of Induction and Associated Knowledge Acquisition Algorithms," Pattern Recognition and Artificial Intelligence, ed. C. Chen, Academic Press, New York 1976.

[20] Stepp, R., "The Investigation of the UNICLASS Inductive Program AQ7UNI and User's Guide, Report No. 949, Department of Computer Science, University of Illinois, Urbana, Illinois, November 1978.

[21] Michalski, R. S., Pattern Recognition as Rule-Guided Inductive Inference, IEEE Trans. on Pattern Analysis and Machine Intelligence, July 1980.

[22] Simon, H.A., G. Lea, "Problem Solving and Rule Induction: A Unified Way," Carnegie-Mellon Complex Information Processing Working Paper #227, Revised June 14, 1973.

[23] Dietterich, T. G., A Methodology of Knowledge Layers for Inducing Descriptions of Sequentially Ordered Events, Report No. 80-1024, Department of Computer Science, University of Illinois, May, 1980.

[24] Michalski, R. S., Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjunctive Concepts, Special Issue on Knowledge Acquisition and Induction, Inter. Journal on Policy Analysis and Information Systems, No. 3, 1980. (Also, Report No. 80-1026, Department of Computer Science, University of Illinois, May, 1980.)

[25] Morgan, C. G., Automated hypothesis generation using extended inductive resolution, Advance Papers of the 4th I. J. Conf. on Artificial Intelligence, Vol. I, pp. 351-356, Tbilisi, Georgia, September 1975.

[26] Fikes, R. E., Hart, R. E. and Nilsson, N. J., Learning and executing generalized robot plans, Artificial Intelligence 3, 1972.

[27] Banerji, R. B., Learning in structural description languages, Temple Uni-ersity Report to NSF Grant MCS 76-0-200, 1977.

[28] Cohen, Brian L., A powerful and efficient structural pattern recognition system, Artificial Intelligence, Vol. 9, No. 3, December 1977.

[29] Hayes-Roth and McDermott, J., An interference matching technique for inducing abstractions, Communications of the ACM, No. 5, Vol. 21, pp. 401-411, May 1978.

[30] Larson, J., R. S. Michalski, "Inductive Inference of VL Decision Rules," Proceedings of the Workshop on Pattern-Directed Inference Systems, Honolulu, Hawaii, May 23-27, 1977, SIGART Newsletter, No. 63, June 1977.

[31] Larson, James, "INDUCE-1: An Anteractive Inductive Inference Program in VL_{21} Logic System," Report No. UIUCDCS-R-77-876, Department of Computer Science, University of Illinois, Urbana, Illinois, May 1977.

[32] Dietterich, T., "Description of Inductive Program INDUCE 1.1," Internal Report, Department of Computer Science, University of Illinois at Urbana-Champaign, October 1978.

[33] Coulon, D., D. Kayser, "Learning Criterion and Inductive Behavior," Pattern Recognition, Vol. 10, No. 1, pp. 19-25, 1978.

[34] Hedrick, C. L., "A Computer Program to Learn Production Systems Using a Semantic Net," Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, July 1974.

[35] Lenat, D., "AM: An artificial intelligence approach to discovery in mathematics as heuristic search," Computer Science Department, Report STAN-CS-76-570, Stanford University, July 1976.

[36] Dietterich, T. and Michalski, R. S., "Learning and Generalization of Characteristic Descriptions: Evaluation Criteria and Comparative Review of Selected Methods," Proceedings of the Sixth International Joint Conference on Artificial Intelligence, pp. 223-231, Tokyo, August 20-23, 1979.

[37] Michalski, R. S. and Larson, J. B., "Selection of Most Representative Training Examples and Incremental Generation of VL_1 Hypotheses: the underlying methodology and the description of programs ESEL and AQ11," Report No. UIUCDCS-R-78-867, Department of Computer Science, University of Illinois, Urbana, Illinois, May 1978.

[38] Mitchell, T. M.,"Vernion Spaces: An approach to concept learning," Doctor of Philosophy Thesis, Stanford University, 1978.

[39] Diday, E. and Simon, J. C., "Clustering Analysis," Chapter in Communication and Cybernetics 10, Ed. K. S. Fu, Springer-Verlag, Berlin, Heidelberg, New York, 1976.

[40] Dietterich, T. G., Papers of the Workshop on CURRENT DEVELOPMENTS in MACHINE LEARNING, July 16-18, Carnegie-Mellon University, Pittsburgh.

[41] Stepp, R., IBID.

APPENDIX 1

Outline of the Top Level Algorithm of INDUCE 2

1. At the first step, the data rules (whose condition parts are in the disjunctive simple forms) are transformed to a new set of rules, in which condition parts are in the form of *c-expressions*. A *c-expression* (a *conjunctive expression*) is a product of selectors accompanied by zero or more quantifier forms, i.e., forms $QFx_1, x_2, \ldots,$ where QF denotes a quantifier. (Note, that due to the use of the internal disjunction and quantifiers, a c-expression represents a more general concept than a conjunction of predicates.)

2. A decision class is selected, say K_i, and all c-expressions associated with this class are put into a set F1, and all remaining c-expressions are put into a set F0 (the set F1 represents events to be *covered* , and set F0 represents constraints, i.e., events not to be *covered*).

3. By application of inference rules (describing the problem environment) and constructive generalization rules, new selectors are generated. The 'most promising' selectors (according to a certain criterion) are added to the c-expressions in F1 and F0.

4. A c-expression is selected from F1, and a set of consistent generalizations (a *restricted star*) of this expression is obtained. This is done by starting with single selectors (called 'seeds'), selected from this c-expression as the 'most promising' ones (according to the preference criterion). In each

subsequent next step, a new selector is added to the c-expression obtained in the previous step (initially the seeds), until a specified number (parameter NCONSIST) of consistent generalizations is determined. Consistency is achieved when a c-expression has NULL intersection with the set F0. This 'rule growing' process is illustrated in fig. A1.

5. The obtained c-expressions, and c-expressions in F0, are transformed to two sets E1 and E0, respectively, of VL_1 events (i.e., sequences of values of certain discrete variables).

A procedure for generalizing VL_1 descriptions is then applied to obtain the 'best cover' (according to a user defined criterion) of set E1 against E0 (the procedure is a version of AQVAL/1 learning program [37].

During this process, the *extension against*, the *closing the interval* and the *climbing generalization tree* rules are applied.

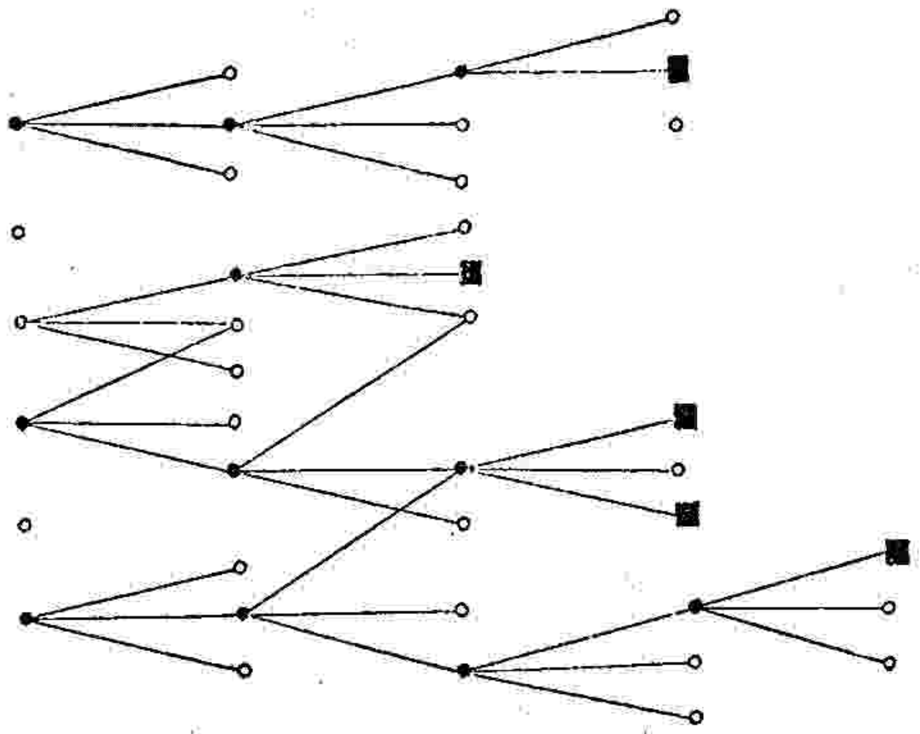
The result is transformed to a new set of c-expressions (a restricted star) in which selectors have now appropriately generalized references.

6. The 'best' c-expression is selected from the restricted star.

7. If the c-expression completely covers F1, then the process repeats for another decision class. Otherwise, the set F1 is reduced to contain only the uncovered c-expressions, and steps 4 to 7 are repeated.

The implementation of the inductive process in INDUCE-1.1 consists of a large collection of specialized algorithms, each accomplishing certain task. Among the most important tasks are:

1. the implementation of the 'rule growing process'
2. testing whether one c-expression is a generalization of ('covers') another c-expression. This is done by testing for subgraph isomorphism.
3. generalization of a c-expression by extending the selector references and forming irredundant c-expressions (includes application of AQ11 AQVAL/1 procedure).
4. generation of new descriptors and new selectors



o - a disgarded c-rule

● - an active c-rule

■ - a terminal node denoting a consistent c-rule

Each arc represents an operation of adding a new selector to a c-rule

The branching factor is determined by parameter ALTER. The number of active rules (which are maintained for the next step of the rule growing process) is specified by parameter MAXSTAR. The number of terminal nodes (consistent generalizations) which program attempts to generate is specified by parameter NCONSIST.

Illustration of the rule growing process
(an application of the dropping selector rule in the reverse order)

Figure A1.