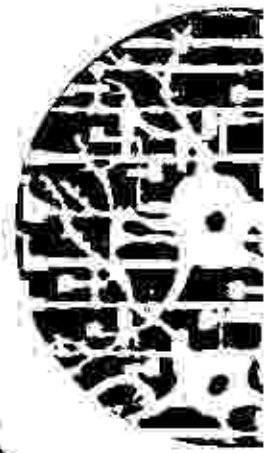
80-11

Reports

Machine Learning & Inference Laboratory



Inductive Inference As Rule-Guided Transformation of Symbolic Descriptions

R. S. Michalski

MLI 80-11

Center for Artificial Intelligence

George Mason University

Invited paper for International Workshop
on Program Construction, Chateau de Bonas
(France), September 8-12, 1980.

INDUCTIVE INFERENCE AS RULE-GUIDED TRANSFORMATION OF SYMBOLIC DESCRIPTIONS

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

ABSTRACT

Inductive inference is viewed as a process of generalizing and simplifying symbolic descriptions, under a guidance of *generalization rules* (representing general inference processes which generalize descriptions) and *problem environment rules* (representing problem specific knowledge). Descriptions are expressed in an extension of predicate logic which uses typed variables and a few novel syntactic forms (a variable-valued logic system VL_2). It is demonstrated that various types of learning from examples (e.g., concept learning or classification), as well as learning from observation can be viewed this way.

Learning from observation is described as a process of partitioning a given collection of entities into clusters (or, in general, into a structure of clusters), such that each cluster represents a single concept selected from a set of a priori known concepts.

INDUCTIVE INFERENCE AS A RULE-GUIDED TRANSFORMATION OF SYMBOLIC DESCRIPTIONS

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

INTRODUCTION

Our understanding of inductive inference 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 computer induction. 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 formal systems e.g. in the form of production rules (Shortliffe [1], Davis [2]) or as 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 seems to be preferable.

Another potential for applying computer induction is in various areas of science, e.g., biology, microbiology, and genetics. Here it 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.

Quite different from the above are goals of research in a special sub-area of computer inductive inference 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 of 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)*:

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

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.

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

("Conceptual clustering," or revealing a conceptual structure underlying a collection of entities.)

Most of the research on computer induction 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 expressing the inductive assertion (e.g., that it should be a most specific conjunctive generalization within the given notational framework, as in (Hayes-Roth [19]) and (Vere [18]), or by 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 discovering a rule governing 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 (1) represent in this case individual entities, and they all belong to the same generalization class (i.e., data descriptions consist of a single row in (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 (usually a quantifier-free) of predicate calculus, 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 have 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 is the question of data relevancy, i.e., of how relevant to the problem under

consideration must be variables (in general, descriptors) in the input data, and how the variables in the output descriptions relate to the initial variables.

We will 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. The 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 1.1, 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:

$$\exists P1,P2\ [color(P1) = r..,blue][weight(P1) > weight(P2)]$$
$$[length(P2) = ..8][ontop(P1,P2)]\wedge$$
$$[shape(P1) \cdot shape(P2) = box]\wedge$$

(see Appendix 1 for explanation)

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

$$(T1 \vee T2 \vee \ldots) \setminus T \qquad (3)$$

where T, T1, T2, ... are c-formulas, and $\setminus$ 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

$$T(T_1 \rightarrow T_2) \qquad (4)$$

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

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

where $R_1, R_2 \ldots$ 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., blood-type).

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 (e.g., see eqs. (8), (9), (10)).

- 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:

$$[x_1 = 0] \Rightarrow [x_3 = NA]$$

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:

$$\forall x_1, x_2, x_3 ([left(x_1, x_2)][left(x_2, x_3)] \Rightarrow [left(x_1, x_3)])$$

   Other types of relationships characteristic for the problem environment 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 defines what is the desired solution to the problem, i.e., what kind of hypotheses are 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 (1) to inductive assertions (2) can be viewed (at least conceptually) as an application of certain

generalization rules.

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 $D_1 \nvdash D_2$ specifies that D_2 is more general than D_1).

Non-constructive rules:

(i) the *extending reference* rule

$$V[L = R_1] :: > K \nvdash V[L = R_2] :: > K$$

where L - is an atomic function

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

V - an arbitrary description.

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

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

$$V[L = R] \; :: > K \;\; \vdash \;\; V \; :: > K$$

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 $[L = R_2]$ has always truth-status TRUE, and as such can be removed.

(iii) The *closing interval* rule

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

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

one or more rules

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

where L is a structured descriptor

s - represents the node at the next level of generality than nodes a, b, ... and i, in the tree domain of 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:

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

(v) The *extension against* rule

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

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

$$\text{one or more rules} \left\{\begin{array}{l} V[p(a,Y)] ::> K \\ V[p(b,Y)] ::> K \\ \quad\vdots \\ V[p(i,Y)] ::> K \end{array}\right| \prec \exists x,\ V[px,Y)] ::> K$$

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.

Constructive Rules:

Constructive generalization rules (*metarules*) generate generalized descriptions of rules in terms of new descriptors, which are functions of the original descriptors, and can be viewed as knowledge-based rules for generating new descriptors. Many such rules can be formulated; we will give here a few examples.

(vii) The *counting* rule

$$V[\text{attribute}_1(P_1)=A]\ldots[\text{attribute}_1(P_k)=A][\text{attribute}_1(P_{k+1})\neq A].$$
$$\ldots[\text{attribute}_1(P_r)\neq A] :: > K \mid\!\!- V[\#P\text{-attribute}_1\text{-}A = k] :: > K$$

where $P_1, P_2, \ldots, P_k, \ldots, P_r$ - are constants denoting, e.g., parts of an object

$\text{attribute}_1(P_i)$ - stands for a certain attribute of P_i-s, e.g., color, size, texture, etc.

$P\#\text{-attribute}_1\text{-}A$ - denotes a new descriptor interpreted as the 'number of P_i-s (e.g., parts) with *attribute* equal A'.

Example:

$$V[\text{color}(P1) = \text{RED}][\text{color}(P2) = \text{RED}][\text{color}(P3) = \text{BLUE}] :: > K$$
$$\mid\!\!- [\#P\text{-color-red} = 2] :: > K$$

(The above is a generalization rule, because a set of objects with any two red parts is a superset of a set of objects with two parts which are red and one part which is blue.)

The rule can be extended to a more general form, in which in addition to the arbitrary context formula V there is a predicate CONDITION $(P_1,\ldots,P_k)$, which specifies some conditions imposed on variables $P_1,\ldots,P_k$.

(viii) The *generating chain properties* rule (a *chain metarule*)

If the arguments of different occurrences of the same relation (e.g., relation 'above', 'left-of', 'next', etc.) form a chain, i.e., are linearly ordered by the relation, the rule generates descriptors relating to specific objects in the chain and computes their properties as potentially relevant characteristics. For example:

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)

position(object) - the position of the object in the chain.

(ix) The *variable association* detection rule

Suppose that in the data descriptions, in the contex of condition $\mathcal{C}$, an ascending order of values of a linear descriptor x_i corresponds to an ascending (descending) order of values of another linear descriptor x_j with the same quantified arguments. For example, whenever descriptor weight(P) takes on increasing values, than also the descriptor lenght(P) takes on the increasing values. In such situations a two-argument predicate descriptor is generated:

$\uparrow(x_i,x_j)$ - if x_j grows with x_i

or

$\downarrow(x_i,x_j)$ - if x_i decreases with x_i

If the number of different occurrences of x_i and x_j is statistically significant, then the "monotonic" definition of descriptors $\uparrow(x_i,x_j)$ and $\downarrow(x_i,x_j)$ can be generalized to:

$$+(x_i,x_j) = \begin{cases} \text{True, if } r(x_i,x_j) \geq T \\ \text{False, otherwise} \end{cases}$$

(positive correlation)

$$+(x_i,x_j) = \begin{cases} \text{True, if } r(x_i,x_j) \leq -T \\ \text{False, otherwise} \end{cases}$$

(negative correlation)

where $r(x_i,x_j)$ denotes the coefficient of statistical correlation, and T is a certain threshold, $0 < T \leq 1$.

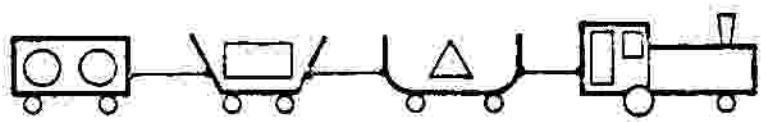
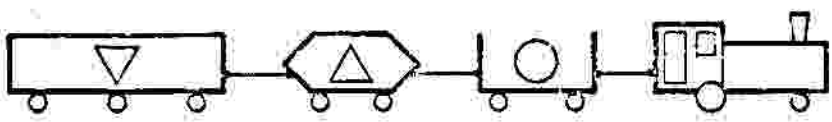
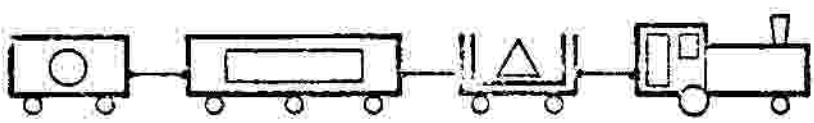
The concept of generalization rules is very useful for understanding and classifying different methods of induction (Dietterich and Michalski [36]).

3. LEARNING FROM EXAMPLES

We will illustrate some aspects of learning from examples by a simple problem involving geometrical constructions. Suppose that two sets of trains, Eastbound and Westbound, are given, as shown in Fig. 1. The problem is to determine a concise, logically sufficient description of each set of trains, which distinguishes one set from the other (i.e., a *discriminant description*, which contains only necessary conditions for distinguishing between the two sets). Using this example we will first briefly describe the learning methodology implemented in computer program INDUCE-1.1 (Larson and Michalski [30], Larson [31], Dietterich [32]) which successfully solved this problem. And next we will discuss some problems for future research.

At the first step, the initial space of descriptors was determined. They were descriptors which seemed to be possibly relevant for the descrimination problem.

1. EASTBOUND TRAINS

1. 
2. 
3. 
4. 
5. 

2. WESTBOUND TRAINS

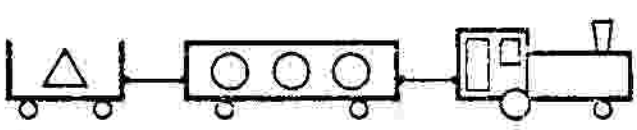
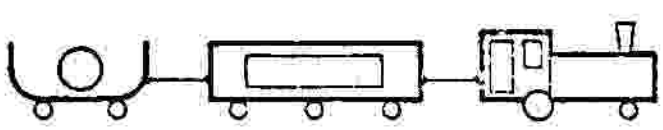
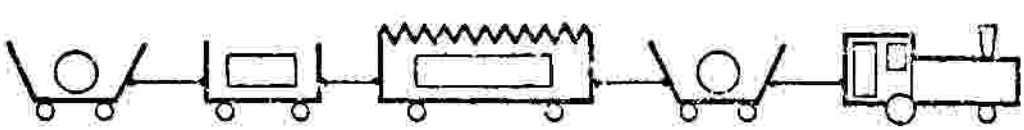
1. 
2. 
3. 
4. 
5. 

Figure 1

Among the eleven descriptors selected in total were:

- **infront**(car_i,car_j) - car_i is in front of car_j.
  (a nominal descriptor)
- length(car_i) - the length of car_i
  (a linear descriptor)
- car-shape(car_i) - the shape of car_i
  (a structured descriptor with 12 nodes in the generalization tree; see eqs. (8) and (9))
- cont-load(car_i,$load_j$) - car_i contains $load_j$
  (a nominal descriptor)
- load-shape($load_i$) - the shape of $load_i$
  (a structured descriptor)
  The value set:

  - circle
  - hexagon
  - triangle
  - rectangle

  → polygon
- nrpts-load(car_i) - the number of parts in the load of car_i
  (a linear descriptor)
- nrwheels(car_i) - number of wheels in car_i
  (a linear descriptor)

At the next step, *data* descriptions were formulated, which characterized trains in terms of the selected descriptors, and specified the train set to which each train belongs. For example, the data description for the second eastbound train was:

$$\exists car_1, car_2, car_3, car_4, load_1, load_2, \ldots$$
$$[\text{infront}(car_1, car_2)][\text{infront}(car_2, car_3)]\ldots[\text{length}(car_1)=\text{long}]\wedge$$

[car-shape(car_1)=engine][car-shape(car_2)=U-shaped][cont-load(car_2,$load_1$)]∧
[load-shape($load_1$)=triangle]...[nrwheels(car_3)=2]..::>[class= Eastbound]

Rules describing the problem environment in this case were rules defining structures of structured descriptors (arguments of descriptors are omitted):

[car-shape=open rctngl,open trapezoid,U-shaped,dbl open rctngl] ⇒ [car-shape=open top] (8)

[car-shape=ellipse,closed rctngl,jagged top,sloping top]⇒[car-shape=closed top] (

[load-shape=hexagon,triangle,rectangle] ⇒ [load-shape=polygon] (1

and that the relation 'infront' is transitive.

The *criterion of preference* was to minimize the number of rules used in describing each class, and, with secondary priority, to minimize the number of selectors (expressions in brackets) in each rule.

The above information was given to INDUCE 1. The program produced the following inductive assertions*:

Eastbound trains:

$\exists car_1$[length(car_1)=short][car-shape(car_1)=closed top]::>[class=Eastbound]

It can be interpreted:

If a train contains a car which is short and has a closed top (1

then it is an eastbound train.

Alternatively,

$\exists car_1, car_2, load_1, load_2$ [infront(car_1,car_2)][cont-load(car_1,$load_1$)]
∧ [cont-load(car_2,$load_2$)][load-shape($load_1$)=triangle]
∧ [load-shape($load_2$)=polygon] :: > [class=Eastbound]

It can be interpreted:

---

*It may be a useful exercise for the reader to try to determine his/her own solutions, before reading the computer solutions.

If a train contains a car whose load is a triangle, and the load of the car behind is polygon, then the train is eastbound. (1

Westbound trains:

[nrcars=3] ∨ car_1[car-shape(car_1)=jagged-top] ::> [class=Westbound]

Either a train has three cars or there is a car with jagged top (1

car_1[nrcars-length-long=2][position(car_1)=3][shape(car_1)=open-top,jagged-top]
::> [class-Westbound]

There are two long cars and the third car has open-top or jagged top. (1

It is interesting to note that the example was constructed with rules (12) and (13) in mind. The rule (11) which was found by the program as an alternative was rather surprising because it seems to be conceptually simpler than rule (12). This observation confirms the thesis of this research that the combinatorial part of an induction process can be successfully handled by a computer program, and, therefore, programs like the above have a potential to serve as an aid to induction processes in various practical problems.

The descriptors underlined by the dotted lines ('nrcars-length-long', 'position(car)') are new descriptors, generated as result of constructive induction. How were they generated? The constructive generalization rules (metarules) are implemented as modules which scan the data rules and search for certain properties. For example, the *counting metarule* checks for each unary descriptor (e.g., length (car)) how many times a value of the descriptor repeats in the data rules.

In our example, it was found that the selector [length (car) = long] occurs for two quantified variables in every Westbound train, and therefore a new descriptor called 'nr cars-length-long' was generated, and a new selector [nr cars-length-long = 2] was formed. This selector, after passing the 'relevancy test', was included in the set of potentially useful selectors. During the generation of alternative assertions, this selector

was used as one of the conditions in the assertion (14). The descriptor 'position(car)' was found by the application of the *chain metarule*.

Now, how does the whole program work? The program is described in papers [Larson 3lab, Michalski 16, Dietterich 32]. In the Appendix 2, we provide a description of the top level algorithm. 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 metarules (generating new descriptors) to the data rules, in order to determine inductive assertions which are consistent and complete. The preference criterion is used to select the most preferable assertions which constitute the 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 2, 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 versus 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, R. S. and Larson, J. B. 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 metarules generating derived descriptors will be increased. We plan to seek solutions to this problem by designing a better *descriptor relevancy test*, determining more adequate dat structures for representing selectors and testing intersections with descriptions, and by applying problem

Another interesting problem is how to provide an inductive program with the ability to discover relevant derived descriptors, which are arithmetic expressions of the input variables and to integrate them as parts of inductive assertions. For example, suppose that the Eastbound trains in fig. 1 are characterized as:

> *in the case of trains with 3 cars, the load of the first two cars is twice the total load of Westbound cars, and in the case of the longer trains, the load of the first two cars is equal the total load of Westbound cars!*

How would one design an efficient algorithm which could discover such an assertion?

Let us now consider a problem of describing, say, the Eastbound trains not in the contex of Westbound trains, but in the contex of every possible train which is not Eastbound. This is a problem of determining *characteristic description* of Eastbound trains (type Ia).

A trivial solution to this problem is a 'zero degree generalization' description, which is the disjunction of descriptions of individual trains. A more interesting solution (although still of 'zero degree generalization') would be some equivalence preserving transformation of such a disjunction, which would produce a computationally simpler description. Allowing a 'non-zero degree generalization' leads us to a great variety of possibilities, called the *version space* (Mitchell [38]). As we mentioned before (Sec.2.1), the most studied solution is to determine the most specific conjunctive generalization (i.e., the longest list of common properties). Another solution is to determine the description of minimal cost whose degree of generality is under certain threshold (Stepp [20]). INDUCE 1.1 gives a solution of the first type, namely, it produces a set of the most specific (longest) c-formulas (quantified logical products of VL_{21} selectors).

Here is an example of such a solution:

- $\exists$car[length(car)=short][car-shape(car)=closed top] $\wedge$
  $\wedge$ [nrwheels(car)=2]

  (In every Eastbound train there is a short car with closed top and two wheels)

- $\exists$car[position(car)=1][car-shape(car)=engine]

- $\exists$car[position(car)=2][car-shape(car)=open-top]

  (The second car in each train has an open-top)

- $\exists$car[position(car)=2,3][shape(car)=traingle]

  [nrcars=4,5]

- $\forall$car[nrwheels=2,3]

The logical product of these formulas is a characteristic description of Eastbound trains.

To keep this paper within reasonable limits, we will skip the discussion of problems of type Ic (i.e., the sequence prediction), referring reader to paper by Diettrich [23].

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.

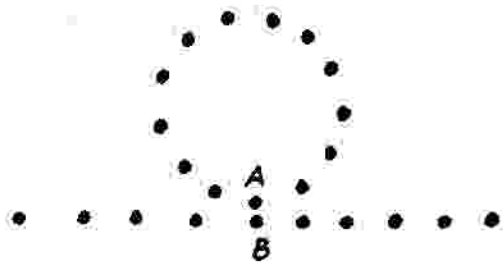


Fig. 2

A typical description of this set by a human is something like 'a circle 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 (within the restriction imposed by the problem, which is that they cannot intersect; as each cluster should be disjoint from other clusters).

We will describe here briefly an algorighm 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,...,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 $a_i \in D(x_i)$, and $D(x_i)$ is the value set of x_i, i=1, 2, ..., n.

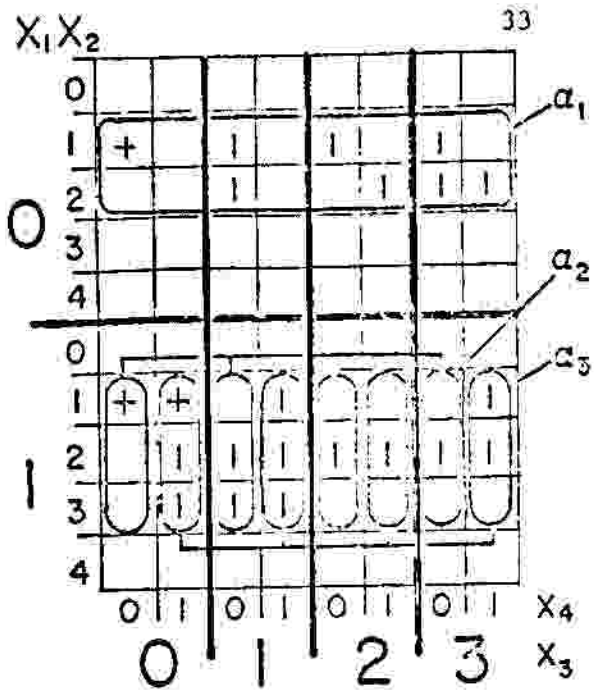
An event e is said to *satisfy* a complex, if values of x_i in e satisfy all selectors.

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.

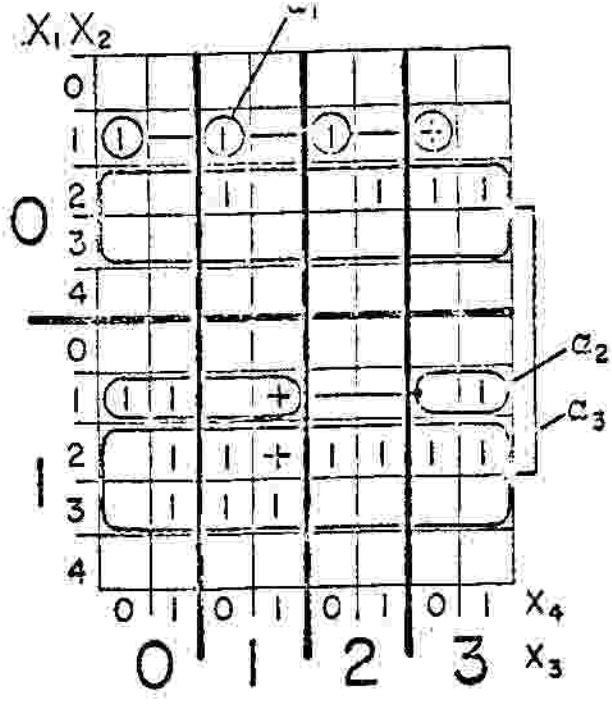
The theoretical basis and an algorithm for a solution of this problem (in somewhat more general formulation, where the clustering criterion is not limited to sparseness) is described in Michalski [24]. The algorithm is interactive, and its general structure is based on dynamic clustering method (Diday 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 have the minimum or subminum total sparseness (or, generally, the assumed cost criterion).

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 sires of their value sets 2, 5, 4 and 2, respectively. The space is represented as a diagram, where each cell represents an event.



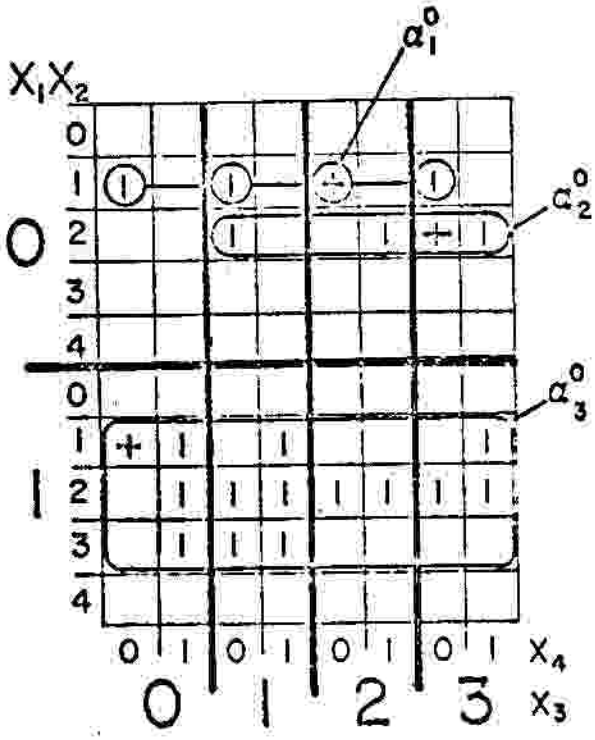
a. ITERATION 1

Sparseness = 18

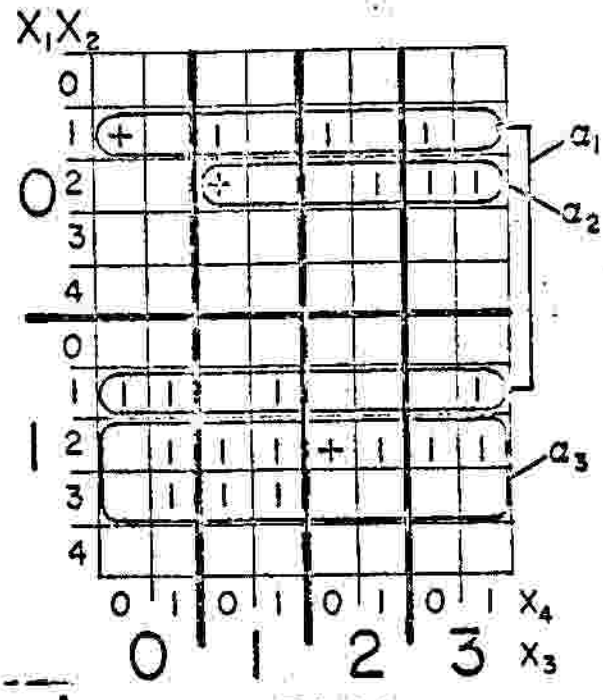


b. ITERATION 2

Sparseness = 20



c. ITERATION 3
(Optimal solution)
Sparseness = 12



d. ITERATION 4

Sparseness = 16

Cells marked by 1 represent data events, remaining cells segment unobserved events. Figure 3a also 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:

$$a_1^o = [x_1 = 0][x_2 = 1][x_4 = 0]$$

$$a_2^o = [x_1 = 0][x_2 = 2][x_3 = 1..3]$$

$$a_3^o = [x_1 = 1][x_2 = 1..3]$$

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.

5. SUMMARY

We have presented a view of inductive inference as a process of generalization and simplification of symbolic descriptions. The process is conducted by applying *generalization rules* and *problem environment rules* (representing problem specific knowledge) to the initial and intermediate descriptions. It is shown that both, learning from examples and learning from observation can be viewed this way.

Learning from examples is described as a problem of 'conceptual clustering', defined as a problem of partitioning a collection of entities into clusters (or, generally determining a structure of clusters) such that each cluster represents a concept selected from a predefined space of concepts.

ACKNOWLEDGMENT

A partial support of this research was provided by the National Science Foundation under grant MCS-79-06614. The author is grateful to Bob Stepp for useful discussions and proofreading the paper.

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.

APPENDIX 1

Definition of variable-valued logic calculus V_{21}

Data rules, hypotheses, problem environment descriptions, and generalization rules are all expressed using the same formalism, that of variable-valued logic calculus VL_{21}.* VL_{21} is an extension of predicate calculus designed to facilitate a compact and uniform expression of descriptions of different degrees and different types of generalization. The formalism also provides a simple linguistic interpretation of descriptions without losing the precision of the conventional predicate calculus.

There are three major differences between VL_{21} and the first order predicate calculus:

1. In place of predicates, it uses *selectors* (or *relational statements*) as basic operands. A selector, in the most general form, specifies a relationship between one or more atomic functions and other atomic functions or constants. A common form of a selector is a test to ascertain whether the value of an atomic function is a specific constant or is a member of a set of constants.

   The selectors represent compactly certain types of logical relationships which can not be directly represented in FOPC but which are common in human descriptions. They are particularly useful for representing changes in the degree of generality of descriptions and for syntactically uniform treatment of descriptors of different types.

*VL_{21} is a subset of a more complete system VL_2, which is a many valued-logic extension of predicate calculus.

2. Each atomic function (a variable, a predicate, a function) is assigned a value set (domain), from which it draws values, together with a characterization of the structure of the value set.

This feature facilitates a representation of the semantics of the problem and the application of generalization rules appropriat to the type of descriptors.

3. An expression in VL_{21} can have a truth status: TRUE, FALSE or ? (UNKNOWN).

The truth-status '?' provides an interpretation of a VL_{21} description in the situation, when, e.g., outcomes of some measurements are not known.

Definition 1: An *atomic function* is a variable, or a function symbol followed by a pair of parentheses which enclose a sequence of atomic functions and/or constants. Atomic functions which have a defined interpretation in the problem under consideration are called *descriptors*.

A *constant* differs from a variable or a function symbol in that its value set is empty. If confusion is possible, a constant is typed in quotes.

Examples

Constants: 2 * red

Atomic forms: x_1 color(box) on-top(p1,p2) $((x_1, g(x_2))$

Exemplary Value sets: $D(x_1) = \{0, 1, \ldots, 10\}$

D(color) = {red, blue,...}

D(on-top) = {true, false}

$D(f) = \{0,1,\ldots, 20\}$

Definition 2: A *selector* is a form

[L / R]

where L - called *referee*, is an atomic function, functions separated by '.'. (The operator '.' is called the *internal conjunction*.)

// - is one of the following relational operators:

$$= \neq \geq < \leq >$$

R - called *reference*, is a constant or atomic function, or a sequence of constants or atomic functions separated by operator ',' or '..'. (The operators ',' and '..' are called the *internal disjunction*, and the *range operator*, respectively).

A selector in which the referee L is a simple atomic function and the reference R is a single constant is called an *elementary selector*. The selector has truth-status TRUE (or FALSE) with regard to a situation if the situation *satisfies* (*does not satisfy*) the selector, i.e., if the referee L is (is not) related by // to the reference R. The selector has the truth-status '?' (and is interpreted as being a *question*), if there is not sufficient information about the values of descriptors in L for the given situation. To simplify the exposition, instead of giving a definition of what it means that 'L is related by // to R', we will simply explain this by examples.

		linguistic description
(i)	[color(box1) = white]	color of box1 is white
(ii)	[length(box1) ≥ 2]	length of box1 is greater than or eq
(iii)	[weight(box1) = 2..5]	weight of box1 is between 2 and 5,
(iv)	[blood-type (P1) = O,A,B]	blood-type of P1 is O or A or B
(v)	[on-top(box1, box2) = T] or simply [on-top(box1, box2)]	box1 is on top of box2
(vi)	[above(box1, box2) = 3"]	box 1 is 3" above box2
(viii)	[weight(box1) > weight (box3)]	the weight of box1 is greater than th weightof box3
(ix)	[length(box1) · length (box2) = 3]*	the length of box1 and box2 is 3
(x)	[type(p_1) · type (P_2) = A,B]	the type of P_1 and the type of P_2 is either A or B.

Note the direct correspondence of the selectors to linguistic descriptions. Note also that some selectors can not be expressed in FOPC in a (pragmatically) equivalent form (e.g., (iv), (ix), (x)).

*This expression is equivalent to [length(box1)=3][length(box2)=3].

A VL_{21} expression (or, here, simply VL expression) is defined by the following rules:

(i) A constant TRUE, FALSE or '?' is a VL expression

(ii) A selector is a VL expression

(iii) If V, V_1 and V_2 are VL expressions then so are:

(V)	formula in parentheses
$\neg V$	inverse
$V_1 \wedge V_2$ or $V_1 V_2$	conjunction
$V_1 \vee V_2$	disjunction
$V_1 \underline{\vee} V_2$	exclusive disjunction
$V_1 \diagdown V_2$	exception
$V_1 \Leftrightarrow V_2$	metaimplication

where $\Rightarrow \in \{\rightarrow, \leftrightarrow, ::>, \Rightarrow, \models, \vdash\}$
(implication, equivalence, decision assignment, inference, generalization, semantical equivalence)

$\exists x_1, x_2, \ldots, x_k(V)$	existentially quantified expression
$\forall x_1, x_2, \ldots, x_k(V)$	universally quantified expression

A VL formula can have truth-status TRUE (T), FALSE (F) or UNKNOWN(?). The interpretation given to connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, is defined in Fig. A1. (This interpretation is consistent with Kleen-Körner 3-valued logic). An expression with the operator $\Rightarrow$, $\models$ or $\vdash$ is assumed to always have the truth-status TRUE and with operator ::>, TRUE or ?. Operators $\diagdown$, $\underline{\vee}$, and $\leftrightarrow$ are interpreted:

$V_1 \diagdown V_2$ is equivalent to $V_1(\neg V_2)$

$V_1 \underline{\vee} V_2$ is equivalent $(V_1 \vee V_2) \diagdown V_1 V_2$

$V_1 \leftrightarrow V_2$ is equivalent to $(V_1 \rightarrow V_2)(V_2 \rightarrow V_1)$

The truth-status of

$\exists x(V)$ is

- TRUE (FALSE) if, there exists (does not exist) a value of x which makes the truth-status of V equal TRUE
- ? if it is not known whether there exists . . .

¬	
?	?
F	T
T	F

∧	?	F	T
?	?	F	?
F	F	F	F
T	?	F	T

∨	?	F	T
?	?	?	T
F	?	F	T
T	T	T	T

→	?	F	T
?	?	?	T
F	T	T	T
T	?	F	T

DEFINITION OF CONNECTIVES
¬, ∧, ∨ AND →
IN VL_{21}

Figure A1

$\forall x(V)$ is { TRUE (FALSE) if for every value of x the truth-status of V is (is not) TRUE
? if it is not known whether for every . . .

A constant * ('irrelevant') is introduced to substitute for R, in a selector [L = R], when R is the sequence of all possible values the L can take.

A VL expression in the form

$$QF_1, QF_2, \ldots (P_1 \vee P_2 \vee \ldots \vee P_1)$$

where QF_i is a quantifier form $\exists x_1, x_2, \ldots$ or $\forall x_1, x_2, \ldots$ and P_i is a conjunction of selectors (a *term*), is called a *disjunctive simple* VL expression (a *DVL* expression).

To make possible to use a name substitution operation in VL_{21}, the following notation is adopted:

- If FORMULA is an arbitrary VL_{21} expression then V : <FORMULA> assigns name V to the FORMULA.
- If FORMULA is a VL_{21} expression containing quantified variables P_1, P_2, ..., P_k, and V is the name of the expression, then

$$P_i(V)$$

denotes the quantified variable P_i in the FORMULA.

This construct enables one to refer to any quantified variables inside of any VL_{21} expression.

APPENDIX 2

Outline of the Top Level Algorithm of INDUCE 1.1.

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 FO. This 'rule growing' process is illustrated in fig. A2.

5. The obtained c-expressions, and c-expressions in FO, are transformed to two sets E1 and EO, 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 EO (the procedure is a version of AQVAL/1 program Larson & Michalski 75j).

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 AQVAL/1 procedure).
4. generation of new descriptors and new selectors

Program INDUCE 1.1 has been implemented in PASCAL (for Cyber 175 and DEC 10); its complete description is given in (Larson[31], Dietterich [32].