INDUCTIVE LEARNING AS RULE–GUIDED GENERALIZATION OF SYMBOLIC DESCRIPTIONS: A THEORY AND IMPLEMENTATION

by

Ryszard S. Michalski

Chapter 24 in Automatic Program Construction Techniques, MacMillan Publishing Company, Alan W. Biermann, Gerard Guiho, Yves Kodratoff, (Editors), New York, 1984.

CHAPTER 24

Inductive Learning as Rule-Guided Generalization of Symbolic Descriptions: A Theory and Implementation

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

Abstract

The theory presented here treats inductive learning as a process of generalizing symbolic descriptions, under the guidance of *generalization* rules and *background knowledge* rules. This approach unifies various types of inductive learning, such as learning from examples and learning from observation.

Two inductive learning programs are presented: INDUCE 1.1 — for learning structural descriptions from examples, and CLUSTER/PAF — for learning taxonomic descriptions (conceptual clustering). The latter program partitions a given collection of entities (objects, computational processes, observations, etc.) into clusters, such that each cluster is described by a single conjunctive statement and the obtained assembly of clusters satisfies an assumed criterion of preference.

The presented methodology can be useful for an automated determination of complete and correct program specification for computer-aided decision making, for knowledge acquisition in expert systems, and the conceptual analysis of complex data.

A. 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 one makes inductive assertions about some piece of reality, there is no natural limit to the level of detail and 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 [1974], Davis [1976]) or as a semantic net (Brachman [1978]). 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 also provide a means for detecting regularities in data bases and knowledge bases. 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 *et al.* [1980]), have indicated that rule-learning from examples is not only feasible, but in certain aspects is preferable.

Another potential applicaton of 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.

An important sub-area of computer inductive inference is automatic programming (e.g., Shaw *et al.* [1975], Jouannaud *et al.* [1979], Burstall *et al.* [1977], Biermann [1978], Smith [1980], and Pettorossi [1980]). 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"). In this case, the *postulate of human* comprehensibility, mentioned below, is of lesser importance. Quite similar to research on automatic programming is research on grammatical inference (e.g., Biermann and Feldman [1972], Yau and Fu [1978]) 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 typically not computational processes in a form of a logical-type expression (e.g., a specification of the program or a classification rule). 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 the following *comprehensibility postulate:*

The results of computer inductive learning 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 use both quantitative and qualitative information in an integrated fashion.

This postulate implies that a single description 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 [1977], which is to develop techniques for *conceptual interface* and *knowledge refinement.*

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

B. Computer Induction as Generalization and Simplification of Symbolic Descriptions

B.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 to generate and validate plausible general descriptions (inductive assertions or hypotheses) that 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 hypothesis 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 [1962]) 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 *observational assertions (data 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} (this association is denoted by the symbol ::>):

$$C_{11} ::> K_1,\ C_{12} ::> K_1 \ldots\ldots C_{1t1} ::> K_1$$
$$C_{21} ::> K_2,\ C_{22} ::> K_2 \ldots\ldots C_{2t2} ::> K_2$$

$$C_{m1} ::> K_m,\ C_{m2} ::> K_m \ldots\ldots C_{mtm} ::> K_m$$

Descriptions C_{ij} can be symbolic specifications of conditions satisfied by given situations, 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, predicate calculus, or a calculus specially developed for inductive inference, such as variable-valued logic systems VL_1 (Michalski [1973]) or VL_2 (Michalski [1978]).

(b) a set of *background knowledge rules* defining information relevant to the problem under consideration. This includes definitions of value sets of all descriptors* used in the input rules, the properties of descriptors and their interrelationships and any "world knowledge" relevant to the problem. The background knowledge also includes a *preference (or optimality)* criterion, which for any two sets of symbolic descriptions of the same generalization class specifies which one is preferable, or that they are equivalent with regard to this criterion.

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

$$\begin{array}{l} C'_{11} ::> K_1,\ C'_{12} ::> K_1,\ \cdots\ C'_{1r_1} ::> K_1 \\ C'_{21} ::> K_2,\ C'_{22} ::> K_2,\ \cdots\ C'_{2r_2} ::> K_2 \\ \cdot \\ \cdot \\ C'_{m1} ::> K_m,\ C'_{m2} ::> K_m,\ \cdots\ C'_{mr_m} ::> K_m \\ \text{where } r_m \leqslant t_m, \end{array}$$

which is the *most preferred* among all sets of rules in the assumed format, that do not contradict the *background knowledge* rules, and 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 that satisfies a data rule of some generalization class either satisfies an inductive assertion of the same class, or does not satisfy any inductive assertion. A set of inductive assertions is *complete* with regard to data rules, if any situation that satisfies some data rules also satisfies some inductive assertion.

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

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 background knowledge 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 [1978]).

B.2 Types of Inductive Learning

We distinguish two major types of inductive learning:

I. Learning from examples

Within this type, three subclasses of problems were given the most attention:

a. concept acquisition, or learning a *characteristic description* of a class of entities,
b. classification learning, or learning *discriminant descriptions* of classes of objects,
c. sequence prediction, or discovery of a rule that generates a given sequence of entities.

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

II. Learning from observation

Under this type we distinguish:

a. conceptual clustering, i.e., discovery of conceptual structure underlying a collection of entities,
b. pattern discovery,
c. theory formation.

Most of the research on computer inductive learning 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 [1970], Vere [1975], Hayes-Roth [1976]). 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 by the form of inductive assertion (e.g., that it should be a most specific conjunctive generalization within the given notational framework, as in (Hayes-Roth [1976]) and (Vere [1975]), or by the assumed *degree of generality* (Stepp [1978]). When there are negative examples the concept of *near miss* (Winston [1970]) can be used to effectively determine the limit of generalization.

A general problem of type Ia is to learn a *characteristic description* (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. In this case, the individual descriptions C_{ij} are generalized so long as it leads to their simplification and preserves the condition of consistency (e.g., Michalski [1980]). Obtained inductive assertions are *discriminant descriptions,* which permit one to distinguish one recognition class from all other assumed classes. A discriminant 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 considered in Simon and Lea [1973]), or nondeterministic, as in the card game EULESIS (Dietterich [1980]). 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 characterization of a collection of entities. In particular, such characterization can be a partition of the collection into clusters representing certain concepts ("conceptual clustering," Michalski [1980], Michalski and Stepp [1983]). In this case, data descriptions in (1) represent 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 [1975], Fikes *et al.* [1972], Banerji [1977], Cohen [1977], Hayes-Roth *et al.* [1978], Vere [1975]).

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 [1978]). It is a much richer language than VL_1, including 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 of different types (measured on different scales). The handling of descriptors each according to its type in the process of generalization is one of the significant aspects of our approach to induction.

B.3 Relevance 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. Two specific questions here are how relevant the variables in the input data must be to the problem, and how the variables in inductive assertions relate to these input variables.

We will distinguish three cases:

1. The input data consist 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 consist of descriptions of objects as in case 1, but the descriptions may involve a relatively large number of irrelevant variables in addition to relevant variables. The machine is to determine a soluton 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. Among irrelevant variables, they must include, however, also variables whose certain transformations (e.g., represented by mathematical expressions or intermediate logical formulas) are relevant derived 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 for selecting relevant variables (thus, this case represents *selective induction.* The formal logic provides such mechanisms, and this fact is one of the advantages of logic-based 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 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 *et al.* [1977], Larson [1977]) and in the newer improved version INDUCE-1.1 (Dietterich [1978]).

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 inductive assertions are multi-level or recursive descriptions.

But this is not the only interesting case. Derived descriptors in the inductive assertions may be any arbitrary, fixed (i.e., not learned) transformations of the input descriptors, specified by a mathematical formula, a computer program, or, implemented in hardware (e.g., the hardware implementation of fast Fourier transform). Their specification may require a language quite different from the accepted formal descriptive language. To determine these descriptors by learning, in the same fashion as the inductive assertions, 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.

B.4 The Background Knowledge 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, and b) *specification of the problem background knowledge* (which includes a *preference criterion).* We will briefly discuss each of these topics.

B.4.1. 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} *complexes),* defined as logical products of VL_{21} selectors, with zero or more quantifiers in front (the logical product is represented by concatenation). For example, a C'_{ij} can be:

$$\exists P1,P2\ [color(P1) = red \vee blue][weight(P1) > weight(P2)]$$
$$[length(P2) = 3..8][ontop(P1,P2)]\&\quad [shape(P1)\ \&\ shape(P2) = box]$$

(see Appendix 1 for explanation)

Since selectors can include internal disjunction (see Appendix 1) 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

$$(C1 \vee C2 \vee ...) \setminus\!\!/ C \tag{3}$$

where C, C1, C2, ... 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. Recently Vere [1978] proposed an algorithm for handling such assertions in the framework of conventional conjunctive statements.

- Implicative assertions

$$C(C_1 \rightarrow C_2) \tag{4}$$

which consist of a context condition C and an implication $C_1 \rightarrow C_2$, which states that properties in C_2 hold only if C_1 is true.

Production rules used in knowledge-based inference systems are a special case of (4), when C 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 represent a wider class of specific assertions.

Various aspects of the last problem within a less general framework were studied, e.g., by Hedrick [1974].

- Case assertions

$$([f = R_1] \rightarrow C_1), ([f = R_2] \rightarrow C_2), \cdots \qquad (5)$$

where $R_1, R_2 \cdots$ are pairwise disjoint sets.

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

B.4.2 Specification of the background knowledge

The background knowledge is defined by the specifying the types of the descriptors and their value sets, the interrelationships between descriptors, rules for generating new descriptors, the preference criterion and any other information relevant to the problem.

- 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 (totally) 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 (e.g. 'a person does not smoke'), then the variable x_3 is 'not applicable' (x_3 — the brand of cigarettes the person smokes). This is represented by a rule:

$$[x_1 = 0] \rightarrow [x_3 = \mathrm{NA}]$$

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:

$$\mathrm{TRUE} \rightarrow \forall\, x_1,x_2[f(x_1,x_2) > g(x_1,x_2)]$$

3. Properties of Predicate Functions

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

$$\forall x_1,x_2,x_3([\mathrm{left}(x_1,x_2)][\mathrm{left}(x_2,x_3)] \rightarrow [\mathrm{left}(x_1,x_3)])$$

Other types of relationships characteristic for the problem environment can be represented similarly.

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

B.4.3 The preference criterion

The preference criterion 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 hypotheses. The dimensions may be, e.g., the number of operators in the set of inductive assertions, the quantity of information required to encode the hypothesis using operators from an a priori defined set (Coulon and Kayser [1978]), 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 'lexicograpic functional' consists of an ordered list of criteria measuring hypothesis quality and a list of 'tolerances' for these criteria (Michalski [1973]).

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 (by maximizing or minimizing the number of selectors in the inductive assertions, respectively).

C. Generalization Rules

The transformation from data rules (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 in the same generalization class into a new description of the same class which is *equivalent* or *more general* than the initial set of descriptions.

A description

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

is *equivalent* to a set

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

if any *event* (a description of an object or situation) which satisfies at least one of the V_i, $i = 1,2, \cdots$, 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 generated inductive assertions are useful or plausible.

We have formalized several generalization rules, both for *selective* and *constructive induction.* Selective induction differs from constructive induction in that selective does not generate any new descriptors in the generalization process. (The notation $D_1 |< D_2$ specifies that D_2 is more general than D_1).

Selective generalization:

(i) *The extending reference rule*

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

where L is an atomic function

$R_2 \supset R_1$, and R_1,R_2 are subsets of the domain,
Dom(L) of descriptor L.
V - a context description (concatenation means conjunction)

This is a generally applicable rule; the type of descriptor L does not matter. For example, the description: 'objects that are blue or red' is more general than 'objects that are red'.

(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 therefore can be removed.

(iii) The *closing interval* rule

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

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 two normal 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

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

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 (i.e., is the most specific common generalization of nodes a,b, ... i).

The rule is applicable only to selectors involving structured descriptors. This rule has been used, e.g., in (Winston [1970], Hedrick [1974], Lenat [1976]).

Example:

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

(v) The *extension against* rule

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

where $R_1 \cap R_2 = 0$

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

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

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 inductive learning methods employing predicate calculus.

Constructive Generalization Rules:

Constructive generalization rules generate generalized descriptions of events in terms of new descriptors which are functions of the original descriptors. They can be viewed as knowledge-based rules for generating new descriptors. Here are a few examples.

(i) The *counting rule*

$$V[\text{attribute}_1(P_1)=A]\ldots[\text{attribute}_1(P_k)=A][\text{attribute}_1(P_{k+1})\neq A]$$

$$\cdots[\text{attribute}_1(\text{Pqsubr})\neq A] ::> K \,|< V[\#P-\text{attribute}_1-A=k] ::> K$$

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 an attribute of P_i, e.g., color, size, texture, etc.

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

Example:

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

This 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 a transitive relation (e.g., relation 'above', 'left of', 'larger than', etc.) form a chain, i.e., are linearly ordered by the relation, the rule generates descriptors relating to specific objects in the chain. 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 rules, in the context of condition C, an ascending order of values of a linear descriptor x_i corresponds to an ascending (or 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, then the descriptor length(P) also takes on the increasing values. In such situations a two-argument predicate descriptor is generated:

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

or

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

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

$$\uparrow\uparrow(x_i,x_j) = \begin{cases} \text{True, if } r(x_i,x_j) \geqslant \tau \\ \text{False, otherwise} \end{cases}$$

(positive correlation)

$$\downarrow\downarrow(x_i,x_j) = \begin{cases} \text{True, if } r(x_i,x_j) \leqslant -\tau \\ \text{False, otherwise} \end{cases}$$

(negative correlation)

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

The concept of generalization rules is very useful for understanding and classifying different methods of induction (Dietterich and Michalski [1979]).

D. 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 Figure 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). Such a description should contain 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 *et al.* [1977], Larson [1977], Dietterich [1978]) which successfully solved this problem, and then we will discuss some problems for future research.

At the first step, the initial space of descriptors was determined. They were all descriptors which seem to be relevant for the posed discrimination problem.

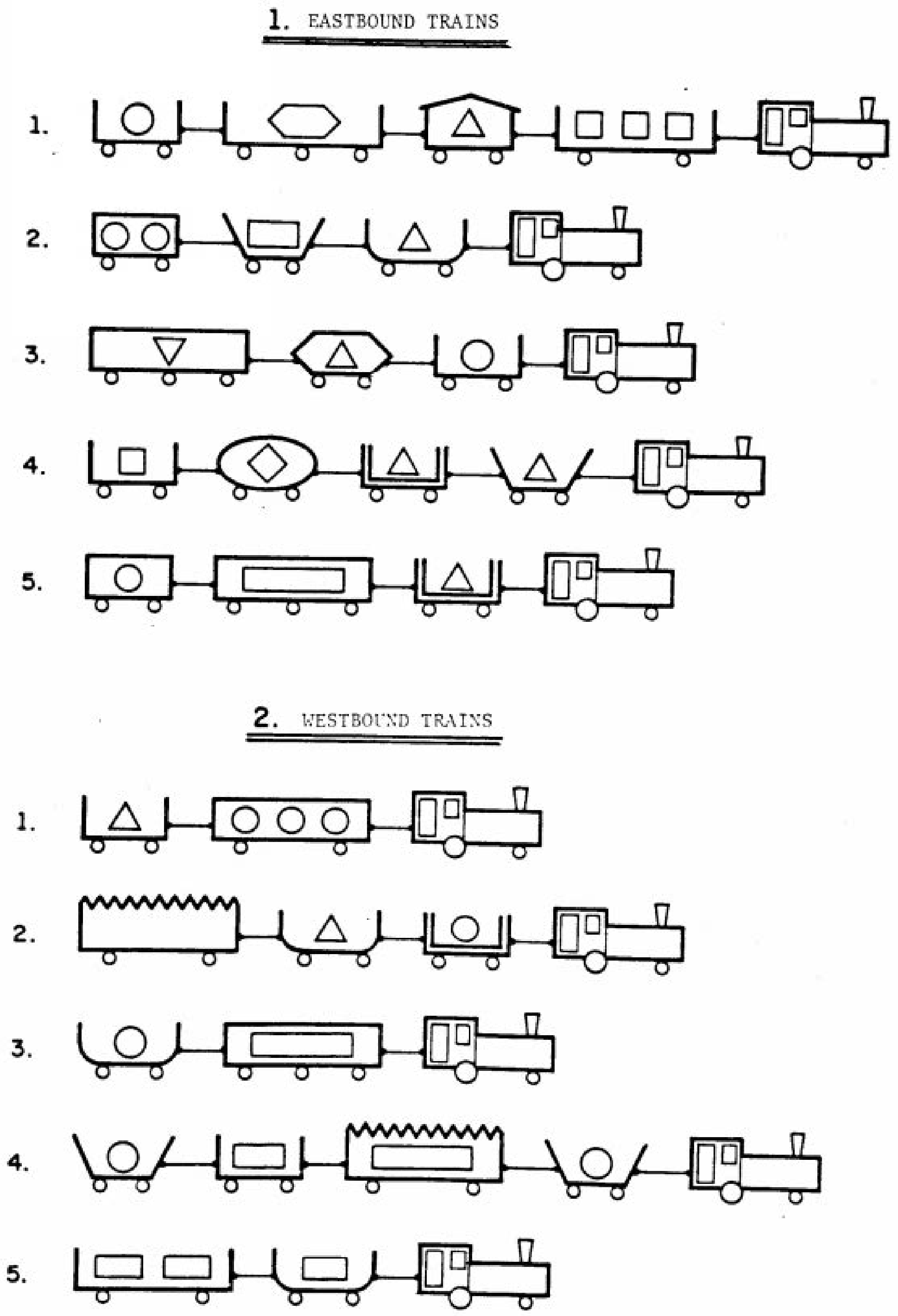


Figure 1. Find a rule distinguishing between these two classes of trains.

Among the eleven descriptors selected in total were:

position(car_i)	- the position of car_i, with engine being at position 1 *(a linear descriptor)*
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))*
contains(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 domain structure is hierarchical: plane figures may be circles or polygons; polygons may be hexagons, triangles, rectangles or squares.
nrpts-load(car_i)	- the number of parts in the load of car_i *(a linear descriptor)*
nr-wheels(car_i)	- the number of wheels on car_i *(a linear descriptor)*

At the next step, *data rules* 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 rule for the second eastbound train was:

$$\begin{aligned}&\exists\ car_1,car_2,car_3,car_4,load_1,load_2,\ldots\\&[\mathrm{infront}(car_1,car_2)][\mathrm{infront}(car_2,car_3)]\ldots[\mathrm{length}(car_1)\ =\mathrm{long}]\&\\&[\mathrm{car-shape}(car_1)=\mathrm{engine}][\mathrm{car-shape}(car_2)=\mathrm{V-shaped}]\&\\&[\mathrm{cont-load}(car_2,load_1)]\&\\&[\mathrm{load-shape}(load_1)=\mathrm{triangle}]\ldots[\mathrm{nrwheels}(car_3)=2]\ldots ::>[\mathrm{class}=\mathrm{Eastbound}]\end{aligned}$$

Background knowledge rules were used to define the structures of structured descriptors (arguments of descriptors are omitted as irrelevant here):

$$[\mathrm{car-shape}=\mathrm{open\ rctngl}\ \vee\ \mathrm{open\ trapezoid},\mathrm{U-shaped}\ \vee\ \mathrm{dbl\ open\ rctngl}]=>\\ [\mathrm{car-shape}=\mathrm{open\ top}] \tag{8}$$

$$[\mathrm{car-shape}=\mathrm{ellipse}\ \vee\ \mathrm{closed\ rctngl}\ \vee\ \mathrm{jagged\ top}\ \vee\ \mathrm{sloping\ top}]=>\\ [\mathrm{car-shape}=\mathrm{closed\ top}] \tag{9}$$

$$[\mathrm{load-shape}=\mathrm{hexagon}\ \vee\ \mathrm{triangle}\ \vee\ \mathrm{rectangle}]=>[\mathrm{load-shape}=\mathrm{polygon}] \tag{10}$$

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 INDUCE program produced the following inductive assertions*:

Eastbound trains:

$$\exists car_1[\text{length}(car_1) = \text{short}][\text{car-shape}(car_1) = \text{closed top}] ::> [\text{class=Eastbound}] \qquad (11)$$

It can be interpreted:

If a train contains a car which is short and has a closed top then it is an eastbound train.

Alternatively,

$$\exists car_1, car_2, load_1, load_2[\text{infront}(car_1, car_2)][\text{contains}(car_1, load_1)]\&$$
$$[\text{contains}(car_2, load_2)][\text{load-shape}(load_1) = \text{triangle}]\&$$
$$[\text{load-shape}(load_2) = \text{polygon}] ::> [\text{class= Eastbound}]$$

It can be interpreted:

If a train contains a car whose load is a triangle, and the load of the car behind is a polygon, then the train is eastbound. (12)

Westbound trains:

$$[\underline{\text{nr-cars=3}}] \vee car_1[\text{car - shape}(car_1) = \text{jagged-top}] ::> [\text{class=Westbound}]$$

Either a train has three cars or there is a car with jagged top. (13)

$$\exists car[\underline{\text{nr-cars-length-long=2}}][\text{position}(car) = 3][\text{shape}(car) = \text{open-top} \vee \text{jagged-top}]$$
$$::> [\text{class-Westbound}]$$

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

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

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 a useful aid to induction processes in various practical problems.

The descriptors underlined by the dotted lines ('nr-cars-length-long') are new descriptors, generated as a result of constructive induction. How were they generated? The constructive generalization rules are implemented as modules which scan the data rules and search for certain properties. For example, the *counting rule* of constructive generalization checks for each unary descriptor (e.g., length(car)) how many times a given 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 a 'relevance test', was included in the set of potentially useful selectors. The relevance test requires that a selector is satisfied by a sufficiently large number of positive examples and a sufficiently small number of negative examples (see the details later). 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 rule.*

Now, how does the whole program work? Various versions of the program were described in (Larson [1977], Michalski [1978], Dietterich [1978]). Appendix 2, provides 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 constructive generalization (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 the 'most relevant' selectors step-by-step) a set of consistent *c-formulas.* A simple relevance test for a selector is to have a large difference between the number of data rules covered by the selector in the given generalization class and the number of rules covered in other generalization classes.

C-formulas are represented as labelled graphs. Testing them for consistency (i.e., for null intersection of descriptions of different generalization classes) or for the *degree of coverage* of the given class is done by determining the subgraph isomorphism. By taking advantage of the labels on nodes and arcs, this operation was greatly simplified. Nevertheless, it consumes much time and space.

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 [1973]), and further generalization is done using AQVAL generalization procedure (Michalski and Larson [1978]). 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 string, and most of the operations done during this process are simple 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 transformation of the data structures representing the rules into more efficient form in the second part of the algorithm (after 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 that a selector from the initial data rules or one generated by constructive generalization rules that did not pass the 'relevance test', could still turn out to be very relevant, if its reference were appropriately generalized. Applying the above generalization rules to each selector represented as a graph structure (i.e., before the AQVAL procedure takes over) is, however, computationally very costly, and we decided against it in the INDUCE program. This problem will be aggravated when the number of constructive generalization rules generating derived descriptors is increased. We plan to seek solutions to this problem by designing a better *descriptor relevance test,* determining more adequate data structures for representing selectors and testing intersections of descriptions, and by applying problem background knowledge.

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

> When the train has 3 cars, the load of the first two cars is twice the total load of Westbound cars, and when the train is longer, 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 context of Westbound trains, but in the context of every possible train which is not Eastbound. This is a problem of determining a *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 [1978]). As we mentioned before (Sec. A.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 [1978]). 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 are examples of such formulas:

For Eastbound trains (some of these formulas may also cover Westbound trains):

$\exists$car[length(car)=short][car−shape(car)=closed top][nr−wheels=2]

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

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

(The first car in each train is 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 $\vee$ 3][load−shape(load)=triangle][contains(car,load)]

(The second or third car in each train contains triangle)

[nr−cars=4 $\vee$ 5]

(number of cars is 4 or 5)

$\forall$car[nr−wheels=2 $\vee$ 3]

(number of wheels in each car in every train is 2 or 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 the reader to paper by Diettrich [1980].

E. Learning From Observation

The major difference between problems of learning a characteristic description from examples (type IA), and problems of conceptual clustering in learning from observation (type II) is that in the latter 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', or unsupervised learning). 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 Figure 2.

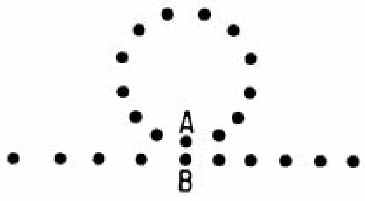


Figure 2.

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 Figure 2 do not constitute a complete circle and straight line, the obtained conceptual clusters represent generalizations of these data points. Consequently, conceptual clustering can be viewed as a form of generalization of symbolic descriptions, similarly as 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. A.1, the collection is considered 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. The restriction imposed by the problem is that the C-formulas logically intersect, as each cluster should be disjoint from other clusters.

We will describe here briefly an algorithm for determining 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 $=$ $\neq$
R_i is a subset of the value set of x_i.

A selector $[x_i = R_i]([x_i \neq R_i])$ is *satisfied* by a value of x_i, if this value is in relation $=$ $(\neq)$ with some (all) values from R_i. Such restricted c-formulas are called VL_1 *complexes* or, briefly, *complexes* (Michalski [1980]).

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 $a_i \in \mathrm{Dom}(x_i)$, and $\mathrm{Dom}(x_i)$ 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 observed 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 *absolute 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 [1980]. The algorithm is interactive, and its general structure is based on dynamic clustering method (Diday and Simon [1976]). Each step starts with k specially selected data events, called *seeds*. The seeds are treated as representatives 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 sub-minimum 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 sizes of their value sets 2, 5, 4 and 2, respectively. The space is represented as a diagram, where each cell represents an event. For example, cell marked e represents event(0,0,2,0).

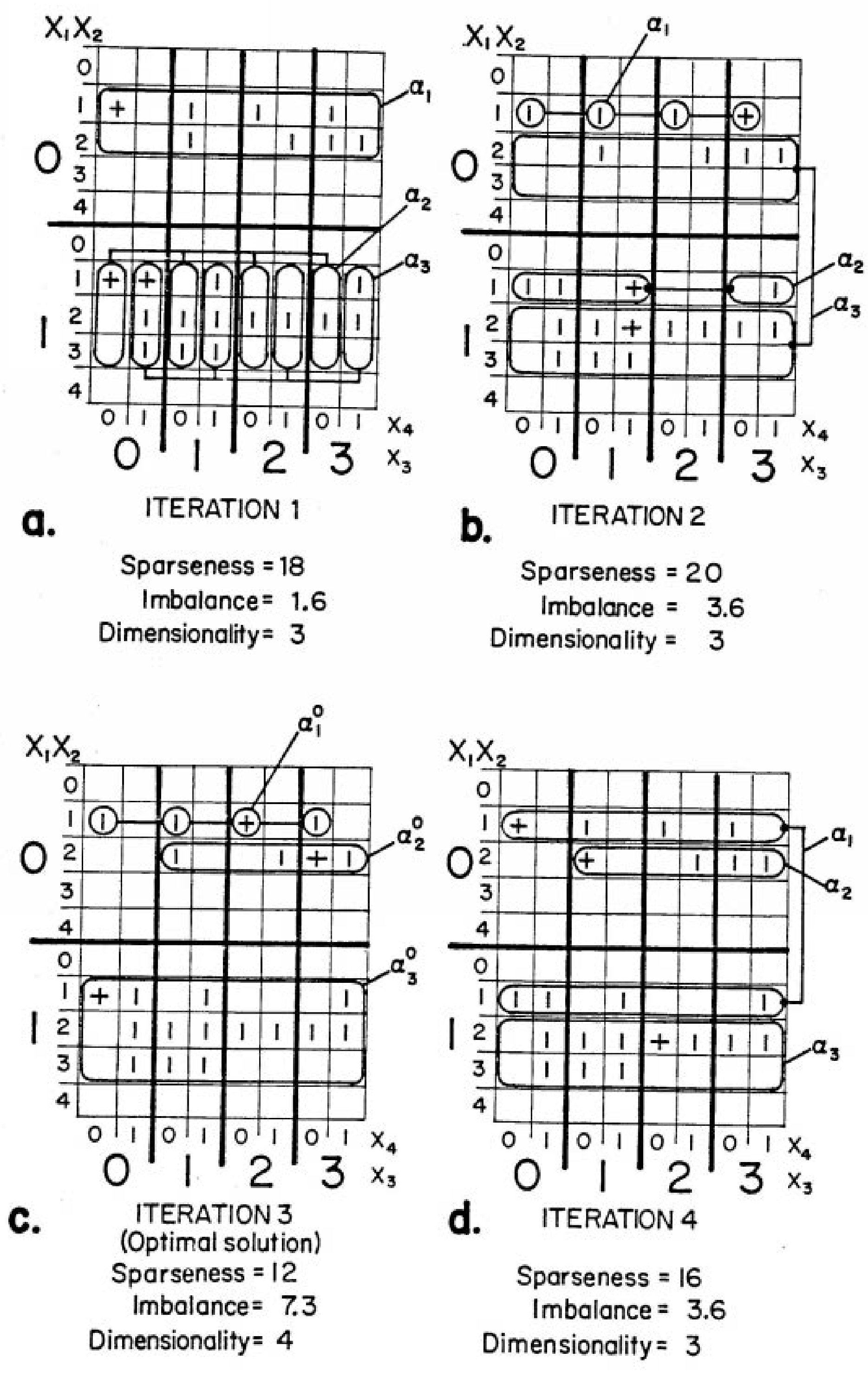


Figure 3. Iterative generation of a conjunctive clustering.

Cells marked by a vertical bar represent data events, while remaining cells represent unobserved events. Figure 3a also shows complexes obtained in the first iteration. Cells representing seed events in each iteration are marked by +. So in the first iteration, assuming k=3, three seeds were chosen: (0,1,0,0), (1,1,0,0) and (1,1,0,1). The complexes determined in this iteration are $\alpha, \alpha_2, \alpha_3$ (Figure 3a), with the total sparseness 18 (the total number of unmarked cells in these complexes). Figures 3b, c, and d show the results of the three consecutive iterations. The solution with the minimum sparseness is shown in Figure 3c. It consists of complexes:

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

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

$$\alpha_3^0 = [x_1 = 1][x_2 = 1..3]$$

This result was obtained by program CLUSTER/PAF* implementing the algorithm in PASCAL language on Cyber 175 (Michalski and Stepp [1982]).

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 contained known characteristic symptoms of the corresponding diseases.

Program CLUSTER/PAF is very general and could be useful for a variety of tasks that require a determination of intrinsically disjunctive descriptions. For example, such tasks are splitting a goal into subgoals, discovering useful subcases in a collection of computational processes, partitioning specific facts into conceptual categories, formulating cases in program specification.

F. Summary

We have presented a view of inductive inference as a process of generalization of symbolic descriptions. The process is conducted by applying *generalization rules* and the *background knowledge 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.

A form of learning from observation, called 'conceptual clustering' was described, which partitions a collection of entities into clusters, such that each cluster represents a certain concept. Presented methods for learning from examples (INDUCE) and automated conceptual clustering (CLUSTER/PAF) generate logic-style descriptions that are easy to comprehend and interpret by humans. Such descriptions can be viewed as formal specifications of programs solving tasks in the area of computer-based decision making and analysis of complex data.

G. Acknowledgement

A partial support of this research was provided by the National Science Foundation under grants MCS79-06614 and MCS82-05116. The author is grateful to Robert Stepp for useful discussions and for proofreading the paper.

* PAF stands for "Polish-American-French".

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 (FOPC):

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.

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 appropriate 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.

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.

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

Examples

Constants 2 * red
Atomic forms: x_1 color(box) on−top(pl,p2) ((x_1,g(x_2))
Exemplary value sets:

$$D(x_1) = \{0,1,\ldots, 10\}$$
$$D(\text{color}) = \{\text{red, blue}, \cdots\}$$
$$D(\text{on-top}) = \{\text{true, false}\}$$
$$D(f) = \{0,1,\ldots, 20\}$$

A *selector* is a form

$$[L \ \# \ R]$$

where
L - called *referee*, is an atomic function, or a sequence of atomic functions separated by '.' . (The operator '.' is called the *internal conjunction.)*
- is one of the following relational operators:

$$= \neq \ \geqslant \ < \ \geqslant \ <$$

R - called *reference*, is a constant or atomic function, or a sequence of constants or atomic functions separated by operator 'V' or '..' . (The operators 'V' 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 relatonal operator # to the reference R. The selector has the truth-status UNKNOWN (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 relational operator # to R', we will simply explain this by examples.

		Linguistic interpretation
(i)	[color(box1) = white]	color of box1 is white
(ii)	[length(box1) $\geqslant$ 2]	length of box1 is greater than or equal to 2
(iii)	[weight(box1) = 2..5]	weight of box1 is between 2 and 5,
(iv)	[blood-type(P1) = 0 V A V B]	blood-type of P1 is 0 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'']	box1 is 3'' above box2
(vii)	[weight(box1) > weight(box3)]	the weight of box1 is greater than the weight of box3
(viii)	[length(box1) ● length(box2) = 3]*	the length of box1 and box2 is 3
(ix)	[type(P_1) ● type(P_2) = A V 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 interpretations.* Note also that some selectors can not be expressed in FOPC in a (pragmatically) equivalent form (e.g., (iv), (ix)).

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

(i) A constant TRUE, FALSE, or 'UNKNOWN' 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 & V_2 or V_1V_2	conjunction
$V_1 \vee V_2$	disjunction
$V_1 \veebar V_2$	exception exclusive disjunction
$V_1 \twoheadrightarrow V_2$	metaimplication

where $\twoheadrightarrow \in \{\rightarrow, \leftrightarrow, ::>, =>, |<, |=, |>\}$

(implication, equivalence, decision assignment, inference, generalization, semantical equivalence, specialization)

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

A VL formula can have truth-status TRUE(T), FALSE(F) or UNKNOWN(?). The interpretation given to connectives $\bar{}$ ('not'), &,('and'),V('or'),$\rightarrow$, is defined in Fig. A1. (This interpretation is consistent with Kleen-Korner 3-valued logic.) An expression with the operator $=>$, $|<$ or $|=$ is assumed to always have the truth-status TRUE and with operator $::>$, TRUE or ?. Operators $\veebar$ and $\leftrightarrow$ are interpreted:

$$V_1 \veebar V_2 \text{ is equivalent to } V_1 \,\&\, \neg V_2 \vee \neg V_1 \,\&\, V_2$$

$$V_1 \leftrightarrow V_2 \text{ is equivalent to } (V_1 \rightarrow V_2)(V_2 \rightarrow V_1)$$

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

The truth-status of

$$\exists x(V) \text{ is } \begin{cases} \text{TRUE\{FALSE\}} & \text{if, there exist} \\ & \text{\{does not exist\} a value of x which makes} \\ & \text{the truth--status of V equal TRUE} \\ ? & \text{if it is not known whether there exist } \cdots \end{cases}$$

	¯
?	?
F	T
T	F

&	?	F	T
?	?	F	?
F	F	F	F
T	?	F	T

V	?	F	T
?	?	?	T
F	?	F	T
T	T	T	T

→	?	F	T
?	?	?	T
F	T	T	T
T	?	F	T

DEFINITION OF CONNECTIVES
¯, &, V, AND →
IN VL_{21}

Figure A1.

$$\forall x(V) \text{ is } \begin{cases} \text{TRUE \{FALSE\} if for every value of x} \\ \qquad\qquad \text{the truth-status of V is \{is not\} TRUE} \\ ? \qquad\qquad \text{if it is not known whether for every } \cdots \end{cases}$$

A VL expression in the form

$$QF_1, QF_2, \cdots \ (C_1 \vee C_2 \vee, \ldots, \vee C_l)$$

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

To make possible a name substitution operation, 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 $P_1, P_2, \ldots, P_k$, and V is the name of the expression, then

$$P_i - V$$

denotes the quantified variable P_i in the FORMULA.

The latter construct enables one to refer in one expression to quantified variables inside of other expressions.

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, whose 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 $QFx_1,x_2,...,$ 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 generalization 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 FO (the set F1 represents events to be *covered,* and set FO represents constraints, i.e., events not to be *covered).*

3. By application of inference rules representing background knowledge and constructive generalization rules, new selectors are generated. The most promising selectors (according to the preference 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 generated (Michalski and Stepp [1982]). 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 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 program Michalski and Larson [1978]).

   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 for the same generalization class.

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

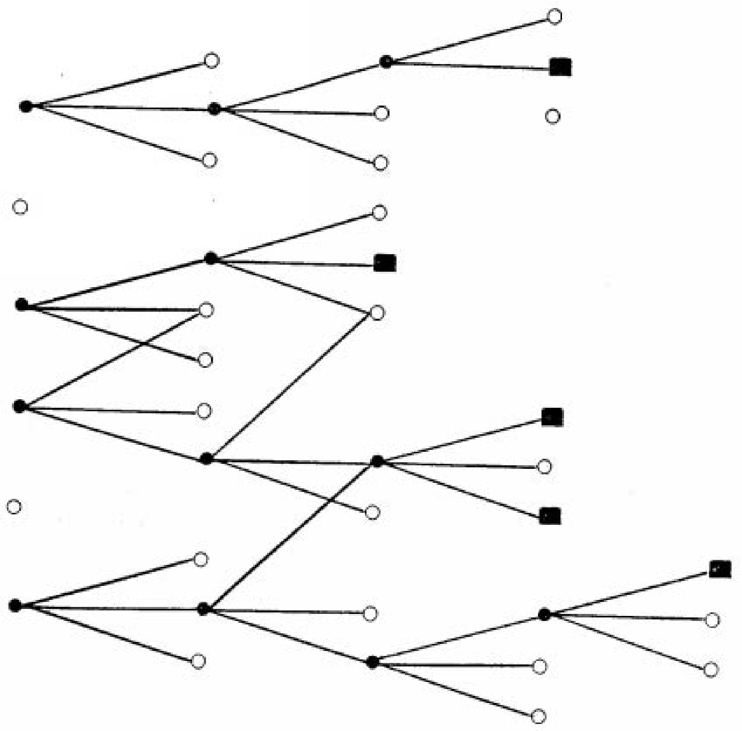
1. "Growing" rules.

2. Testing whether one c-expression is a generalization of ('covers') another c-expression. (This is done by testing for subgraph isomorphism).

3. Generalizating a c-expression by extending the selector references and forming irredundant c-expressions (includes application of AQVAL/1 procedure).

4. Generating new descriptors and new selectors.

Program INDUCE 1.1 has been implemented in PASCAL (for Cyber 175) its description is given in (Larson [1977] and Dietterich [1978]).



○ - 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 A2.

References

Banerji [1977]
R.B. Banerji, "Learning in structural description languages," Temple University Report to NSF Grant MCS 76−0−200 (1977).

Biermann [1978]
A.W. Biermann, "The inference of regular LISP programs from examples," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. SMC−8(8) (Aug. 1978) pp. 585−600.

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

Brachman [1978]
R.T. Brachman, "On the epistomological status of semantic networks," Report No. 3807, Bolt, Beranek and Newman (April 1978).

Burstall and Darlington [1977]
R.M. Burstall and J. Darlington, "A transformation system for developing recursive programs," *JACM, Vol. 24, No. 1* (1977) pp. 44−67.

Carnap [1962]
R. Carnap, "The aim of inductive logic," in *Logic, Methodology and Philosophy of Science,* E. Nagel, P. Suppes, and A. Tarski, eds., Stanford, California: Stanford University Press (1962) pp. 303−318.

Cohen [1977]
B.L. Cohen, "A powerful and efficient structural pattern recognition system," Artificial Intelligence, Vol. 9, No. 3 (December 1977).

Coulon and Kayser [1978]
D. Coulon and D. Kayser, "Learning criterion and inductive behavior," *Pattern Recognition,* Vol. 10, No. 1, pp. 19−25 (1978).

Davis [1976]
R. Davis, "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).

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

Dietterich [1980]
T.G. Dietterich, "A methodology of knolege layers for inducing descriptions of sequentially ordered events," Report No. 80−1024, Department of Computer Science, University of Illinois (May 1980).

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

Dietterich and Michalski [1979]
T. Dietterich and R.S. Michalski, "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).

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

Hayes-Roth [1976]
F. Hayes-Roth, "Patterns of induction and associated knowledge acquisition algorithms," Pattern Recognition and Artificial Intelligence, ed. C. Chen, Academic Press, New York (1976).

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

Hedrick [1974]
C.L. Hedrick, "A computer program to learn production systems using a semantic net," Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburg (July 1974).

Jouannaud and Kodratoff [1979]
J.P. Jouannaud and Y. Kodratoff, "Characterization of a class of functions synthesized from examples by a Summers-like method using a B.M.W. matching technique," *Sixth International Joint Conference on Artificial Intelligence* Tokyo (1979) pp. 440−447.

Larson [1977]
J. Larson, "INDUCE-1: An interactive inference program in VL_{21} logic system," Report No. UIUCDCS−R−77−876, Department of Computer Science, University of Illinois, Urbana, Illinois (May 1977).

Larson and Michalski [1977]
J. Larson and 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).

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

Michalski [1980]
R.S Michalski, "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.)

Michalski [1980]
R.S. Michalski, "Pattern recognition as rule-guided inductive inference," IEEE Trans. on Pattern Analysis and Machine Intelligence (July 1980).

Michalski [1978]
R.S. Michalski, "Pattern recognition as knowledge-guided computer induction," Report No. 78−927, Department of Computer Science, University of Illinois, Urbana, Illinois (June 1978).

Michalski [1973]
R.S. Michalski, "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).

Michalski and Chilausky [1980]
R.S. Michalski and R.L. Chilausky, "Learning by being told and learning from examples," *Policy Analysis and Information Systems,* Special Issue on Knowledge Acquisition and Induction, No.2 (1980)

Michalski and Larson [1978]
R.S. Michalski and J. Larson, "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).

Michalski and Stepp [1982]
R.S. Michalski and R. Stepp, "Revealing conceptual structure in data by inductive inference." In: Machine Intelligence 10, D. Michie, J.E. Hayes, Y.H. Pao (Eds.), Ellis Horwood, Ltd., New York, 1982, pp. 173−196.

Michie [1977]
D. Michie, "New face of artificial intelligence," *Information 3* (1977) pp. 5−11.

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

Morgan [1975]
C.G. Morgan, "Automated hypothesis generation using extended inductive resolution," Advance Papers of the 4th International Joint Conference on Artificial Intelligence, Vol. I, Tbilisi, Georgia (September 1975) pp. 351−356.

Pettorossi [1980]
A. Pettorossi, "An algorithm for reducing memory requirements in recursive programs using annotations," IBID.

Shaw, Swartout, and Green [1975]
D.E. Shaw, W.R. Swartout, and C.C. Green, "Inferring Lisp programs from examples," *Proceedings of the 4th International Conference on Artificial Intelligence,* Vol. I, pp. 351−356, Tbilisi (Sept. 1975).

Shortliffe [1974]
E.G. Shortliffe, "A rule based computer program for advising physicians antimicrobial therapy selection," Ph.D. Thesis, Computer Science Department, Stanford University (Oct. 1974).

Smith [1980]
D.R. Smith, "A survey of the synthesis of LISP programs from examples," International Workshop on Program Construction, Bonas (Sept. 1980).

Simon and Lea [1973]
H.A. Simon and G. Lea, "Problem solving and rule induction: A unified way," Carnegie-Mellon Complex Information Processing Working Paper No. 277, Revised June 14, 1973.

Stepp [1978]
R. Stepp, "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).

Vere [1975]
S.A. Vere, "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).

Winston [1970]
P.H. Winston, "Learning structural descriptions from examples," Technical Report AI TR–231, MIT AI Lab, Cambridge, Massachusetts (1970).

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