Pattern Recognition as Rule-Guided Inductive Inference

RYSZARD S. MICHALSKI

Abstract—The determination of pattern recognition rules is viewed as a problem of inductive inference, guided by generalization rules, which control the generalization process, and problem knowledge rules, which represent the underlying semantics relevant to the recognition problem under consideration. The paper formulates the theoretical framework and a method for inferring general and optimal (according to certain criteria) descriptions of object classes from examples of classification or partial descriptions. The language for expressing the class descriptions and the guidance rules is an extension of the first-order predicate calculus, called variable-valued logic calculus $V_{LV}$. $V_{LV}$ involves typed variables and contains several new operators especially suited for conducting inductive inference, such as selector, internal disjunction, internal conjunction, exception, and generalization.

Important aspects of the theory include:
1) a formulation of several kinds of generalization rules;
2) an ability to uniformly and adequately handle descriptors (i.e., variables, functions, and predicates) of different type (nominal, linear, and structured) and of different arity (i.e., different number of arguments);
3) an ability to generate new descriptors, which are derived from the initial descriptors through a rule-based system (i.e., an ability to conduct the so-called constructive induction);
4) an ability to use the semantics underlying the problem under consideration.

An experimental computer implementation of the method is briefly described and illustrated by an example.

Index Terms—Computer consulting systems, generalization methods, inductive inference, knowledge acquisition, learning from examples, many-valued logic, pattern recognition techniques, plausible inference, theory formation.

I. INTRODUCTION

A pattern recognition rule can be viewed as a rule

\[ \text{DESCRIPTION} \Rightarrow \text{RECOGNITION CLASS} \]  

which assigns a situation (an object, a process, etc.) to the recognition class, when the situation satisfies the description. In the decision theoretic approach the description is an analytical expression involving a set of numerical variables selected a priori. Variables spanning the decision space are treated uniformly, are usually assumed to be measured on at least an interval scale, and are desired to be relevant and independent characteristics of the objects. When the variables are strongly interconnected and/or the relevant object characteristics are not numerical variables but various relations among other variables, or among parts or subparts of objects, then the decision theoretical approach becomes inadequate. In such situations the structural approach can be useful.

In the structural (or syntactic) approach the description is a formal grammar (usually a phase-structure grammar) in which terminals are certain elementary parts of objects, called primitives. The types of relationships which can be expressed naturally in terms of a formal grammar are, however, quite limited. If the relevant characteristics include, for example, some numerical measurements in addition to relations and symbolic concepts, then grammars involving them are very cumbersome or inadequate. This is a strong limitation, because in many problems an adequate class description requires both numerical characterizations of objects and a specification of various relationships among properties of objects and/or of object parts, i.e., involve descriptors of mixed arity and measured on different scales. To partially overcome this limitation attributed grammars were proposed [1].

Both the decision theoretic approach and the syntactic approach make a little use of the underlying semantics of the problem under consideration, and therefore the scope of patterns they are able to discover is limited. Also they tend to produce "mathematical type" descriptions which are not easily comprehensible by humans, rather than "conceptual type" descriptions which human experts would develop observing the same data, and would prefer to use. Although in many applications human comprehensibility may not be important, in other applications (e.g., in expert computer consulting systems) it is crucial.

This paper presents results, still early and limited, of an attempt to develop a uniform conceptual framework and an implementation method which appropriately handles descriptors of different type, is able to use the semantics of the problem and satisfies the requirement of human comprehensibility. Another aspect of this method is that the final descriptions which it produces may involve new descriptors (variables or relations) which were not included in the initial characterization of objects. This is achieved through the application of "meta-rules" which represent the underlying knowledge of the problem at hand and of the properties of descriptors used in formulating the descriptions of exemplary data. The presented theory uses as a language for expressing the class descriptions.

1 "Arity"—The number of arguments of a descriptor. Unary descriptors are called attributes, or, generally, variables. Two or more argument descriptors with the value set \{TRUE, FALSE\} are called predicates.
and rules an extension of the first-order predicate calculus, called variable-valued logic system $VL_{21}$, and is most closely related to the body of work termed computer induction. The ability to develop new descriptors (variables, predicates, functions) in addition to those given a priori, places this work in the category of what we call constructive induction, as opposed to nonconstructive induction, in which the final descriptions relate only descriptors initially provided.

II. RELATED RESEARCH

It would be a very difficult task, requiring more space than provided, to characterize adequately various important contributions to computer induction. Therefore, we will make here only a very limited and certainly not adequate review of some more recent works.

A dissatisfaction with the early work on general methods of induction in the early sixties led some workers to concentrate on inductive tasks within a specific problem domain. For example, programs collectively called METADENDRAL [2] use a model-directed heuristic search to determine rules that describe the molecular structure of an unknown chemical compound from mass spectrometry data. Winston [3] describes a method for determining a graph description of simple block structures from examples. A program developed by Lenat [4] generates concepts (represented as collections of a priori defined properties) of elementary mathematics, under the guidance of a large body of heuristic rules. Soloway and Riseman [5] describe a method for creating multilevel descriptions of a part of a baseball game, starting with "snapshots" of the game, and using rules representing general knowledge of the game.

The programs such as those mentioned above usually incorporate a large body of task-specific knowledge and tend to perform quite well on tasks they were designed for. They demonstrate again that high performance requires specialized solutions. On the other hand, it is usually not easy to determine the general ideas they contribute to the understanding of inductive processes. Also, it is difficult to apply such methods directly to other problem areas.

A significant part of research has been concerned with determining patterns in sequences of symbols (e.g., Simon [6] and Waterman [7]). Simon [6] found that descriptions of such patterns consistently incorporate only a few basic relations: "some" and "next" between symbols, iterations between subpatterns, and hierarchical phrase structure. Gaines [8] developed a method for generating finite-state automata which approximate a given symbol string, and represent different tradeoffs between the complexity and expressiveness of the system. Shaw, Schwartz, and Green [9] developed a program for inferring Lisp code from a set of examples of Lisp statements. Also, Jouanaud, Guillo, and Treuille [10] have developed an interactive system which can infer a class of Lisp linear recursive functions from a set of examples.

The above works are related to the general subject of grammatical inference, i.e., inference of a grammar which may have produced a given set of strings. Early work in concerned with the inference of a phrase structure (e.g., Feldman et al. [11]). More recent work involves "multidimensional" grammars (e.g., work of Fu [12]).

In recent years there has been a new trend toward development of general methods of induction.

In previous papers the author and his collaborators [13]-[15] have described a methodology and computer programs for learning optimal discriminant descriptions of object classes from examples (in the framework of an extended propositional calculus with many-valued variables called $VL_{1}$). Examples are presented as sequences of attribute-value pairs. Each attribute has an associated value set and type. Work in a similar spirit, although more limited in scope, was reported by Stoffel [16] (the elementary statements used there are restricted to the "variable-value" forms, i.e., to elementary selectors as described in Section IV).

An early work which recognizes the need for logic-style descriptions for pattern recognition was done by Banerji [17]. A more recent continuation of this work is in Banerji and Cohen [19], who developed a logic-based description code utilizing Lisp-like notation.

An important aspect of induction, that of empirical abstraction, was studied by Zagarulis [25], who developed a general method of "strengthening hypotheses" by narrowing the uncertainty ranges of values of output variables. Hendrick [26] developed a method of learning of production systems describing symbol series using a semantic net of predefined concepts.

Many authors use a restricted form (usually quantifier-free) of the first-order predicate calculus (FOPC) or some equivalent notation as the formal framework for expressing descriptions and hypotheses. Morgan [20] describes a formal method of hypothesis generation, called resolution, which stems from deductive resolution principles. Various theoretical issues of induction in FOPC were considered by Plotkin [21]. Fikes, Hart, and Nilsson [22] describe an algorithm for generalizing robot plans. Hayes-Roth and McDermott (e.g., [23]), and also Verbeek [24], describe methods and computer programs for generating conjunctive descriptions of least generality (which they call "maximal abstractions"), of a set of objects represented by products of n-ary predicates. The rules of generalization which they use can be characterized as "dropping a condition" and "adding constants into variables" (see Section V-C).

This paper presents a theoretical framework for generalizing and optimizing descriptions of object classes in the form of decision rules. The decision rules can involve descriptors of three different types: nominal, linear, and structured; employ some new syntactic forms; and use problem knowledge for guiding induction and generating new descriptors. The formal notation is a modification and extension for FOPC, called variable-valued logic system $VL_{21}$. This formalism is more adequate than the traditional FOPC as a conceptual framework for describing the inductive processes under consideration. The paper is an extension and modification of the report [27], and stresses the conceptual principles.
of the induction method, rather than specific algorithms and implementation details. Most of the latter are described in [28]-[30].

III. Problem Statement

A VL transformation rule is defined as a rule

\[ \text{DESCRIPTION}_1 \rightarrow \text{DESCRIPTION}_2 \quad (2) \]

where \text{DESCRIPTION}_1 and \text{DESCRIPTION}_2 are expressions in VL system (Section IV) and \( \rightarrow \) stands for various transformation operators which define the meaning of the rule.

A DESCRIPTION may look like

\[ \exists p_1, p_2 (\text{on-top}(p_1, p_2)) [\text{size}(p_1) = 3.5] \]
\[ \Lambda [\text{color}(p_2) = \text{blue}, \text{yellow}, \text{red}] \]
\[ \Lambda [\text{length}(p_1) - \text{length}(p_2) = \text{small}] \]

where

\text{the range operator}

\text{(after the equality sign)} denotes the internal disjunction,

\text{and}

\text{denotes the internal conjunction.}

(For explanation of notation see Section IV.)

We will consider here the following transformation operators.

i) \( \rightarrow \) The operator defines a decision rule. \text{DESCRIPTION}_2 specifies a decision (or a sequence of decisions) which is assigned to a situation which satisfies \text{DESCRIPTION}_1. In the application to pattern recognition, \text{DESCRIPTION}_2 defines the recognition class. If a situation does not satisfy the \text{DESCRIPTION}_1, the rule assigns to it a NULL decision.

ii) \( = \) The operator defines an inference rule. If a situation satisfies \text{DESCRIPTION}_1, the rule assigns the truth-status "TRUE" to \text{DESCRIPTION}_2, otherwise truth-status of \text{DESCRIPTION}_2 is "?". In an inference rule \text{DESCRIPTION}_1 is called the condition and \text{DESCRIPTION}_2 is called the consequence.

iii) \( \subseteq \) The operator denotes a generalization rule, which states that the \text{DESCRIPTION}_2 is more general than \text{DESCRIPTION}_1, i.e., the set of situations which satisfy \text{DESCRIPTION}_2 is a superset of the set of situations satisfying \text{DESCRIPTION}_1.

iv) \( \Rightarrow \) The operator denotes an equivalence preserving rule, i.e., when the above mentioned sets are equal. The rule is a special case of a generalization rule.

The problem considered in this paper is defined as follows.

\text{- Given are the following.}

a) A set of VL decision rules, called data rules, which specify initial knowledge. \( C_{ij} \), about some situations (objects, processes, . . . ) and the recognition class, \( K \), associated with them:

\[ C_{11} \Rightarrow K_1, \ C_{12} \Rightarrow K_1 \quad \cdots \quad C_{1r_1} \Rightarrow K_1 \]
\[ C_{21} \Rightarrow K_2, \ C_{22} \Rightarrow K_2 \quad \cdots \quad C_{2r_2} \Rightarrow K_2 \]
\[ \cdots \]
\[ C_{m1} \Rightarrow K_m, \ C_{m2} \Rightarrow K_m \quad \cdots \quad C_{mr_m} \Rightarrow K_m \quad (3) \]

b) The problem knowledge rules which represent the background knowledge about the recognition problem under consideration. This knowledge includes the type of each descriptor used in the data rules, its value set, the problem constraints, the relationship among descriptors that reflect the semantics of the problem and various constructive generalization rules (see Section V-C).

\text{- A preference criterion, which for any two "comparable" sets of decision rules specifies which one is more preferable, or states that they are equally preferable.}

- The problem is to determine, through an application of generalization rules and problem knowledge rules, a new set of decision rules called output rules or hypotheses:

\[ C'_{11} \Rightarrow K_1, \ C'_{11} \Rightarrow K_1 \quad \cdots \quad C'_{1r_1} \Rightarrow K_1 \]
\[ C'_{21} \Rightarrow K_2, \ C'_{21} \Rightarrow K_2 \quad \cdots \quad C'_{2r_2} \Rightarrow K_2 \]
\[ \cdots \]
\[ C'_{m1} \Rightarrow K_m, \ C'_{m2} \Rightarrow K_m \quad \cdots \quad C'_{mr_m} \Rightarrow K_m \quad (4) \]

which is most preferable among all sets of rules that with regard to the input rules are consistent and complete.

The output rules are consistent with regard to input rules, if for any situation to which the input rules assign a non-NULL class, the output rules assign to it the same class, or the NULL class.

The output rules are complete with regard to input rules, if for any situation to which the input rules assign a non-NULL class, the output rules also assign to it a non-NULL class.

It is easy to see that if the output rules are consistent and complete with regard to the input rules then they are semantically equivalent (i.e., assign the same decision to the same situation) or more general than the input rules (i.e., they may assign a non-NULL class to situations to which the input rules assign a NULL class).

From a given set of data rules it is usually possible to derive many different sets of output rules which are consistent and complete and which satisfy the problem constraints. The rule of the preference criterion is to select one (or a few alternative sets of rules) which is most desirable in the given problem domain. The preference criterion may refer to e.g.

- the computational simplicity (or complexity) of the rules,
- the cost of measuring the information needed for rule evaluation,
- the degree-of-fit to the data.

In this paper we accept the restriction that the \( C_{ij} \) and \( C'_{ij} \) are disjunctive simple VL expressions (Section IV). Such expressions have a very simple linguistic interpretation, and seem to be of interest to many applications.

IV. VL Expressions as Descriptions

A. Definition of VL

Data rules, hypotheses, problem knowledge rules, and generalization rules are all expressed using the same formalism. that of variable-valued logic calculus VL

\[ \text{VL} \]

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.

\[ \text{VL}_{21}^2 \text{is a subset of a more complete version VL}_{21} \text{under development.} \]
To make the paper self-contained, we will provide here a brief description of VL21.

There are three major differences between VL21 and the first order predicate calculus (FOPC).

1) In place of predicates, it uses selectors (or relational statements) as basic operands. A selector, in its most general form, specifies a relationship between one or more atomic functions and/or 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 cannot 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, and its type, which defines the structure of the value set (see Section V-B).

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

\[
\begin{align*}
&
\text{i) } [\text{color(box1)} = \text{white}] \\
&
\text{ii) } [\text{length(box1)} \geq 2] \\
&
\text{iii) } [\text{weight(box1)} = 2.5] \\
&
\text{iv) } [\text{blood-type(P1)} = 0, A, B] \\
&
\text{v) } [\text{on-top(box1, box2)} = \text{T}] \\
&\quad \text{or simply:} \\
&\quad \text{[on-top(box1, box2)]} \\
&\text{vi) } [\text{above(box1, box2)} = \text{3}] \\
&\text{vii) } [\text{weight(box1)} > \text{weight(box3)}] \\
&\text{viii) } [\text{length(box1)} \cdot \text{length(box2)} = 3] \\
&\text{ix) } [\text{type(p1)} \cdot \text{type(p2)} = \text{A, B}] \\
\end{align*}
\]

3) An expression in VL21 can have a truth status: TRUE, FALSE, of ? (UNKNOWN).

The truth-status "?" provides an interpretation of a VL21 description in the situation, when, e.g., outcomes of some measurements are not known.

Definition I: 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 selectors.

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, x, color(box)

Atomic forms: on-top(p1, p2) \( f(x_1, 2, x_2) \)

Exemplary value sets:

\[
\begin{align*}
&\text{D}(x_1) = \{0, 1, \ldots, 10\} \\
&\text{D(color)} = \{\text{red}, \text{blue}, \ldots\} \\
&\text{D(on-top)} = \{\text{true}, \text{false}\} \\
&\text{D(r)} = \{0, 1, \ldots, 20\}
\end{align*}
\]

Definition 2: A selector is a form:

\[
[L \neq R] 
\]

L, called referent, is an atomic function, or a sequence of atomic functions separated by \( \ldots \). (The operator \( \ldots \) is called the internal conjunction.)

\# is one of the following relational operators:

\[
\begin{align*}
&= \neq < > < .
\end{align*}
\]

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

A selector in which the referent 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 referent 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. Instead of giving a definition of what it means that "L is related by \# to R," we will simply explain this by examples. (See Section V-A for more details.)

Linguistic description

\[
\begin{align*}
&\text{color of box1 is white} \\
&\text{length of box1 is greater than or equal to 2} \\
&\text{weight of box1 is between 2 and 5,} \\
&\text{blood-type of P1 is 0 or A or B} \\
&\text{box1 is on top of box2} \\
&\text{box1 is 3\# above box2} \\
&\text{the weight of box1 is greater than the weight of box3} \\
&\text{the length of box1 and box2 is 3} \\
&\text{the type of P1 and the type of P2 is either A or B.}
\end{align*}
\]

Note the direct correspondence of the selectors to linguistic descriptions. Note also that some selectors cannot be expressed in FOPC in a grammatically equivalent form [e.g., \( (V), (V, x) \)].

A VL21 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 \( V_1 \) and \( V_2 \) are VL expressions then so are

\[
\begin{align*}
&\neg V_1 \\
&V_1 \land V_2 \quad \text{or} \quad V_1 \lor V_2 \\
&V_1 \land V_2 \\
&V_1 \lor V_2 \\
&V_1 \implies V_2 \\
&\text{formula in parentheses} \\
&\text{inverse} \\
&\text{conjunction} \\
&\text{disjunction} \\
&\text{exclusion (V1 except when V2)} \\
&\text{metamorphication} \\
&\text{where } \implies \in \{=, =, >, =, \leq, \geq, \}
\end{align*}
\]

Note: this expression is equivalent to \([\text{length(box1)} = 5] \land \text{length(box2)} = 3\).
is used by applying it to situations. A situation is, in general, a source of information about values of variables and atomic functions in description 1 (the condition part of the rule). A situation can, e.g., be a database storing values of variables and procedures for evaluating atomic functions, or it can be an object on which various tests are performed to obtain these values.

A decision rule is viewed as a special case of an inference rule, when description 2 (the consequence or decision part of the rule) is a constant, an elementary selector, or a product of elementary selectors involving decision variables (i.e., the description uniquely defines a decision or a sequence of decisions). The truth status of the condition and decision part of a rule, before applying it to a situation, is assumed to be unknown.

Let $Q$ denote the set of all possible situations under consideration. To characterize situations in $Q$, one determines a set $S$, called the descriptor set, which consists of variables, predicates and atomic functions (called, generally, descriptors), whose specific values can adequately characterize (for the problem at hand) any specific situation. We will assume here that the arguments of atomic functions are single variables, rather than other atomic functions. A situation is characterized by an event which is a sequence of assignments $L := v$, where $L$ is a variable or an atomic function with specific values of arguments, and $v$ is a value of the variable or atomic function which characterizes the situation. It is assumed that each descriptor has defined a value set (domain) which contains all possible values the descriptors can take for any situation in $Q$. Certain descriptors may not be applicable to some situations and therefore it is assumed that a descriptor in such cases takes value NA, which stands for not applicable. Thus, the domains of all descriptors always include by default the value NA. The set of all possible events for the given descriptor set $S$ is called the event space, and denoted $\mathcal{E}(S)$. The domain of quantified variables are assumed to be determined by a given situation or object. For example, if the quantified variable is a part, then its values are assumed to be individual parts of the object. In an event describing such an object, there will be a sequence of pairs $(L := V_1, i = 1, 2, \ldots)$, where $L$ is a quantified variable, and $V_i$ stands for different values this variable takes in the object.

An event $e \in \mathcal{E}(S)$ is said to satisfy a selector:

$$[f(x_i, \ldots, x_k) \neq R]$$

if the value of function $f$ for values of $x_i, i = 1, 2, \ldots, k$, as specified in the event $e$, is related to $R$ by =. For example, the event

$$e: (x_1 := a_1, x_2 := a_2, f_{50}(a_1, a_2) := 5, \ldots)$$

satisfies the selector

$$[f_{50}(x_1, x_2) = 1, 3, 5]$$

A satisfied selector is assigned truth-status TRUE. If an event does not satisfy a selector then the selector is assigned truth-status FALSE. If an event does not have enough information in order to establish whether a selector is satisfied or not then the selector has an unknown truth-status with regard to the event.
Let us assume first that the condition part of an inference rule is a quantifier-free formula. Interpreting the connectives $\forall$, $\exists$, $\top$, as described in Fig. 1, one can determine from the truth status of selectors the truth-status of the whole formula. An event is said to satisfy a rule, iff an application of the condition part of the rule to the event gives the formula truth-status $\top$. Otherwise, the event is said to not satisfy the rule.

Suppose now that the condition formula is in the form
$$\exists x (V(x)).$$

An application of this formula to an event assigns status $\top$ to the formula iff there exists in $e$ a value assigned to $x$ such that $V$ achieves status $\top$. For example, the formula
$$\exists p \text{ part}[\text{color}(\text{part}) = \text{red}]$$
is satisfied by the event
$$e = (\ldots \text{ part} = P1, \text{color}(P1) = \text{blue}, \text{part} = P2, \text{color}(P2) = \text{yellow}, \text{part} = P3, \text{color}(P3) = \text{red}) \ldots$$

If the condition part is a form
$$\forall x (V(x)),$$
then it is assigned status $\top$ if every value of $x$ in the event applied to it satisfies $V$.

If the condition part assumes truth-status $\top$, then the decision part is assigned status $\top$. When the decision part reaches status $\top$ then variables and functions which occur in it are assumed to have values which make this formula $\top$. These values may not, in general, be unique.

For example, suppose that $V$ is a decision part with status $\top$:
$$V: \{p(x_1, x_2) = 2\} \land [x_3 = 2 \ldots 5] \land [x_4 = 7].$$

$V$ is interpreted as a description of a situation in which $p$ has value 2 (if a specification of $p(x_1, x_2)$ is known, then from it we can infer that values of $x_1$ and $x_2$ might be), $x_3$ has a value between 2 and 5, inclusively, and $x_4$ has value 7. (Note that the formula does not give precise information about the value of $x_3$.) After applying a formula to an event, the truth status of the condition and decision part returns to $\top$.

The role of an inference rule can then be described as follows: the rule is applied to an event, and if the event satisfies the condition part, then an assignment of values to variables and functions is made as defined by the decision part. This assignment defines a new event (or a set of events which satisfy the decision part). Another inference rule can now be applied to this event (or set of events), and if satisfied by it (or by all of them), a new assignment of values to some variables and functions can be made.

Examples of $\forall E$ inference rules are:
$$[p(x_1, x_2) = 2] \land [q(x_2) = 2] \land [x_4 = \neq 0] \Rightarrow [d(y_1) = 7]$$
$$\lor [p(x_1, x_2) = 2]$$
$$\exists x_3 ([p(x_3, x_5) = 2 \ldots 3] \land [q(x_7, x_5) \geq 2]) \lor [r(x_1) = 1] \Rightarrow [d(y_1) = 7]$$
$$\land \top \Rightarrow [p(x_2, x_3) = 2] \land [x_2 = 2, 3, 5].$$

B. Specification of the Problem Environment in the Form of Inference Rules

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 here 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).

3) Tree-Ordered: Elements of the domain are ordered into a tree structure. A superior node in the tree structure represents a concept which is more general than the concepts represented by the subordinate nodes (e.g., the superior 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 (13), (14), (15)].

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

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

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

\[ \text{NA = not applicable.} \]

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 by

\[ T \Rightarrow \forall x_1, x_2 \ [f(x_1, x_2) \geq g(x_1, x_2)]. \]

3) Properties of Predicate Functions: For example, suppose that a predicate function is transitive. We represent this by

\[ T \Rightarrow \forall x_1, x_2, x_3 \ [\left[ \text{left}(x_1, x_3) \right] \ [\left[ \text{left}(x_2, x_3) \right] \]

\[ \Rightarrow \left[ \text{left}(x_1, x_3) \right]). \]

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

C. Generalization Rules

The transformation of data rules (3) into hypotheses (4) can be viewed as a process of applying certain generalization rules to data rules. A generalization rule transforms one or more decision rules associated with the same generalization class (which, in our case, is the same as recognition class) into a
new decision rule, which is equivalent to or more general than the initial rules.

A decision rule

$$V ::> K$$

(9)

is equivalent to a set of decision rules

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

(10)

if any event which satisfies at least one of the $$V_i, i = 1, 2, \ldots,$$

$$W[\text{color} \ (\text{wall}) = \text{blue}] ::> K,$$

$$W[\text{color} \ (\text{wall}) = \text{red}] ::> K,$$

satisfies also $$V,$$ and conversely. If the converse is not required, the rule (9) is said to be more general than (10).

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 (is a hypothesis); its truth-status has to be tested on new data.

Below is a list of a few basic generalization rules ($$K$$ denotes a generalization class).

**Nonconstructive Rules:**

i) **Dropping Condition Rule:** If a description is a logical product of conditions which must be satisfied, then one way to generalize it is to drop one or more of these conditions. For example,

$$[\text{size} \ (\text{box}) = \text{small}] \land [\text{color} \ (\text{box}) = \text{blue}] ::> K$$

$$\text{size} \ (\text{box}) = \text{small} ::> K.$$  

This reads: the description “small and blue box” can be generalized to “small box.” ($$\land$$ is the generalization operator.)

In general this rule can be expressed

$$W[L = R] ::> K \quad \text{or} \quad W ::> K,$$

where $$W$$ is an arbitrary description. This rule is generally applicable (the type of $$L$$ does not matter).

ii) **Turning Constants to Variables Rule:** When we have two or more descriptions, each referring to a single object in a class, and the descriptions differ in having different constants in the same predicate, then they can be generalized into one description with an existentially quantified variable in the place of the constants,

$$\begin{cases} W[p(a, Y)] ::> K \\ W[p(b, Y)] ::> K \\ \vdots \\ W[p(i, Y)] ::> K \end{cases}\quad \exists x \ W[p(x, Y)] ::> K,$$

one or more rules

where $$p$$ is a predicate and $$Y$$ stands for one or more arguments of the predicate $$p.$$ For example,

$$\begin{cases} W[\text{INSIDE} \ (\text{ball}, \text{box})] ::> K \\ W[\text{INSIDE} \ (\text{cup}, \text{box})] ::> K \end{cases} \quad \exists x \ W[\text{INSIDE} \ (x, \text{box})] ::> K.$$

The generalization (on the right of $$\exists$$) states that if an object is a box which has something inside, then it belongs to class $$K.$$ This rule together with the dropping condition rule are two basic generalization rules used in the literature on computer

induction. Both these rules can, however, be viewed as special cases of the following rule.

iii) **Generalization by Internal Disjunction (The Extending Reference Rule):** A description can be generalized by extending the set of values that a description (a variable, predicate, or a function) is allowed to take on in order than an object satisfies the description. This extension is expressed by the internal disjunction (Definition 2), i.e., logical or involving values of the same variable. For example,

$$W[\text{color} (\text{wall}) = \text{blue}, \text{red, green}, \ldots] ::> K.$$

(The “,” denotes internal disjunction.) In general, we have

$$W[L = R_1] ::> K \quad \text{or} \quad W[L = R_2] ::> K,$$

where $$L$$ is an atomic function and $$R_1, R_2$$ are references (i.e., subsets of values from the domain of $$L$$ expressed as internal disjunction) and $$R_1 \subseteq R_2.$$

Although the internal disjunction seems at first glance to be just a notational abbreviation, this operation is one of fundamental operations people use in generalizing descriptions. In addition to the previous two rules, there are two more important special cases of this rule. First, when the descriptor involved takes on values which are linearly ordered (a linear descriptor) and the second when the descriptor takes on values which are natural language concepts representing different levels of generality (a structured descriptor).

In the case of a linear descriptor we have the following.

iv) **Closing Interval Rule:** For example, suppose two objects of the same class have all the same characteristics except that they have different sizes, $$a$$ and $$b.$$ Then, it is plausible to hypothesize that all objects which share these characteristics but which have sizes between $$a$$ and $$b$$ are also in this class.

$$W[\text{size} (\text{x}) = a] ::> K \quad \text{or} \quad W[\text{size} (\text{x}) = b] ::> K.$$  

In general,

$$W[L = a] ::> K \quad \text{or} \quad W[L = b] ::> K.$$

This rule is applicable only when $$L$$ is a linear descriptor. In the case of structured descriptors we have the following.

v) **Climbing Generalization Tree Rule:** Suppose the value set of the shape descriptor is the tree (in general it could be a partially ordered set):

```
  polygon  plane geometric figure
  |          
  triangle  rectangle  pentagon  ellipse  circle.
  ```

With this tree structure, values such as triangle and rectangle can be generalized by "climbing the generalization tree" into a polygon.
[shape(x) = rectangle] ::= K
[shape(x) = triangle] ::= K
\{ [shape(x) = polygon] ::= K \}

A general rule is
\[
\begin{align*}
W[L = a] &::= K \\
W[L = b] &::= K \\
\vdots \\
W[L = i] &::= K
\end{align*}
\]

where L is a structured descriptor and s represents the superior node (a concept at the next level of generality) of 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 [3], [4], [26].

vi) Extension Against Rule: This rule applies when a description is being generalized in the presence of another description, representing the “negative examples” of the given recognition class. The latter description provides an obvious limit for the generalization of the given description, since these two descriptions should not intersect in order to avoid inconsistency. For example,
\[
\exists p_1, p_2 \,(\text{on-top}(p_1, p_2)) \text{color}(p_1) = \text{red} \] ::= K <
\exists p_1, p_2 \,(\text{on-top}(p_1, p_2)) \text{color}(p_1) = \text{green} \] ::= K
\]

The description produced by the rule: “there exists p whose color is not green” is the most general statement which satisfies both premises on the left of <.
In general, the rule is
\[
W_1[L = R_1] ::= K < [L \neq R_2] ::= K
\]
where \( R_1 \cap R_2 = \emptyset \) and \( W_1 \) and \( W_2 \) are arbitrary descriptions.
This rule is very useful in generating the discriminative descriptions of object classes (see next section). It is one of the basic rules used in the inductive program AQUAL/1 [14], whose version is used as a basic procedure in program INDECT-1 described in Section VI-B.

Constructive Rules

Constructive rules generate generalized descriptions of the data rules in terms of certain new descriptors (metadescrptions or derived descriptors). There can be very many such rules. We will restrict ourselves here to two examples. Some constructive rules are encoded as specialized procedures.

vii) The Counting Rule
\[
W[\text{attribute}_1(P_1) = A] \cdots [\text{attribute}_1(P_k) = A]
\]
\[\lambda \, [\text{attribute}_1(P_{k+1}) \neq A] \cdots [\text{attribute}_1(P_t) \neq A]
\]
\[::= K < W[\#P_{\text{attribute}_1} \geq k] ::= K
\]

where
\[
P_1, P_2, \ldots, P_k, \ldots, P_t
\]
are constants denoting, e.g., parts of an object
\( \text{attribute}_1 \)
stands for a certain attribute of \( P_1 \)-e.g., color, size, texture.-

\( \#P_{\text{attribute}_1} \) denotes a new descriptor interpreted as the “number of \( P_1 \)-s (e.g., parts) with \( \text{attribute}_1 \) equal A”.

Example:
\[
W[\text{color}(P_1) = \text{RED}] \text{color}(P_2) = \text{RED}
\]
\[\lambda \, [\text{color}(P_3) = \text{BLUE}] ::= K
\]
\[< W[\#P_{\text{color}_{\text{RED}}} \geq 2] ::= K
\]

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

viii) The Generating Chain Properties Rule: If the arguments of different occurrences of the same relation in an event are linearly ordered by the relation (e.g., are objects ordered linearly by a relation above, left-of, next-to, contains, etc.), that is to form a chain, the rule generates descriptors which characterize various 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 “most object,” i.e., the object at the end of the chain.

MDL-object: the “middle” object

Ith-object: the ith object in the chain

or characterize the chain itself, for example the chain-length.

D. The Preference Criterion

The preference criterion defines what is the desired solution to the problem, i.e., what kind of hypotheses are being sought. The question of what should be the preference criterion is a broad subject beyond the scope of the paper. We will, therefore, discuss here only the underlying ideas behind the presented approach. First, we disagree with many authors who seem to be seeking for one universal criterion which should guide induction. Our position is that there are many dimensions, independent and interdependent, on which a hypothesis can be evaluated. The weight given to each dimension depends on the ultimate use of the hypothesis. Among these dimensions are various forms of simplicity of a hypothesis (e.g., the number of operators in it), the quantity of information required to encode a hypothesis using operators from an a priori defined set [31], etc.), the scope of the hypothesis, which relates the events predicted by a hypothesis to the events actually observed (e.g., “degree of generalization” [14], the “precision” [31]), 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 permits a user to define various specific criteria to the inductive program, which are appropriate to the application. The form, called “lexicographic functional” consists of an ordered list of criteria (of dimensions of hypothesis quality) and a list of “tolerances” for these criteria [13], [14].

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. The characteristic description specifies the common properties shared by the objects of the same class (most work on induction considers only this type of descriptions, e.g., [3], [5], [22], [23]), while the discriminant description specifies only the properties necessary for distinguishing the given class from all the other classes (Michalski [13], [32] and Larson [28]).

E. Arithmetic Descriptors

In addition to initial linear descriptors used in the data rules, new linear descriptors can be formulated as arithmetic functions of the original ones. These descriptors are formulated by a human expert as suggestions to the program.

VI. OUTLINE OF ALGORITHM AND OF COMPUTER IMPLEMENTATION

In this section we outline the top level algorithm for rule induction and its implementation in the computer program INDUCE-1.1 [28], [29], [30]. The algorithm is illustrated by an example.

INDUCE-1.1 is considered to be only an aid to rule induction. Its successful application to practical problems requires a cooperation between the program and an expert, whose role is to determine the initial set of descriptors, to formulate data rules and the problem knowledge rules, to define the preference criterion and other parameters, evaluate the obtained rules, repeat the process if desired, etc.

A. Computer Representation of VI. Decision Rules

Decision rules are represented as graphs with labeled nodes and labeled directed arcs. A label on a node can be:

a) a selector with a descriptor without the argument list,
b) a logical operation,
c) a quantifier form $\exists x$ or $\forall x$.

Arcs link arguments with selectors or descriptors, and are labeled by 0, 1, 2, ..., to specify the position of an argument in the descriptor indicated at the head of the arc (0 indicates that the order of arguments is not important).

Several different types of relations may be represented by an arc. The type of relation is determined by the label on the node at each end of the arc. The types of relations are 1) functional dependence, 2) logical dependence, 3) implicit variable dependence, and 4) scope of variables.

Fig. 2 gives a graph representing a VI. expression. The two arcs connected to the logical operation ($\Lambda$) represent the logical dependence of the value of the formula on the values of the two selectors. The other arcs in the figure represent the functional dependence of $f$ on $x_1$ and $x_2$, and $g$ on $x_2$.

B. Outline of the Top Level Algorithm

The implementation of the inductive process in the program INDUCE-1.1 was based on ideas and algorithms adopted from the earlier research on the generalization of VI. expressions (Michalski [13], [32], and some new ideas and algorithms developed by Larson [28], [29].

The top level algorithm (in somewhat simplified form) can be described as follows:

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 one or more quantifier forms, i.e., forms $\text{QP}x_1, x_2, \ldots$, where QP 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 (used, e.g., in [23], [24]).

2) A decision class is selected, say $K_1$, and all c-expressions associated with this class are put into a set $F_1$, and all remaining c-expressions are put into a set $F_0$ (the set $F_1$ represents events to be covered, and set $F_0$ represents constraints, i.e., events not to be covered).

3) By application of inference rules (describing the problem environment), constructive generalization rules, and rules generating arithmetic descriptors (Section V-E) new selectors are generated. The “most promising” selectors (according to a chosen criterion) are added to the c-expressions in $F_1$ and $F_0$.

4) A c-expression is selected from $F_1$, 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 $F_0$. This “rule growing” process is illustrated in Fig. 3.

5) The obtained c-expressions, and c-expressions in $F_0$, are transformed to two sets $E_1$ and $E_0$, respectively, of VI. events (i.e., sequences of values of certain discrete variables).

A procedure for generalizing VIII. descriptions is then applied to obtain the “best cover” (according to a user defined criterion) of set $E_1$ against $E_0$ (the procedure is a version of AQUAL/1 program [13]).

During this process, the extension against, the closing the internal, 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 $F_1$, then the process repeats for another decision class. Otherwise, the set $F_1$ is reduced to contain only the uncovered c-expressions, and steps 4)-7) are repeated.

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

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 redundant c-expressions (includes application of AQLVAL/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 [30].

C. Example

We will present now an example illustrating some of the features of INDUCE-1.1. Suppose given are two sets of trains, Eastbound and Westbound, as shown in Fig. 4. 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).

As the first step, an initial set of descriptors is determined (by a user) for describing the trains. Eleven descriptors are selected in total. Among them are:

- infront(car₁,car₂)  car₁ is in front of car₂ (a nominal descriptor)
- length(car₁)  the length of car₁ (a linear descriptor)
- car-shape(car₁)  the shape of car₁ (a structured descriptor with 12 nodes in the generalization tree; \( \phi = \{1, 2, \ldots, 12\} \))

The data rules consist of descriptions of the individual trains in terms of the selected descriptors, together with the specification of the train set they belong to. For example, the data rule describing the second eastbound train is

\[
\exists \text{car}_1, \text{car}_2, \text{car}_3, \text{car}_4, \text{load}_1, \text{load}_2, \ldots
\]
\[
[\text{infront(car}_1, \text{car}_2) \text{]} \quad [\text{infront(car}_2, \text{car}_3) \text{]} \quad \ldots
\]
\[
[\text{length(car}_1) = \text{long}] \quad [\text{car-shape(car}_1) = \text{engine}] \quad
\]
\[
[\text{car-shape(car}_2) = \text{U-shaped}] \quad [\text{load-shape(load}_1) = \text{triangle}] \quad [\text{nrwheels(car}_3) = 2] \ldots
\]
\[
::: \quad [\text{class} = \text{Eastbound}] \quad (12)
\]

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

\[
\text{consists of a large collection of specialized algorithms, each accomplishing a certain task. Among the most important tasks are the following.}
\]
\[
\text{1) The implementation of the “rule growing process.”}
\]
\[
\text{2) Testing whether one c-expression is a generalization of (“covers”) another c-expression. This is done by testing for subgraph isomorphism.}
\]
\[
\text{3) Generalization of a c-expression by extending the selector references and forming redundant c-expressions (includes application of AQLVAL/1 procedure).}
\]
\[
\text{4) Generation of new descriptors and new selectors.}
\]
\[
\text{Program INDUCE 1.1 has been implemented in PASCAL (for Cyber 175 and DEC 10); its complete description is given in [30].}
\]
\[
\text{C. Example}
\]
\[
\text{We will present now an example illustrating some of the features of INDUCE-1.1. Suppose given are two sets of trains, Eastbound and Westbound, as shown in Fig. 4. The problem is to determine a concise (logically sufficient) description of each set of trains, which distinguishes one set from the other (i.e., a discriminant description which contains only necessary conditions for distinguishing between the two sets).}
\]
\[
\text{As the first step, an initial set of descriptors is determined (by a user) for describing the trains. Eleven descriptors are selected in total. Among them are:}
\]
\[
\text{- infront(car}_1, \text{car}_2)  \quad \text{car}_1 \text{ in front of car}_2  \quad \text{a nominal descriptor)}
\]
\[
\text{- length(car}_1)  \quad \text{the length of car}_1  \quad \text{a linear descriptor)}
\]
\[
\text{- car-shape(car}_1)  \quad \text{the shape of car}_1  \quad \text{a structured descriptor with 12 nodes in the generalization tree; \( \phi = \{1, 2, \ldots, 12\} \)}
\]
\[
\text{The data rules consist of descriptions of the individual trains in terms of the selected descriptors, together with the specification of the train set they belong to. For example, the data rule describing the second eastbound train is}
\]
\[
\exists \text{car}_1, \text{car}_2, \text{car}_3, \text{car}_4, \text{load}_1, \text{load}_2, \ldots
\]
\[
[\text{infront(car}_1, \text{car}_2) \text{]} \quad [\text{infront(car}_2, \text{car}_3) \text{]} \quad \ldots
\]
\[
[\text{length(car}_1) = \text{long}] \quad [\text{car-shape(car}_1) = \text{engine}] \quad
\]
\[
[\text{car-shape(car}_2) = \text{U-shaped}] \quad [\text{load-shape(load}_1) = \text{triangle}] \quad [\text{nrwheels(car}_3) = 2] \ldots
\]
\[
::: \quad [\text{class} = \text{Eastbound}] \quad (12)
\]
\[
\text{Rules describing the problem environment in this case are only rules defining structures of structured descriptors (arguments of descriptors are omitted):}
\]
\[
\text{At this moment, before proceeding further, the reader is advised to look at the figures and try to figure this problem on his/her own.}
\]
[car-shape=open rectngl,open trapezoid,U-shaped],
  ⇒ [car_shape = open top]
(13)
[car-shape=ellipse,closed rectngl,jagged top,sloping top],
  ⇒ [car_shape = closed top]
(14)
[load_shape = hexagon, triangle, rectangle],
  ⇒ [load_shape = polygon].
(15)

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

Rules of constructive generalization included in the program are the ability to construct, among other descriptors, such descriptors as the length of a chain, properties of elements of a chain, number of objects satisfying a certain relation, etc. For example, from the data rule (12), the constructive generalization rules can produce new selectors such as

[nrcars=4] the number of cars in the train is 4 (the length of chain defined by relation infrinct)

[nrcars-length-long=1] the number of long cars is 1 (the engine)

[nr-pts-load(last-car)=2] the number of parts in the load of the last car is 2

[position(car)=i] the position of car_i is i.

Suppose that eastbound trains are considered first. The set F1 contains then all c-expressions describing eastbound trains, and F0; all c-expressions describing westbound trains. The description e is selected from F1 (suppose it is the above description of the second eastbound train), and supplemented by “most promising” metadescrcriptors generated by problem environment rules and constructive generalization rules. In this case, the metascriptor [shape(last-car) = rectangle] is added to e. Next, a set G (a restricted star) of certain number (nconsistent) of consistent generalizations of e is determined.

This is done by forming a sequence of partial stars (a partial star may include inconsistent generalizations of e). If an element of a partial star is consistent, it is placed into the set G. The initial partial star (P₀) contains the set of all selectors of e. This partial star and each subsequent partial star is reduced according to a user specified preference criterion to the “best” subset, before a new partial star is formed. The size of the subset is controlled by a parameter called MAXSTAR. A new partial star P_i+1 is formed from an existing partial star P_i in the following way: for each c-expression in P_i, a set of c-expressions is placed into P_i+1, each new c-expression containing the selectors of the original c-expression plus one new selector from e, which is not in the original c-expression. Once a sufficient number of consistent generalizations have been formed, a version of the AQVAL/1 program (Michalski [14]) is applied to extend the references of all selectors in each consistent generalization. As the result, some selectors may be removed and some may have more general references.

In the example, the best subset of selectors of e (i.e., the reduced partial star (P₁) was

∃ car₁ [car-shape(car₁) = U-shaped]
(16)
∃ car₁ [car-shape(car₁) = open-trapezoid]
(17)
∃ car₁ [car-shape(car₁) = rectangle]
(18)
[car-shape(last-car) = rectangle].
(19)

The last c-expression is consistent (has empty intersection with c-expressions in F0) and, therefore, is placed in G. From the remaining, a new partial star is determined. This new partial star contains a consistent generalization:

∃ car₁ [car-shape(car₁) = rectangle] [length(car₁) = short]
(20)

which is added to G. Suppose G is restricted to have only two elements (nconsistent = 2). Now, the program AQVAL/1 is applied to generalize references of the selectors in c-expressions of G, if it leads to an improvement (according to the preference criterion).

In this case, a generalization of (20) produces a consistent and complete generalization:

∃ car₁ [car-shape(car₁) = closed top] [length(car₁) = short]
(21)

(The generalization of (19), [car-shape(last-car) = polygon], is not complete; it does not cover all F1.)

In this example, only two partial stars were formed, and two consistent generalizations were created. In general, a set of consistent generalizations is created through the formation of several partial stars. The size of each partial star and the number of alternative generalizations are controlled by user supplied parameters.

Assuming a larger value of nconsistent, and applying the above procedure to both decision classes, the program INDUCE-1.1 produced the following alternative descriptions of each set of trains.

(The selectors or references underlined by a dotted line were generated by application of constructive generalization rules or problem environment rules.)

Eastbound Trains:

∃ car₁ [length(car₁) = short] [car-shape(car₁) = closed top] 
  ::= [class = Eastbound]
(22)

[the same as (21)]. It can be interpreted as follows.
If a train contains a car which is short and has a closed top, then it is an eastbound train. Alternatively,

∃ car₁, car₂, load₁, load₂ [infront(car₁, car₂)]
  [cont-load(car₁, load₁)]
  [cont-load(car₂, load₂)]
  [load_shape(load₁) = triangle]
  [load_shape(load₂) = polygon] ::= [class = Eastbound]
(23)

It can be interpreted as follows.
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.
Westbound Trains:
\[ \text{[nCars=3]} \lor \exists \text{car} \ [\text{car_shape(car)}=\text{jagged-top}] \]
\[ \implies [\text{class}=\text{Westbound}] \]  \hspace{1cm} (24) \\
\[ \exists \text{car} \ [\text{nCars_length} \land \text{long} = 2] \land [\text{position(car)}=3] \]
\[ \implies [\text{class}=\text{Westbound}] . \]  \hspace{1cm} (25)

It is interesting to note that the example was constructed with rules (23) and (24) in mind. The rule (22) found by the program as an alternative was rather surprising because it seems to be conceptually simpler than rule (23).

This example shows that the combinatorial part of an induction process can be successfully handled by a computer program. Therefore, programs like the above have a potential to serve as an aid to induction processes in various applied sciences, pattern unknown to the authors.

ACKNOWLEDGMENT

The author acknowledges the collaboration with J. Larson of Rockwell International, Inc., in developing several ideas presented here and, in particular, his outstanding implementation of the first version of the program, INDUCE-I. Among many people who helped through discussions or through their interest in the work, the author would specially like to mention K. S. Fu, B. Gaines, D. Michie, R. Reddy, L. Travis, and L. Uhr. Thanks go also to A. B. Baskin and T. Dietterich for proof-reading of the paper.

REFERENCES


Ryszard S. Michalski received the B.Sc., M.Sc., and Ph.D. degrees from Warsaw Technical University, Leningrad Polytechnical Institute, and the Technical University of Silesia, respectively. From 1962-1970 he worked first as a Research Scientist, and then as the Leader of the Pattern Recognition Group at the Institute of Automatic Control of the Polish Academy of Sciences, Warsaw. In 1970 he joined the Department of Computer Science, University of Illinois, Urbana, where he is an Associate Professor. He is the author of more than 40 research and technical papers published in the U.S. and abroad. Currently he is the principal investigator of an NSF-funded project on computer induction and plausible reasoning, and a co-principal investigator of a project on the application of computer inference to agriculture, founded by the U.S. Department of Agriculture.