

PATTERN RECOGNITION AS  
KNOWLEDGE-GUIDED  
COMPUTER INDUCTION

by

*Ryszard S. Michalski*

Report No. 927, Department of Computer Science, University of Illinois,  
Urbana, Illinois, June 1978.

PATTERN RECOGNITION  
AS  
KNOWLEDGE-GUIDED COMPUTER INDUCTION

Ryszard S. Michalski  
University of Illinois  
at Urbana-Champaign

*Abstract:* The determination of pattern recognition rules is viewed as a problem of computer induction, under the guidance of *generalization rules* and rules representing knowledge of the recognition problem at hand. The paper formulates the underlying theory for generalization and optimization of descriptions of object classes, expressed in the form of decision rules. The language for formulating descriptions is an extension of the first order predicate calculus, called the variable-valued logic calculus VL<sub>21</sub>. VL<sub>21</sub> contains several new syntactic forms, specially oriented for expressing inductive processes. The presented approach uniformly combines descriptors (variables, predicates, functions) of three different types: nominal, linear and structured, and has an ability to generate new descriptors not used in the initial data rules.

Index Terms pattern recognition, decision rules, generalization techniques, computer induction, knowledge-acquisition, learning from examples, many-valued logic, computer inference, computer consulting systems, theory formation, inductive inference

This work was supported in part by the National Science Foundation under Grant NSF MCS 76-22940. The author is with the Department of Computer Science, University of Illinois, Urbana, Illinois 61801



## 1. INTRODUCTION

A pattern recognition rule can be viewed as a rule

DESCRIPTION ::> RECOGNITION CLASS (1)

which assigns a *situation* (an object, a process, etc.) to the RECOGNITION CLASS, when the situation satisfies the DESCRIPTION. In the decision space approach the DESCRIPTION is in the form of 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 the relevant object characteristics are various relations among the variables, or among parts or subparts of objects, then the decision space approach becomes inadequate. In such situations the structural approach can be useful.

In the structural approach, the DESCRIPTION is a formal grammar (usually a phrase-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, of object parts, logical conditions on properties, etc.; i.e., involve *descriptors* of mixed arity and measured on different scales. Thus, there is a need for a method which could handle simultaneously all such descriptors.

Both the decision space approach and the syntactic approach tend to produce descriptions which are not easily comprehensible by humans. This is so because these descriptions do not directly correspond to the 'natural language type' descriptions which human experts would develop observing the same data and which they would normally like to use. Although in some applications 'human comprehensibility' may not be important, in other applications (e.g., in expert computer consulting systems) it is a crucial requirement.

This paper presents results, still early and limited, of an attempt to develop a uniform conceptual framework and an implementation method which would satisfy both of the above requirements. In addition, an important 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 'metarules' which represent the underlying knowledge of the problem at hand and of the properties of descriptors used in formulating the descriptions of exemplary data. Therefore, the approach taken is in the spirit of research in artificial intelligence. The method uses logic as the basic formal framework (specifically, a certain syntactic 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, in addition to those given a priori, places this work in the category of what we call 'constructive induction'\* as opposed to 'non-constructive induction', in which the final descriptions relate only descriptors initially provided.

---

\* The author thanks Larry Travis of the University of Wisconsin for suggesting this name.

## 2. RELATED RESEARCH

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

Many results consider inductive tasks within a specific problem domain. For example, programs collectively called METADENDRAL [1] use a model-directed heuristic search to determine rules that describe the molecular structure of an unknown chemical compound from mass spectrometry data. In [2] Winston describes a method for determining a graph description of simple block structures from examples. A program developed by Lenat [3] 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 [4] describe a method for creating multi-level 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 represent an important achievement and demonstrate again that high performance requires specialized solutions. An important problem which they raise, however, is how to untangle and systematize the ideas which they contribute, in order to extend understanding of inductive processes at large, and to apply in other problem areas.

A significant part of research has been concerned with determining patterns in sequences of symbols (e.g., Simon [5], Waterman [6]). Simon [5] found that descriptions of such patterns consistently incorporate only a few basic relations: 'some' and 'next' between symbols, iterations between subpatterns, and hierarchic phrase structure. Gaines [7] developed a method for generating finite-state automata, which approximate a given symbol string, and represent different trade-offs between the complexity and poor-ness-of-fit. Shaw, Swartout and Green [8] developed a program for inferring Lisp code from a set of examples of Lisp statements.

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 this area was concerned with the inference of a phrase structure grammar (e.g., Feldman et al [9]). More recent work moves into inferring 'multi-dimensional' grammars (e.g., work by Brayer and Fu [10]).

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

Michalski and his collaborators (e.g., [11, 12, 13]) have developed a methodology (using a sentential calculus with discrete variables, called variable-value logic system  $VL_1$ , as a formal basis) and computer programs for determining generalized and optimal in some sense discriminant descriptions of classes of objects from examples. The examples are presented as sequences of values of discrete variables with an associated recognition class. Work in a similar spirit, although more limited in scope, was reported by Stoffel [14] (the elementary statements used there are restricted to the 'variable-value' forms, i.e., to 'elementary selectors', as described in Section 4).

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 formulating hypotheses. Morgan [15] describes a formal method of hypothesis generation, called f-resolution, which stems from deductive resolution principles. Various theoretical issues of induction in FOPC were considered by Plotkin [16]. Fike, Hart and Nilsson [17] describe an algorithm for generalizing robot plans. Hayes-Roth and McDermott (e.g., [18]), also Vere [19], 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 'turning constants into variables' (see section 5.3). Related, but different in spirit, is work by Zagoruiko [20] on a general method for 'strengthening hypotheses' by narrowing the uncertainty intervals of values of output variables, and work by Hedrick [21] on determining production systems using a semantic net of predefined concepts.

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<sub>21</sub>. This formalism is claimed to be 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 [22], and stresses the conceptual principles



of induction method rather than specific algorithms and implementation details. Most of the latter are described in [23, 24, 25].

### 3. 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  $\text{VL}_{21}$  system (section 4),

$\Rightarrow$  stands for various transformation operators which define the meaning of the rule.

A DESCRIPTION may look like:

$\exists x_1, x_2 [\text{on-top}(x_1, x_2)] [\text{size}(x_1) = 3..5] [\text{color}(x_2) = \text{blue, yellow, red}] \wedge$   
[length( $x_1$ ) . length( $x_2$ ) = small]

(For explanation of notation see section 4).

We will consider here the following transformation operators:

(i)  $::>$  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)  $\Rightarrow$  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 the 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*.)

A decision rule can be viewed as a special case of an inference rule, namely, when DESCRIPTION<sub>2</sub> is a constant, an elementary selector, or a product of elementary selectors involving decision variables (see def. 2), also, when its truth-status is TRUE (in general, it may be not TRUE).

- (iii)  $\vDash$  the operator defines a *generalization rule*, which states that the DESCRIPTION<sub>2</sub> is *more general* than DESCRIPTION<sub>1</sub>, i.e., the set of situations which satisfy DESCRIPTION<sub>2</sub> is a superset of the set of situations satisfying DESCRIPTION<sub>1</sub>.
- (iv)  $\vDash$  the operator specifies an *equivalence preserving transformation rule* (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:

• Given is

- (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,  $K_i$ , associated with them:

$$\begin{array}{ccccccc} C_{11} ::> K_1, & C_{12} ::> K_1 & \dots & C_{1t1} ::> K_1 \\ C_{21} ::> K_2, & C_{22} ::> K_2 & \dots & C_{2t2} ::> K_2 \\ & \cdot & & \\ & \cdot & & \\ C_{m1} ::> K_m, & C_{m2} ::> K_m & \dots & C_{mtm} ::> K_m \end{array} \quad (3)$$

- (b) a set of VL inference rules which define a *problem environment*, i.e., represent knowledge about the recognition problem under consideration. This includes value sets of descriptors used in the data rules, the properties of descriptors and their interrelationships characteristic to the problem at hand.
- (c) a *preference (or optimality) 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 ( sec. 5.3), a new set of decision rules (called *output rules* or *hypotheses*):

$$\begin{aligned} C'_{11} &::> K_1, & C'_{11} &::> K_1, & \dots & C'_{1r1} &::> K_1 \\ C'_{21} &::> K_2, & C'_{21} &::> K_2, & \dots & C'_{2r2} &::> K_2 \\ & \vdots & & & & & \\ C'_{m1} &::> K_m, & C'_{m2} &::> K_m, & \dots & C'_{mr^m} &::> K_m \end{aligned} \tag{4}$$

which are most preferable among all sets of rules that do not contradict the *problem environment* rules, and 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 decision, the output rules assign to it the same decision, or the NULL decision.

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

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 decision to situations to which the input rules assign a NULL decision).

From a given set of data rules it is usually possible to derive many different sets of rules which are consistent and complete and which satisfy the problem environment rules. The role of the preference criterion is to select one (or a few alternative sets of rules) which is most desirable in the given application. The preference criterion may refer to the simplicity of the rules (defined in some way), their generality, the cost of measuring the information needed for rule evaluation, degree of approximation to the given facts, etc. (section 5.4). In this paper we accept the restriction that the DESCRIPTIONS,  $C_{ij}$  and  $C'_{ij}$  are disjunctive simple VL<sub>21</sub> expressions (section 4). Such expressions have a very simple interpretation, and seem to be sufficient for many applications.

#### 4. VL EXPRESSIONS AS DESCRIPTIONS

##### 4.1 Definition of VL<sub>21</sub>

Data rules, hypotheses, problem environment descriptions, and generalization rules are all expressed using the same formalism, that of variable-valued logic calculus VL<sub>21</sub>.<sup>\*</sup> VL<sub>21</sub> 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. To make the paper self-contained, we will provide here a brief description of VL<sub>21</sub>.

There are three major differences between VL<sub>21</sub> and the first order predicate calculus:

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

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

---

<sup>\*</sup>VL<sub>21</sub> is a subset of a more complete system VL<sub>2</sub> under development.

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.

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

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

*Examples*

Constants            2 \* red

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

Exemplary

value sets:  $D(x_1) = \{0, 1, \dots, 10\}$

$D(\text{color}) = \{\text{red}, \text{blue}, \dots\}$

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

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

Definition 2: A *selector* is a form

[L # R]

(5)

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:

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

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

A selector in which the referee L is a simple atomic function and the reference R is a single constant is called an *elementary selector*. The selector has truth-status TRUE {or FALSE} with regard to a situation if the situation *satisfies* {*does not satisfy*} the selector, i.e., if the referee L is {is not} related by # to the reference R. The selector has the truth-status '?' (and is interpreted as being a *question*), if there is not sufficient information about the values of descriptors in L for the given situation. To simplify the exposition, instead of giving a definition of what it means that 'L is related by # to R', we will simply explain this by examples. (See section 5.1 for more details).

- (i) [color(box1) = white]                      color of box1 is white
- (ii) [length(box1) ≥ 2]                      length of box1 is greater than or equal to
- (iii) [weight(box1) = 2..5]                      weight of box1 is between 2 and 5,
- (iv) [blood-type (P1) = 0,A,B].                      blood-type of P1 is 0 or A or B
- (v) [on-top(box1, box2) = T]                      box1 is on top of box2  
or simply  
[on-top(box1, box2)]
- (vi) [above(box1, box2) = 3"]                      box 1 is 3" above box2
- (viii) [weight(box1) > weight (box3)]                      the weight of box1 is greater than the  
weight of box3
- (ix) [length(box1) . length (box2) = 3]                      the length of box1 and box2 is 3
- (x) [type(p<sub>1</sub>) . type (P<sub>2</sub>) = A,B]                      the type of P<sub>1</sub> and the type of P<sub>2</sub>  
is either A or B.

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

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

- (i) A constant TRUE, FALSE or '?' is a VL expression
- (ii) A selector is a VL expression
- (iii) If  $V$ ,  $V_1$  and  $V_2$  are VL expressions then so are:

(V) formula in parentheses

$\neg V$  inverse

$V_1 \wedge V_2$  or  $V_1 V_2$  conjunction

$V_1 \vee V_2$  disjunction

$V_1 \underline{\vee} V_2$  exclusive disjunction

$V_1 \searrow V_2$  exception

$V_1 \Rightarrow V_2$  metaimplication

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

$\exists x_1, x_2, \dots, x_k(V)$  existentially quantified expression

$\forall x_1, x_2, \dots, x_k(V)$  universally quantified expression

A VL formula can have truth-status TRUE (T), FALSE (F) or UNKNOWN(?).

The interpretation given to connectives  $\neg, \wedge, \vee, \rightarrow$ , is defined in Fig. 1. (This interpretation is consistent with Kleen-Körner 3-valued logic). An expression with the operator  $\Rightarrow, \vDash$  or  $\vDash$  is assumed to always have the truth-status TRUE and with operator  $::>$ , TRUE or ?. Operators  $\searrow, \underline{\vee}$ , and  $\leftrightarrow$  are interpreted:

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

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

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

The truth-status of

$\exists x(V)$  is  $\left\{ \begin{array}{ll} \text{TRUE \{FALSE\}} & \text{if, in a given situation, there exists} \\ & \text{\{does not exist\} a value of x which makes} \\ & \text{the truth-status of V equal TRUE} \\ ? & \text{if it is not known whether there exists . . . .} \end{array} \right.$



	$\neg$
?	?
F	T
T	F

$\wedge$	?	F	T
?	?	F	?
F	F	F	F
T	?	F	T

$\vee$	?	F	T
?	?	?	T
F	?	F	T
T	T	T	T

$\rightarrow$	?	F	T
?	?	?	T
F	T	T	T
T	?	F	T

DEFINITION OF CONNECTIVES  
 $\neg, \wedge, \vee$  AND  $\rightarrow$   
 IN VL<sub>21</sub>

Figure 1

$\forall x(V)$  is  $\begin{cases} \text{TRUE \{FALSE\}} & \text{if for every value of } x \text{ in a given situation,} \\ & \text{the truth-status of } V \text{ is \{is not\} TRUE} \\ ? & \text{if it is not known whether for every . . .} \end{cases}$

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

A VL expression in the form

$$QF_1, QF_2, \dots (P_1 \vee P_2 \vee \dots \vee P_n) \quad (7)$$

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

## 5. INFERENCE AND GENERALIZATION RULES

### 5.1 Interpretation of Inference Rules

An inference rule

$$\text{DESCRIPTION}_1 \Rightarrow \text{DESCRIPTION}_2 \quad (8)$$

is used by applying it to *situations*. A *situation* is, in general, a source of information about values of variables and atomic functions in  $\text{DESCRIPTION}_1$  (the *condition part* of the rule). A situation can, e.g., be a data base 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  $\text{DESCRIPTION}_2$  (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  $\text{DESCRIPTION}_2$  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  $\mathcal{E}(S)$ . It should be noted that within a single event certain variables (variables which are quantified in formulas) may be assigned a number of different values, i.e., there may be more than one pair  $(L:=v_i)$ , where  $L$  is a variable and  $v_i$ ,  $i = 1, 2, \dots$  represent different values.

An event  $e \in \mathcal{E}(S)$  is said to *satisfy* a selector  $[f(x_1, \dots, x_k) \# R]$  iff the value of function  $f$  for values of  $x_i$ ,  $i = 1, 2, \dots, k$ , as specified in the event  $e$ , is related to  $R$  by  $\#$ . For example, the event

$$e: (\dots x_5 := a_1, x_6 := a_2, f_{20}(a_1, a_2) := 5, \dots)$$

satisfies the selector:

$$[f_{20}(x_5, x_6) = 1, 3, 5]$$

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 UNKNOWN truth-status with regard to this event.

Let us assume first that the condition part of an inference rule is a quantifier-free formula. Interpreting the connectives  $\neg$ ,  $\wedge$ ,  $\vee$ , as described in figure 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 TRUE. Otherwise, the event is said to *not satisfy* the rule.

Suppose now that the condition formula is in the form

$$\exists x(V)$$

An application of this formula to an event assigns status TRUE to the formula iff there exists in  $e$  a value assigned to  $x$  such that  $V$  achieves status TRUE ( $x$  may have a number of different values assigned to it). For example, the formula

$$\exists \text{part} [\text{color}(\text{part}) = \text{red}]$$

is satisfied by the event:

$$e = (\dots \text{part}:=P1, \text{color}(P1):=\text{blue}, \text{part}:=P2, \text{color}(P2):=\text{yellow}, \\ \text{part}:=P3, \text{color}(P3):=\text{red}\dots)$$

If the condition part is a form

$$\forall x(V)$$

then it is assigned status TRUE if every value of  $x$  in the event applied to it satisfies  $V$ .

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

For example, suppose that  $V$  is a decision part with status TRUE:

$$V: [p(x_1, x_2) = 2][x_3 = 2:5][x_5 = 7]$$

$V$  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 what values of  $x_1$  and  $x_2$  might be),  $x_3$  has a value between 2 and 5, inclusively, and  $x_5$  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 UNKNOWN. 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 now can 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 VL inference rules:

$$\begin{aligned} & [p(x_1, x_2) = 3][q(x_2) = 2, 5][x_7 \neq 0] \Rightarrow [d(y_1) = 7][p(y_1, y_2) = 2] \\ \exists x_3 & ([p(x_1, x_3) = 2..3][q(x_7, x_3) \geq 2]) \vee [t(x_1) = 1] \Rightarrow [d(y_1) = 7] \\ \text{TRUE} & \Rightarrow [p(x_2, x_7) = 2][x_7 = 2, 3, 5] \end{aligned}$$

## 5.2 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 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. (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:

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

NA = not applicable

2. Relationships Between Atomic Functions

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

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

3. Properties of Predicate Functions

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

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

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

### 5.3. Generalization rules

In order to transform data rules (3) into hypotheses (4), generalization rules are applied 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

$$V ::=> K \quad (9)$$

is *equivalent* to a set of decision rules

$$\{V_i ::=> K\}, i = 1, 2, \dots \quad (10)$$

if any event which satisfies at least one of the  $V_i$ ,  $i = 1, 2, \dots$ , 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 may have 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).

#### Non-constructive rules:

(i) the *extending reference* rule

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

where  $L$  - is an atomic function

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

$V$  - an arbitrary description (here a VL expression).

This is a generally applicable rule; the type of descriptor

$L$  does not matter.

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

$$V[L = R] ::> K \quad \leftarrow \quad V ::> K$$

This rule is also generally applicable. It is one of the most commonly used rules for generalizing information.

It can be derived from rule (i), by assuming that  $R_2$  in (i) is equal the value set  $D(L)$ . In this case the selector  $[L = R_2]$  has always truth-status TRUE, and as such can be removed.

- (iii) The *closing interval* rule

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

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

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

- (iv) The *climbing generalization tree* rule

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



where L is a structured descriptor

s - represents the predecessor node (a concept at the next 'level of generality') of nodes a,b,...and i, the tree domain of L.

The rule is applicable only to selectors involving structured descriptors. This rule has been used, e.g., in [2], [3], [21].

Example:

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

(v) The *extension against rule*

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

where  $R_1 \cap R_2 = \emptyset$

$V_1$  and  $V_2$  - arbitrary descriptions.

This rule is of general applicability. 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

$$\left. \begin{array}{l} V[p(a,Y)] \text{ ::> } K \\ V[p(b,Y)] \text{ ::> } K \\ \vdots \\ V[p(i,Y)] \text{ ::> } K \end{array} \right\} V[p(x,Y)] \text{ ::> } K$$

one or more rules

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

\* is a variable whose value set includes a,b,...,i.  
 This is a rule of general applicability. It is the basic rule used in works on induction employing predicate calculus.

Constructive Rules:

Constructive rules generate descriptions of the data rules in terms of certain new descriptors, and, therefore, are a form of generalization rules. They also can be viewed simply as rules which generate new descriptors ('metadescriptors'). There can be very many such rules. We will restrict ourselves here to two examples. Some constructive rules are encoded as specialized procedures.

(vi) the *counting* rule

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

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

- where  $P_1, P_2, \dots, P_k, \dots, P_r$  - are constants denoting, e.g., parts of an object
- $\text{attribute}_1$  - stands for a certain attribute of  $P_i$ -s, e.g., color, size, texture, etc.
- $\#P\text{-attribute}_1\text{-}A$  - denotes a new descriptor interpreted as the 'number of  $P_i$ -s (e.g., parts) with *attribute* equal  $A$ '.

Example:

$$V[\text{color}(P1)=\text{RED}][\text{color}(P2)=\text{RED}][\text{color}(P3)=\text{BLUE}]:: > K$$

$$| < \quad \#P\text{-color-red}=2 \quad :: > K$$

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

(viii) the *generating chain properties* rule

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

LST-object - the 'least object', i.e., the object at the beginning of the chain (e.g., the bottom object in the case of relation 'above')

MST-object - the object at the end of the chain (e.g., the top object)

ith-object - the ith object of the chain.

5.4 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 searching for one universal criterion which should guide induction. Our position is that 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. Among these dimensions are various forms of simplicity of the hypothesis (e.g., the number of operators in it, the quantity of information required to encode the hypothesis using operators from an a priori defined set [26], etc.), the scope of the hypothesis, which relates the events predicted by the hypothesis to the events actually

observed (e.g., the 'degree of generalization' [12], the 'precision' [26]), 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 a 'lexicographic functional' consists of an ordered list of criteria (of dimensions of hypothesis quality) and a list of 'tolerances' for these criteria [12, 23].

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., [2], [5], [18]), while the *discriminant description* specifies only the properties necessary for distinguishing the given class from all the other classes (Michalski [12, 27], Larson [23]).

#### 5.5 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.

### 6. 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 ([23][24][25]). 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 formulate data rules and the problem environment rules, define the preference criterion and other parameters, evaluate the obtained rules, repeat the process if desired, etc.

### 6.1 Computer representation of VL decision rules

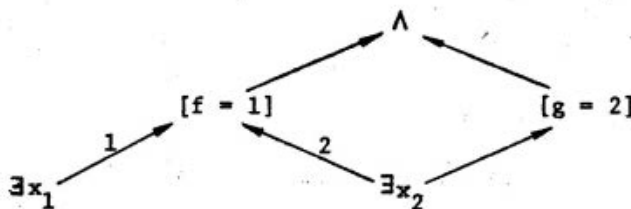
Decision rules are represented as graphs with labeled nodes and labeled directed arcs. A label on a node can be:

- 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, 4) scope of variables.

Figure 2 gives a graph representing a  $VL_{21}$  expression. The two arcs connected to the logical operation ( $\wedge$ ) 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  $x_2$ .



VL Graph Structure:  $\exists x_1 x_2 ([f(x_1, x_2) = 1][g(x_2) = 2])$

Figure 2

## 6.2. Outline of the Top Level Algorithm

The implementation of the inductive process in the program INDUCE-1 was based on ideas and algorithms adopted from the earlier research on the generalization of  $VL_1$  expressions (Michalski [12,27], and some new ideas and algorithms developed by Larson [23,24].

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  $QFx_1, x_2, \dots$ , 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 (used, e.g., in [18][19]).
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 (sec.5.5), new selectors are generated. The 'most promising' selectors (according to a certain 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 F0. This 'rule growing' process is illustrated in fig. 3.

5. The obtained c-expressions, and c-expressions in F0, are transformed to two sets E1 and E0, respectively, of VL<sub>1</sub> events (i.e., sequences of values of certain discrete variables).

A procedure for generalizing VL<sub>1</sub> 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 [12]).

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

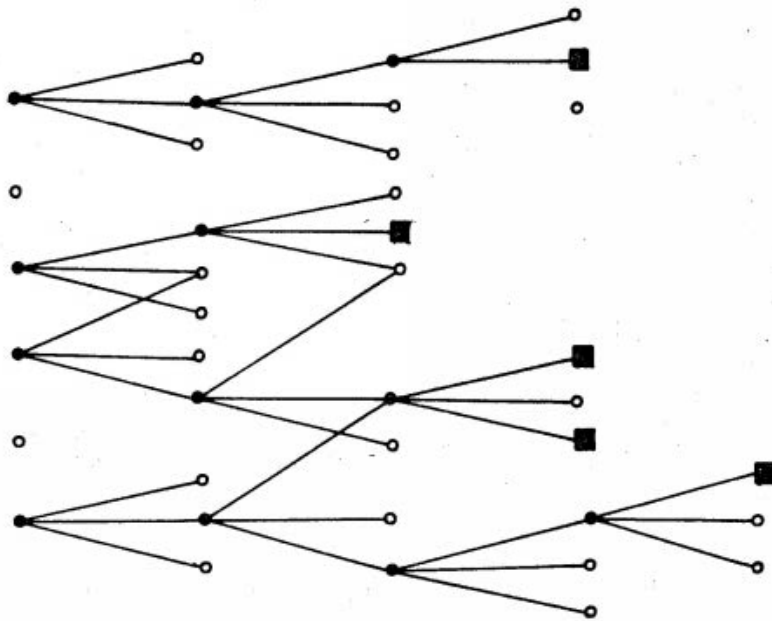
The result is transformed to a new set of c-expressions (a restricted star) in which selectors have now appropriately generalized references.

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

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

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

1. the implementation of the 'rule growing process'
2. testing whether one c-expression is a generalization of ('covers') another c-expression. This is done by testing for subgraph isomorphism.



○ - a discarded 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 3



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

4. Generation of new descriptors and new selectors.

Program INDUCE 1.1 has been implemented in PASCAL (for Cyber 175 and DEC 10); its complete description is given in [25].

### 6.3. 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).

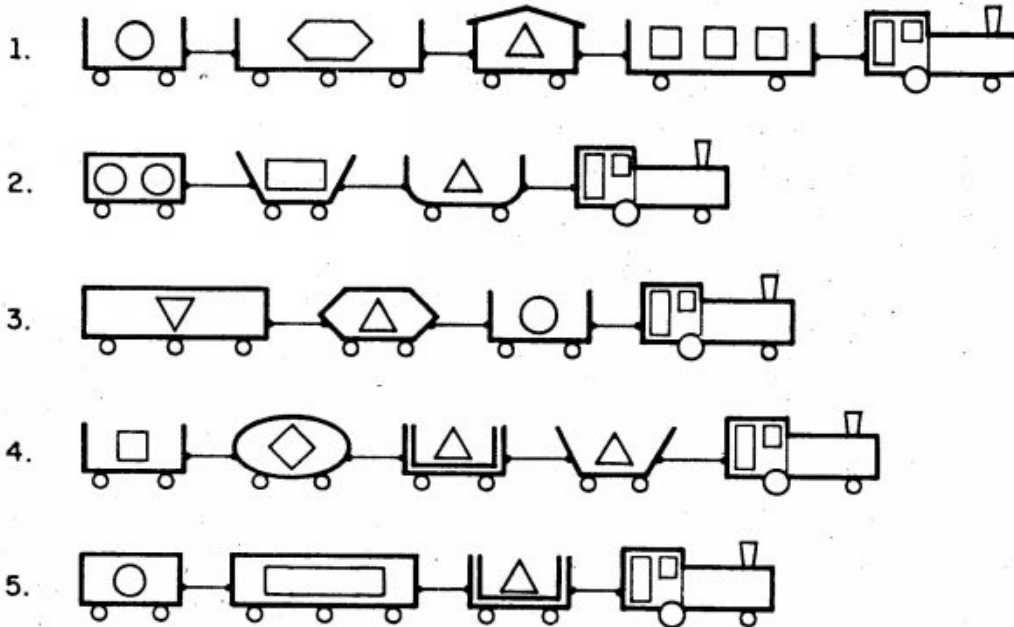
As the first step, an initial set of descriptors is determined for describing the trains. Eleven descriptors are selected in total.

Among them:

- $\text{infront}(car_i, car_j)$  -  $car_i$  is in front of  $car_j$   
(a nominal descriptor)
- $\text{length}(car_i)$  - the length of  $car_i$   
(a linear descriptor)
- $\text{car-shape}(car_i)$  - the shape of  $car_i$   
(a structured descriptor with 12 nodes in the generalization tree; see eqs. (13) and (14))
- $\text{cont-load}(car_i, load_j)$  -  $car_i$  contains  $load_j$   
(a nominal descriptor)
- $\text{load-shape}(load_i)$  - the shape of  $load_i$   
(a structured descriptor)  
The value set:

- circle
  - hexagon
  - triangle
  - rectangle
- polygon

1. EASTBOUND TRAINS



2. WESTBOUND TRAINS

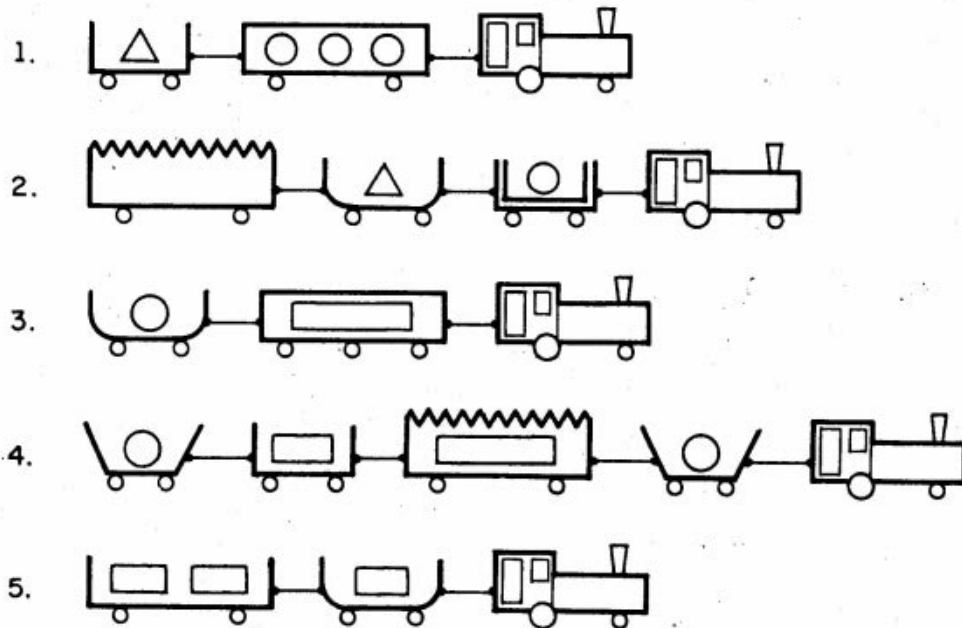


Figure 4

- $nrpts-load(car_i)$  - the number of parts in the load of  $car_i$   
(a linear descriptor)
- $nrwheels(car_i)$  - number of wheels in  $car_i^*$   
(a linear descriptor)

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:

$$\begin{aligned} & \exists car_1, car_2, car_3, car_4, load_1, load_2, \dots \\ & [infront(car_1, car_2)] [infront(car_2, car_3)] \dots [length(car_1) = long] \wedge \\ & [car-shape(car_1) = engine] [car-shape(car_2) = U-shaped] [cont-load(car_2, load_1)] \wedge \quad (12) \\ & [load-shape(load_1) = triangle] \dots [nrwheels(car_3)] \dots :> [class = Eastbound] \end{aligned}$$

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

$$[car-shape = open\ rctngl, open\ trapezoid, U-shaped, dbl\ open\ rctngl] \Rightarrow [car-shape = open\ top] \quad (13)$$

$$[car-shape = ellipse, closed\ rctngl, jagged\ top, sloping\ top] \Rightarrow [car-shape = closed\ top] \quad (14)$$

$$[load-shape = hexagon, triangle, rectangle] \Rightarrow [load-shape = polygon] \quad (15)$$

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 able 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:

---

\* At this moment, before proceeding further, the reader is advised to look at the pictures and to try to solve this problem on his/her own.

- [nrcars=4] - the number of cars in the train is 4  
(the length of chain defined by relation  
infront)
- [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<sub>i</sub>)=i] - the position of car<sub>i</sub> is i

Suppose that eastbound trains are considered first. The set F1 contains then all c-expressions describing eastbound trains, and FC, 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' metadescrptors generated by problem environment rules and constructive generalization rules. In this case, the metaselector [shape(last-car)=rectangle] is added to  $e$ . Next, a set G (a *restricted star*) of certain number (NCONSIST) 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 ( $P_1$ ) contains the set of all selectors of  $e_1$ . 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 [12]) 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 ( $P_1$ )) was:

$$\exists \text{car}_1 [\text{car-shape}(\text{car}_1) = \text{U-shaped}] \quad (16)$$

$$\exists \text{car}_1 [\text{car-shape}(\text{car}_1) = \text{open trapezoid}] \quad (17)$$

$$\exists \text{car}_1 [\text{car-shape}(\text{car}_1) = \text{rectangle}] \quad (18)$$

$$[\text{car-shape}(\text{last-car}) = \text{rectangle}] \quad (19)$$

The last c-expression is consistent (has empty intersection with c-expressions in  $F_0$ ) and, therefore, is placed in  $G$ . From the remaining, a new partial star is determined. This new partial star contains a consistent generalization:

$$\exists \text{car}_1 [\text{car-shape}(\text{car}_1) = \text{rectangle}] [\text{length}(\text{car}_1) = \text{short}] \quad (20)$$

which is added to  $G$ . Suppose  $G$  is restricted to have only two elements ( $\text{NCONSIST}=2$ ). 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:

$$\exists \text{car}_1 [\text{car-shape}(\text{car}_1) = \text{closed top}] [\text{length}(\text{car}_1) = \text{short}] \quad (21)$$

(the generalization of (19),  $[\text{car-shape}(\text{last-car}) = \text{polygon}]$ , is not complete; it does not cover all  $F_1$ ).

In this example, only 2 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 NCONSIST, 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:*

$$\exists \text{car}_1 [\text{length}(\text{car}_1) = \text{short}] [\text{car-shape}(\text{car}_1) = \text{closed top}] \text{::>} [\text{class} = \text{Eastbound}] \quad (22)$$

(the same as (21)). It can be interpreted:

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

$$\begin{aligned} \exists \text{car}_1, \text{car}_2, \text{load}_1, \text{load}_2 [\text{infront}(\text{car}_1, \text{car}_2)] [\text{cont-load}(\text{car}_1, \text{load}_1)] \\ \wedge [\text{cont-load}(\text{car}_2, \text{load}_2)] [\text{load-shape}(\text{load}_1) = \text{triangle}] \\ \wedge [\text{load-shape}(\text{load}_2) = \text{polygon}] \text{::>} [\text{class} = \text{Eastbound}] \quad (23) \end{aligned}$$

It can be interpreted:

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:*

$$[\text{nrcars} = 3] \vee \exists \text{car}_1 [\text{car-shape}(\text{car}_1) = \text{jagged-top}] \text{::>} [\text{class} = \text{Westbound}] \quad (24)$$

$$\exists \text{car}_1 [\text{nrcars-length-long} = 2] [\text{position}(\text{car}_1) = 3] [\text{shape}(\text{car}_1) = \text{open-top, jagged-top}] \text{::>} [\text{class} = \text{Westbound}] \quad (25)$$

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

## 7. SUMMARY

We have presented an approach to pattern recognition which views it as knowledge-guided computer induction. Let us briefly review the main advantages and limitations of this approach. Among the advantages are the generality of the method and the simplicity of interpretation of the pattern recognition rules. More specifically, the approach:

- takes into consideration three types of descriptors (nominal, linear and structured) and can use descriptors of different arity (variables, n-ary relations and functions)
- takes into consideration the properties of the interrelationships of descriptors, characteristic to the recognition problem at hand
- gives the possibility of defining (within limits) a preference criterion, measuring the quality of the rules, that is most suited to the application
- has an ability to generate new descriptors ('metadescriptors') and blend them smoothly with the initial ones to provide a basis from which the final description chooses its most appropriate descriptors
- provides uniformity of the representation of initial and final descriptions (i.e., in terms of VL rules) and of inference and generalization rules
- permits the person stating the problem to suggest various arithmetic transformations of the original (linear) variables which look promising as relevant characterization of object classes.

Among major limitations of the presented work is a quite limited form of expressing initial and final descriptions (i.e., in the form of a disjunctive simple VL<sub>21</sub> expressions), and a restricted number of operators the program (implementing the approach) understands and uses

in inducing descriptions. Another limitation is that the program does not differentiate among possible types of linear descriptors (e.g., ordinal, interval, ratio and absolute). Also, it does not take into consideration any probabilistic information, nor it is able to automatically search for appropriate algebraic transformations. These limitations do not seem, however, to be inherent to the approach.

Also, the questions pertinent to the computational efficiency of algorithms used have not been investigated.

#### ACKNOWLEDGEMENT

The research presented here has been supported in part by the National Science Foundation Grant NSF MCS 76-22940.

The author acknowledges the collaboration with James 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-1. Among many people who helped through discussions and through their interest in the work, the author would like to specially mention K. S. Fu, Donald Michie, Brian Gaines, Raj Reddy, Len Uhr, Larry Travis, A. B. Baskin and Tom Dietterich.



REFERENCES

- [ 1] Buchanan, B. G., Mitchell, T., Model-directed learning of production rules, Computer Science Department, Report No. STAN-CS-77-597, Stanford University, March 1977 .
- [ 2] Winston, P. H., Learning structural descriptions from examples, Tech. Rep. AI TR-231, MIT AI Lab, Cambridge 1970.
- [ 3] Lenat, D. B., AM: An artificial intelligence approach to discovery in mathematics as heuristic search, Computer Science Department, Report No. STAN-CS-76-570, Stanford University, July 1976.
- [ 4] Soloway, E. M. and Riseman, E. M., Levels of pattern description in learning, Proceedings of the 5th International Joint Conference on Artificial Intelligence, August 22-25, MIT, 1977.
- [ 5] Simon, H. A., Complexity and the representation of patterned sequences of symbols, Psychological Review, Vol. 79, pp. 369-382, 1972 .
- [ 6] Waterman, D. A., Adaptive production systems, Working paper #285, Department of Psychology, Carnegie-Melon University, Pittsburgh, 1974 .
- [ 7] Gaines, B. R., Behavior/structure transformations under uncertainty, Int. Journal on Man-Machine Studies, Vol. 8, pp. 337-365, 1976.
- [ 8] Shaw, D. E., Swartout, W. R. and Green, C. C., Inferring Lisp programs from examples, Proceedings of the 4th International Joint Conference on Artificial Intelligence, Vol. I, pp. 351-356, Tibilisi, September 1975.
- [ 9] Feldman, J. A., Gips, J., Horning, J. J., and Reder, S. Grammatical complexity and inference, CS report No. 125, Computer Science Department, Stanford University 1969 .
- [10] Brayer, J. M., Fu, K. S., Web grammars and their application to pattern recognition, TR-EE 75-1, School of EE, Purdue University, December 1975.
- [11] Michalski, R. S., A variable-valued logic system as applied to picture description and recognition, GRAPHIC LANGUAGES, eds. F. Nake and A. Rozenfeld, North-Holland 1972.
- [12] Michalski, R. S., AQVAL/1--computer implementation of a variable-valued logic system and the application to pattern recognition, Proceedings of the First International Joint Conference on Pattern Recognition, Washington, D.E., October 30-November 1, 1973.

- [13] Larson, James, A multi-step formation of variable-valued logic hypotheses, Proceedings of the Sixth Annual International Symposium on Multiple-Valued Logic at Utah State University, May 25-28, 1976.
- [14] Stoffel, J. C., The theory of prime events: data analysis for sample vectors with inherently discrete variables, Information Processing 74, North-Holland Publishing Company, pp. 702-706, 1974.
- [15] Morgan, C. G., Automated hypothesis generation using extended inductive resolution, Advance Papers of the 4th I. J. Conf. on Artificial Intelligence, Vol. I, pp. 351-356, Tbilisi, Georgia, September 1975.
- [16] Plotkin, G. D., A further note on inductive generalization. In Machine Intelligence 6, B. Meltzer and D. Michie, Eds., American Elsevier, New York, 1971.
- [17] Fikes, R. E., Hart, R. E. and Nilsson, N. J. Learning and executing generalized robot plans, Artificial Intelligence 3, 1972.
- [18] Hayes-Roth and McDermott, J. An interference matching technique for inducing abstractions, Communications of the ACM, No. 5, Vol. 21, pp. 401-411, May 1978.
- [19] Vere, S., Induction of concepts in the predicate calculus, Advance Papers of the 4th I. J. Conf. on Artificial Intelligence, Vol. I, pp. 351-356, Tbilisi, Georgia, September 1975.
- [20] Zagoruiko, N. G., Iskustviennui intellekt i empiricheskoe predskazanie, Novosibirskij Gosudarstviennyi Universitet, 1975.
- [21] Hedrick, C. L., A computer program to learn production systems using a semantic net, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, July 1974.
- [22] Larson, J., Michalski, R. S., 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.
- [23] Larson, James, Inductive inference in the variable-valued predicate logic system VL<sub>21</sub>: methodology and computer implementation, Ph.D. Thesis, Report No. UIUCDCS-R-77-869, Department of Computer Science, University of Illinois, Urbana, May, 1977.

- [24] Larson, James, INDUCE-1: an interactive inductive inference program in VL<sub>21</sub> logic system, Report No. UIUCDCS-R-77-876, Department of Computer Science, University of Illinois, Urbana, May, 1977 .
- [25] Dietterich, T., INDUCE 1.1 - the program description and a user's guide, Internal Report, Department of Computer Science, University of Illinois, Urbana, July, 1978.
- [26] Coulon, D., Kayser, D., Learning criterion and inductive behaviour, Pattern Recognition, Vol. 10, No. 1, pp. 19-25, 1978.
- [27] Michalski, R. S., A system of programs for computer-aided induction: a summary, 5th International Joint Conference on Artificial Intelligence, MIT, Boston, Massachusetts, August, 1977.

<b>BIBLIOGRAPHIC DATA SHEET</b>	1. Report No. UIUCDCS-R-78-927	2.	3. Recipient's Accession No.
	4. Title and Subtitle PATTERN RECOGNITION AS KNOWLEDGE-GUIDED COMPUTER INDUCTION		5. Report Date June 15, 1978
7. Author(s) Ryszard S. Michalski	9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, IL 61801		8. Performing Organization Rept. No.
12. Sponsoring Organization Name and Address National Science Foundation Washington, D.C.		10. Project/Task/Work Unit No.	11. Contract/Grant No. NSF MCS 76-22940
15. Supplementary Notes		13. Type of Report & Period Covered	
16. Abstracts The determination of pattern recognition rules is viewed as a problem of computer induction, under the guidance of <i>generalization rules</i> and rules representing knowledge of the recognition problem at hand. The paper formulates the underlying theory for generalization and optimization of descriptions of object classes, expressed in the form of decision rules. The language for formulating descriptions is an extension of the first order predicate calculus, called the variable-valued logic calculus VL <sub>21</sub> . VL <sub>21</sub> contains several new syntactic forms, specially oriented for expressing inductive processes. The presented approach uniformly combines descriptors (variables, predicates, functions) of three different types: nominal, linear and structured, and has an ability to generate new descriptors not used in the initial data rules.		14.	
17. Key Words and Document Analysis. 17a. Descriptors pattern recognition, decision rules, generalization techniques, computer induction, knowledge-acquisition, learning from examples, many-valued logic, computer inference, computer consulting systems, theory formation, inductive inference		17b. Identifiers/Open-Ended Terms	
17c. COSATI Field/Group		19. Security Class (This Report) UNCLASSIFIED	
18. Availability Statement		20. Security Class (This Page) UNCLASSIFIED	21. No. of Pages
		22. Price	