INDUCTIVE INFERENCE OF VL DECISION RULES

by

J. B. Larson
R. S. Michalski

Invited paper for the Workshop in Pattern-Directed Inference Systems, Hawaii, May 23-27, 1977 and published in SIGART Newsletter, ACM, No. 63, pp. 38-44, June 1977.

Workshop on: PATTERN - DIRECTED
INFERENCE SYSTEMS

Hawaii, May 23-27, 1977

INDUCTIVE INFERENCE OF VL DECISION RULES

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

(217) 333-6725

Abstract

The problem considered is a transformation of a set of user given decision rules into a set of new rules which are more general than the original ones and more optimal with regard to a user defined criterion. The decision rules are expressed in the VL_{21} logic system which permits a more general rule format than typically used, and facilitates a compact and easy to understand expression of descriptions of different degrees of generality. The paper gives a brief description of methodology for rule induction and of a computer program.

INDUCTIVE INFERENCE OF VL DECISION RULES

J. Larson and R. S. Michalski

1. INTRODUCTION AND STATEMENT OF THE PROBLEM

The paper reports recent results on the development of a methodology and a computer program for generalization and optimization of VL decision rules. The VL decision rules are similar to production rules used by various authors (e.g., Shortliffe 74, Waterman 75, Rychener 76) in that they follow a general schema:

CONDITION ⇒ DECISION

That is, if a *situation* satisfies the CONDITION, then the rule assigns to it DECISION, otherwise NULL decision is assigned. The difference between a usual form of decision rules and VL rules is that in the latter CONDITION and DECISION can have a more general form, namely a form of an expression in the VL_2 logic system (Michalski 76).The system can be viewed as a multi-valued form of the first order predicate logic , with some additional operators. In this paper we restrict ourselves to a subset of VL_2, called VL_{21}, in which formulas can have truth status only TRUE, FALSE or UNKNOWN, and certain VL_2 forms are not permitted (see section 2).

The goal of this work is to ultimately develop a set of programs which can aid a computer user in solving certain types of inductive problems.

The specific induction problem we are investigating is as follows:

Authors gratefully acknowledge the support from the National Science Foundation, Grant NSF MCS 74-03514, for conducting the research reported in this paper.

- Given is a set of decision rules

$$
\begin{array}{llll}
C_{11} \Rightarrow D_1, & C_{12} \Rightarrow D_1 & \cdots\cdots & C_{1t1} \Rightarrow D_1 \\
C_{21} \Rightarrow D_2, & C_{22} \Rightarrow D_2 & \cdots\cdots & C_{2t2} \Rightarrow D_2 \\
\vdots & & & \\
C_{m1} \Rightarrow D_m, & C_{m2} \Rightarrow m & \cdots\cdots & C_{mtm} \Rightarrow D_m
\end{array}
\qquad (1)
$$

where C_{ij} and D_i are expressions in VL_{21}, which represent a CONDITION and DECISION of decision rules, respectively. For example, C_{ij} may look like

$$\exists x_1, x_3\ ([p(x_1,x_2,x_3) = 0,1][x_2 = 2..5])$$

or $$[t(x_2,x_5) \neq 3][x_8 \geq 3][x_9 = 0]$$

(For the explanation of the formalism see section 2.3.) It is also assumed that for any situation either NULL decision or only one DECISION (i.e., a specific D_i) is assigned by the rules (1). The rules (1) may represent specific examples of situations to which a certain decision is assigned (e.g., learning examples in pattern recognition), or they may represent a partial knowledge that a certain set of situations should be assigned a given decision.

In this paper we make a simplifying assumption that variables and atomic functions in C_{ij} do not occur in D_i's.

- The problem is to determine, through an application of generalization rules, (See section 3.3) a new set of decision rules:

$$
\begin{array}{l}
C_1 \Rightarrow D_1 \\
C_2 \Rightarrow D_2 \\
\vdots \\
C_m \Rightarrow D_m
\end{array}
\qquad (2)
$$

(within the framework of the same language, in this case VL_{21}) which are, with regard to rules (1):

1. consistent
2. complete
3. optimal (according to a user defined criterion) among all the rules which satisfy 1 and 2.

The rules are *consistent* if for any situation for which the new rules assign a DECISION (i.e., a non NULL decision), the initial rules assign the same DECISION or a NULL decision. The rules are *complete* if for any situation for which the initial rules assign a DECISION, the new rules assign a DECISION. (It is easy to see that if the new rules are consistent and complete with regard to the initial rules, then they are equivalent to or more general than the initial ones.)

From an initial set of rules, it is usually possible to derive very many sets of rules which are consistent and complete. Therefore, a criterion of optimality (defined by a user according to his problem) is used to select one or a few alternative sets of rules which are most desirable according to the given inductive problem (such a criterion may refer in some sense to the simplicity of the rules, the memory required for storage, the time for rule evaluation, the cost of measuring the information needed for rule evaluation, etc.).

In this paper we briefly describe the format of VL_{21} decision rules (for short VL rules), generalization rules, the methodology for solving the inductior problem and a computer implementation. Chapter 2 and 3 use to a large extend material in Michalski 76.

2. VL FORMULAS

2.1 The Concept of a Selector

The logic system VL_{21} is used here as a language for describing situations (objects, classes of objects, etc.) and expressing decision and inference rules. It was especially designed to facilitate a compact expression of descriptions of different degrees of generality and to provide an easy linguistic interpretation of the descriptions (or rules) without loosing the precision of other logic systems.

There are two major differences between VL_{21} and the first order predicate logic:

1. Instead of predicates it uses *selectors* which can be viewed as tests for membership of values of predicates and atomic functions in a certain set. (The concept of a selector is useful for compactly expressing descriptions of different degrees of generality.)

2. Each variable, predicate and function symbol is assigned a domain (value set) together with a characterization of the structure of the domain. (This feature facilitates the process of rule induction and an application of different generalization rules according to the structure of the domains.)

For the lack of space, we will give here only a brief informal description of the concept of selector and of VL_{21} system.

Definition 1: An *atomic form* is a constant, variable or a function symbol followed by a pair of parentheses enclosing a sequence of atomic forms.

Examples of atomic forms:

a, b, carl	- constants
x_1, x_2, f, g, h, color	- variables
$f(a,b,g(x_3),x_4)$, $on\text{-}top(x_1,x_2)$, $color(carl)$	- function symbols followed by a pair of parentheses enclosing a sequence of atomic functions

If a variable or function symbol has the domain {T, F}, where T - true, F - false, then it is called a *predicate variable* or *predicate symbol*, respectively.

A single predicate variable or a predicate symbol followed by a pair of parentheses enclosing a sequence of atomic forms is called a *predicate form*.

Let $f(x_1, x_2, \ldots, x_k)$ be an atomic form and $D(f)$, $D(x_1)$, $D(x_2)$, $\ldots$, $D(x_k)$ denote domains of function symbol f and variables $x_1, x_2, \ldots, x_k$, respectively. An atomic form represents a function (called an *atomic function*) which maps the cartesian product of the domains, $D(x_i)$, $i = 1, 2, \ldots, k$ into $D(f)$:

$$f: \; D(x_1) \times D(x_2) \times \ldots \times D(x_k) \rightarrow D(f)$$

A concept of selector is introduced to represent a unit of information about a situation. The selector can have various truth status with regard to a situation and is used as an argument of various logical operations.

Definition 2: A *selector* is a form:

$$[L \; \# \; R] \tag{3}$$

where L - called *referee*, is an atomic form,

\# is one of the symbols: $=, \neq, \geq, >, \leq, <$

R - called *reference*, is either

a list of elements from the domain D (if the list contains a sequence of consecutive integers from a to b, then it is written as a..b), or

an atomic form, or

symbol '*' (an irrelevant value).

A selector in which the reference R is not an atomic form is called a *simple selector*. A selector in which the referee L is a single variable and reference R is a single element of D or * is called an *elementary selector*.

If R is a list, then L *is related to R by* # if:

when '#' is '=' or '$\neq$'	then the value of L for given values of arguments *is* or *is not*, respectively, on the list R,
when '#' is '$\geq$' or '>' or '$\leq$' or '<'	then the value of L is greater or equal, greater, smaller or equal, smaller, respectively, than any element on the list R.

If R is an atomic form, then L *is related to* R if the value of L is related by # to the value of atomic form. If R is *, then L *is related to* R for any value of L (in this case, # is always '=').

A selector can have, with regard to a given situation, truth-status (briefly, TS), UNKNOWN (?), TRUE (T) or FALSE (F). If a selector has truth-status UNKNOWN (TS = ?), then it is interpreted as a condition or a question. If it has truth-status TRUE (TS = T), then it is interpreted as a statement that 'value of L is related by # to R.'

Examples of a Selector

		Interpretation when TS = T
(i)	[color(wall) = white]	color of wall is white
(ii)	[length(box 1) $\geq$ 2]	the length of box 1 is greater than or equal to 2
(iii)	[blood-type(P1) = O,A]	the blood type of P1 is O or A
(iv)	[weight(B) = 2..5]	weight of B is an integer between 2 and 5, inclusively
(v)	[on-top(B1, B2) = T] or simply [on-top(B1, B2)]	it is true that B1 is on top of B2
(vi)	[above(B1, B2) = 3"]	B1 is 3 inches above B2
(vii)	[weight(A) > weight(B)]	weight of A is greater than weight of B

3. VL DECISION RULES AND VL GENERALIZATION RULES

3.1 Definition of VL Decision Rules

VL decision rules are in the form:

$$V_1 \Rightarrow V_2 \tag{4}$$

(an implicative decision rule)

where V_1 and V_2 can, in general, be any VL formulas. In the implicative rule (4), formula V_1 is called a *condition formula* (or *condition part*) and V_2 is called a *decision formula* (or *decision part*).

The interpretation of an implicative rule (4) is that if the condition formula V_1 reaches truth status TRUE, then the truth status TRUE is assigned to the decision formula V_2. Variables and atomic functions in V_2 are assumed to take values which make it TRUE.

3.2 The Use of VL Decision Rules

A decision rule is used by applying it to *situations*. A situation is, in general, a source of information about values of variables and atomic functions in the condition part of a decision rule. A situation can, e.g., be a data base storing values of variables and atomic functions, or it can be an object on which various tests are performed to obtain these values.

The truth status of the condition and decision formula in 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 *descriptor set*, which consists of variables, predicate and atomic functions (called, generally, descriptors) whose specific values can characterize adequately (for the problem at hand) any specific situation. We will assume here that arguments of predicate and 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 form with specific values of arguments, and v is a value of the given descriptor (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 descriptor 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. A set of all events possible assuming a given descriptor set S is called the *event space*, and denoted **E**(S). It should be noted that certain variables (variables which are quantified in formulas) may have in an event 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, ... represent different values.

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

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

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 a condition part of a decision rule is a quantifier-free formula. Interpreting the connectives $\neg$, $\wedge$, $\vee$, as described in section 2.3, from the truth status of selectors one can determine 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

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

is satisfied by an event:

e = (... part:=P1, color(P1):=blue, part:=P2, color(P2):=yellow,

part:=P3, color(P3):=red...)

If the condition formula 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 formula of an implicative rule is assigned truth status TRUE then the decision formula is assigned status TRUE. If a decision formula reaches status TRUE, then variables and functions which occur in it are assumed to have values (possibly new) which make this formula TRUE. This value may not, in general, be unique.

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

$$V: \quad [p(x_1,x_2) = 2][x_3 = 2{:}5][x_5 \neq 7]$$

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 a decision rule can then be described as follows: Therule 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 formula). Another decision 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 implicative VL decision rules:

$$[p(x_1, x_2) = 3][q(x_2) = 2,5][x_7 \neq 0] \Rightarrow [d(y_1) = 7][p(y_1, y_2) = 2]$$

$$\exists x_3([p(x_1, x_3) = 2..3][q(x_7, x_3) \geq 2]) \vee [t(x_1) = 1] \Rightarrow [d(y_1) = 7]$$

$$T \Rightarrow [p(x_2, x_7) = 2][x_7 = 2,3,5]$$

3.3 Examples of VL Generalization Rules

In order to transform an original set of decision rules (1) into a new set of rules (2), a condition of optimality is defined, and VL generalization rules are applied to the original rules. VL generalization rules are inductive inference rules which transform one or more VL decision rules into a new decision rule which is an equivalent or more general one.

A decision rule

$$V \Rightarrow D \tag{5}$$

is *equivalent* to a set of decision rules

$$\{V_i \Rightarrow D\},\ i = 1, 2, \ldots \tag{6}$$

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

The generalization rules are a tool for transforming initial decision rules. In order to obtain rules (2), the transformation is carried on in such a way that the conditions of consistency, completeness and optimality are satisfied. Section 4 outlines an algorithm for such a process.

Below are examples of VL generalization rules (the symbol $\vdash$ is used to denote the transformation from a set of rules into one more general rule).

(i) the *'dropping selector'* rule

$$V[L = R_1] \Rightarrow D \quad \vdash \quad V \Rightarrow D \qquad \text{(V - any VL formula)}$$

This rule is generally applicable, no matter what kind of descriptor is in the selector.

(ii) the *'extension against'* rule

$$\left.\begin{array}{l} V_1[L = R_1] \Rightarrow D \\ V_2[L = R_2] \Rightarrow \neg D \end{array}\right| \vdash \quad V_1[L \neq R_2] \Rightarrow D \quad (V_1, V_2 \text{ - any VL formulas})$$

assuming $R_1 \cap R_2 = \emptyset$

This rule is also generally applicable. It takes into consideration the 'negative examples.'

(iii) the *'closing interval'* rule

$$\left.\begin{array}{l} V[L = a] \Rightarrow D \\ V[L = b] \Rightarrow D \end{array}\right| \vdash \quad V[L = a..b] \Rightarrow D$$

where L is an interval descriptor

This rule is applicable only when a selector involves an interval variable.

(iv) the *'finding next level generalization'* rule

$$\left.\begin{array}{l} V[L = a] \Rightarrow D \\ V[L = b] \Rightarrow D \end{array}\right| \vdash \quad V[L = c] \Rightarrow D$$

where L is a structural descriptor,

c represents the predecessor of the nodes a and b in the tree domain of L.

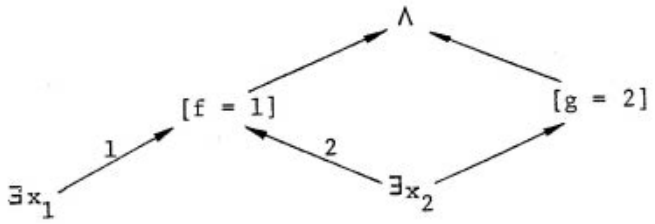
This rule is applicable only to selectors involving structural descriptors.

4.1 Computer Representation of VL Decision Rules

A VL decision rule can be represented as a graph with labeled nodes and directed labeled edges. The labels on the nodes can be: a) a selector containing k-ary descriptors without argument lists, b) a k-ary descriptor without arguments, c) a quantified variable, d) a logical operator. (From here on, we refer to a node by its label, e.g., a selector node means a node with a selector label.) The edges are labelled with integers from 0, 1, ... Edges not labelled 0 refer to the position of an argument in the label at the head of the edge. (Edges have non-0 labels only if the position in the argument list of the head node is important. Labels of 0 may be dropped for convenience.)

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

Figure 2 gives a graph of a VL_2 formula. 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 and g on x_1 and x_2.



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

Figure 2

4.2 Outline of an Algorithm for Rule Induction and Computer Programs INDUCE-1

In the algorithm implemented in the program INDUCE-1, the following assumptions have been made: A1) The condition part of every decision rule is in the form of a sequence of zero or more existentially quantified variables followed by a disjunction of products of selectors ('existential disjunctive form' or EDF); A2) It is further assumed that the condition part of every c-rule (*c-condition*) can be represented by a weakly connected (directed) graph whose edges represent only functional dependence; A3) In every existential quantifier form

$$\exists x_1, x_2, \ldots, x_k$$

each x_i, $i = 1, 2, \ldots, k$ is assumed to have the same domain. A quantifier form postulates an existence of a sequence $x_1, x_2, \ldots, x_k$ in which each x_i have different value $(x_i \neq x_2 \neq \ldots \neq x_k)$. In a single EDF, there can be several different quantifier forms.

The algorithm described here uses to a large extent ideas and algorithms developed for the generalization of VL_1 expressions (Michalski 75, Larson and Michalski 75).

The basic input information to the algorithm consists of:

a) a set of decision rules, such as (1)

and

b) the optimality criterion.

All rules are assumed to be c-rules, and they are grouped into decision classes, each class consisting of rules with the same decision part. The algorithm is applied to each decision class separately. Let $F1 = \{e_1, e_2, \ldots, e_k\}$ - a set of condition parts (c-condition) of rules in one

class, and F0 - a set of condition parts of all the rules from the remaining classes.

An e_i is selected from F1 and a set of formulas is generated (G) which are different irredundant generalizations of e_i but not a generalization of any formula in F0. (If a formula e_1 is more general than e_2, then we say that e_1 *covers* e_2.) By an *irredundant generalization*, we mean a c-formula from which no selector can be removed without violating the consistency condition (i.e., that the formula will not cover any elements of F0).

From G, one c-formula is selected, LQ, which is the 'best' according to the optimality criterion. All elements of F1 which are covered by LQ are removed from F1. Another formula is selected from F1, and the whole process is repeated again until F1 becomes an empty set. To implement this process, a number of specific algorithms have been developed, each accomplishing one basic task. Among the most important tasks are:

T1. Formation of generalizations of a c-formula

T2. Testing whether one c-formula is a generalization of another c-formula

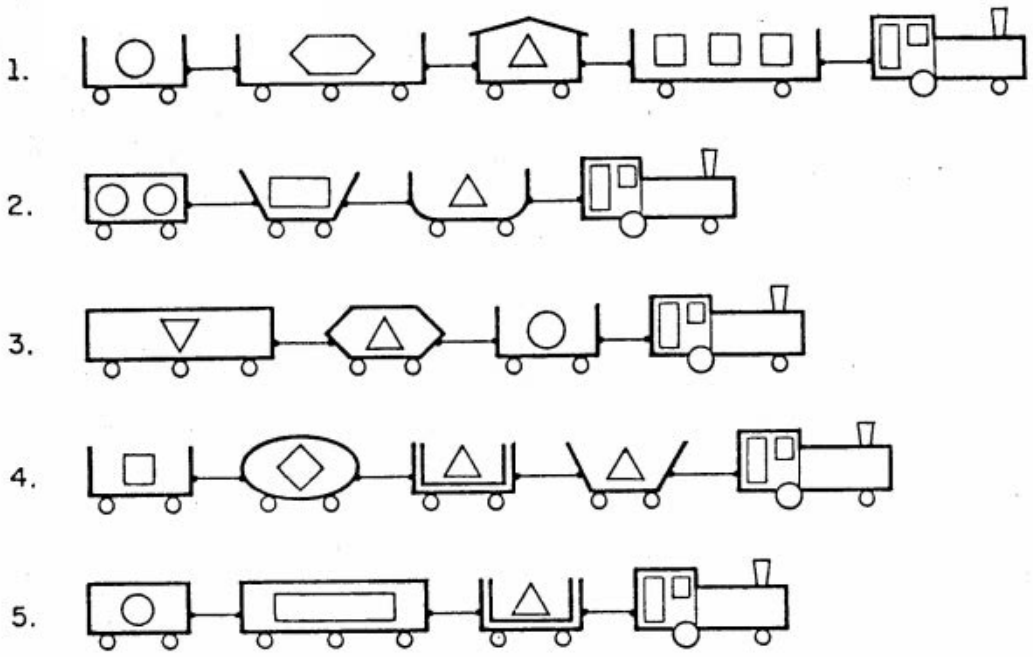
T3. Generalization of a c-formula by extending the selector references and formation of irredundant c-formulas

T4. Generation of metadescriptors (or inferred descriptors) which are functions of the initial descriptors. The current program generates only unary descriptors (such as 'number of objects or parts which show one or more properties').

Algorithms for solving these tasks will be published in a separate paper.

The computer program INDUCE-1 which implements the above inductive algorithm has been written in PASCAL for the CDC CYBER-175. It contains approximately 3000 PASCAL statements and 40 basic procedures.

1. TRAINS GOING EAST



2. TRAINS GOING WEST

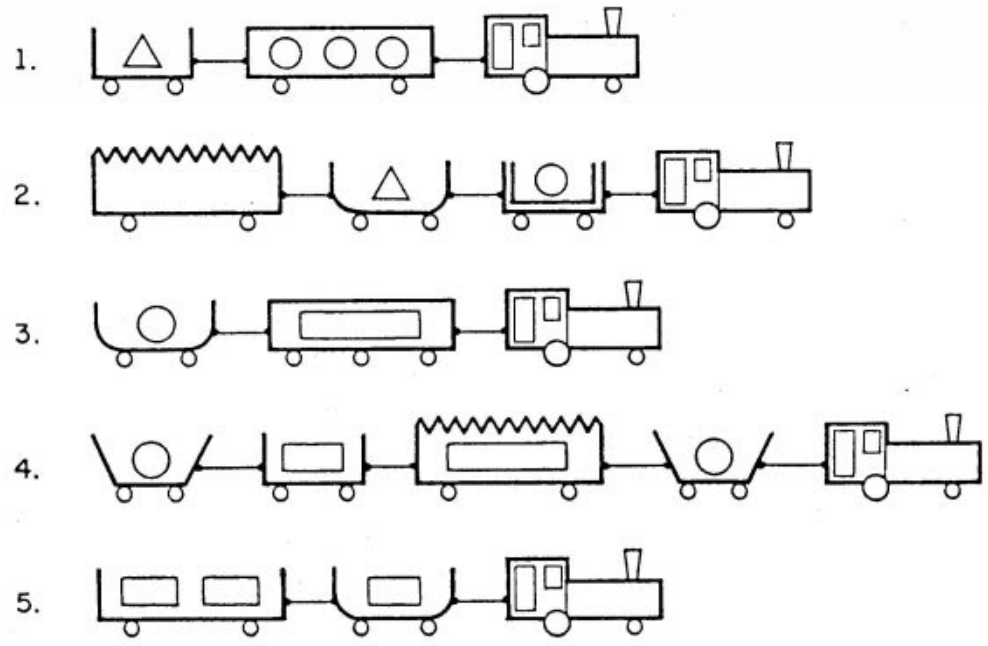


Figure 3

5. EXAMPLE

Given are two sets of trains: Eastbound and Westbound; and the problem is to determine concise decision rules distinguishing between these two sets (Figure 3). The descriptor set for the trains involve such descriptors as:

NCARS	number of cars (an interval descriptor),
CAR-SHAPE (car)	shape of a car (a structural descriptor with 12 nodes in the generalization tree),
LN (car)	length of a car,
INFRONT (car_i, car_j)	a predicate indicating that car_i is in front of car_j,
LCONT (car_i, lod_j)	a predicate indicating that car_i contains load lod_j,
LOAD-SHAPE (lod_i)	a shape of the load lod_i (a structural descriptor with 5 nodes in the generalization tree),

and some others, total 12 descriptors.

Each train is input to the program as a VL decision rule involving 12 descriptors. For the purpose of the following discussion, our attention will be limited to only a subset of these descriptors. The decision rule associated with train number 2 going EAST using a descriptor subset is the following (assume that all function arguments are existentially quantified):

[NCARS = 4][IF (CAR_1, CAR_2)][INFRONT (CAR_2, CAR_3)][INFRONT (CAR_3, CAR_4)]
[CAR-SHAPE (CAR_1) = ENGINE][CAR-SHAPE (CAR_2) = U-SHAPED]
[CAR-SHAPE (CAR_3) = OPEN-TRAPEZOID]
[CAR-SHAPE (CAR_4) = RECTANGLE][LN (CAR_1) = LONG]
[LN (CAR_2) = SHORT][LN (CAR_3) = SHORT][LN (CAR_4) = SHORT]

$\Rightarrow$ [D = EAST]

Two sets of formulas are created from the description of each set of trains (the set F1 contaning descriptions of trains going EAST, the set F0 containing descriptions of trains going WEST). An e_i is selected from the set F1 (assume that e_i is the description of train number 2).

The object is to generate a set G (a star) of consistent generalizations of e_i. The program forms a sequence of partial starts (a *partial star* is a set of generalizations of e_i which may be inconsistent). If an element of a partial star is consistent, it is placed into the star G. The initial partial star (P_1) contains the set of all selectors of e_i. This partial star and each subsequent partial star is reduced according to a user specified optimality 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 such that for each product in P_i, a set of products is placed into P_{i+1} where each new product contains the selectors of the original product plus one new selector of e_i which is not in the original product. Once a sufficient number of consistent generalizations have been formed, a version of the AQVAL/1-AQ7 program [Larson-Michalski 75] 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 (i.e., the reduced partial star P_1) was:

{[CAR-SHAPE (CAR_1) = U-SHAPED],
[CAR-SHAPE (CAR_2) = OPEN-TRAPEZOID],
[CAR-SHAPE (CAR_3) = RECTANGLE]}

The criterion used here was to maximize the number of events covered in F1 and with secondary priority to minimize the number of selectors in a c-condition. Since none of these are consistent (i.e., there is at least one element in F0 with each of these properties), a new partial star is formed. This partial star contains the consistent generalization:

[CAR-SHAPE (CAR_1) = RECTANGLE] [LN (CAR_1) = SHORT]

so the consistent generalization is placed in the star G.

The value in the reference of each selector is then generalized by the program AQVAL/1-AQ7 to form the consistent, complete generalization

[CAR-SHAPE (CAR_1) = CLOSED-TOP][LN (CAR_1) = SHORT].

In this abbreviated example, only 2 partial stars were formed, and one consistent generalization was 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.

Using the complete description of each train, the program produced the following alternative decision rules using as optimalily criterion the number of c-conditions, and with lower priority, the number of selectors (computation time on a CDC-CYBER 175: 10 seconds):

Eastbound trains (EB):

(i) $\exists CAR_1, CAR_2, LOD_1, LOD_2$ [INFRONT (CAR_1, CAR_2)][LCONT (CAR_1, LOD_1)]
[LCONT(CAR_2, LOD_2)][LOAD-SHAPE(LOD_1) = TRIANGLE]
[LOAD-SHAPE(LOD_2) = POLYGON] $\Rightarrow$ [D = EB]

(ii) $\exists CAR_1$[LN(CAR_1) = SHORT][CAR-SHAPE(CAR_1) = CLOSED TOP] $\Rightarrow$ [D = EB]

Westbound trains (WB):

(iii) [NCAR = 3] $\vee$ $\exists CAR_1$[CAR-SHAPE(CAR_1) = JAGGED TOP] $\Rightarrow$ [D = WB]

(iv) $\exists CAR_1$[#CARS (LN=LONG) = 2]
[CSHAPE (CAR_1) = OPEN RECTANGLE, U-SHAPED] $\vee$
[LOCATION (CAR_1) = 2][CSHAPE (CAR_1) = CLOSED RECTANGLE]
$\Rightarrow$ [D = WB]

The first selector in rule (iv) uses a meta descriptor generated by the program which counts the number of long cars in a train.

It is interesting to note that the authors constructed the example with rules (i) and (iii) in mind. The rule (ii) found by the program as an alternative was rather surprising because it seems to be conceptually simpler than the author's original rule (i). This observation indicates one of potential applications of the program, namely as a tool to determine alternative hypotheses.

6. SUMMARY

Let us briefly review the main advantages and limitations of the presented method of induction. Among the advantages are generality and simplicity. Due to these features, the program INDUCE-1 has a potential to be applied to a variety of inductive tasks. Also, it is easy to comprehend what kind of initial information it needs and how to interpret the results. Among other, more specific properties are:

- An ability to take into consideration different structures of value sets of descriptors and various properties of descriptors pertinent to the given practical problem.
- A possibility to specify different criteria of optimality for the inferred descriptions or rules.
- An ability to generate new descriptors (meta descriptors) and blend them smoothly with the initial ones to provide a basis from which the final description chooses its most appropriate descriptors.
- A uniformity of the representation of initial and final descriptions (i.e., in terms of VL rules) and their simplicity.

A major limitation of the presented work is the quite limited number of operators which the program understands and uses in inducing descriptions. Another limitation is that induction is done only on the left hand side of the decision rules. These limitations do not seem, however, to be inherent to the approach. We have also not yet investigated questions pertinent to the computational efficiency of algorithms and adequacy of the accepted data structures.

References

Hayes, Roth F., and McDermott, J. Knowledge Acquisition from Structural Descriptions. Departmental Report, Department of Computer Science, Carnegie-Melon University, Pittsburgh, 1976.

Larson, J. and Michalski, R. S. AQVAL/1(AQ7) User's Guide and Program Description. Report 731, Department of Computer Science, University of Illinois, Urbana, 1975.

Michalski, R. S. Synthesis of Optimal and Quasi-Optimal Variable-Valued Logic Formulas. Proceedings 5th International Symposium on Multiple-Valued Logic, Bloomington, Indiana, 1975.

Michalski, R. S. STUDIES IN INDUCTIVE INFERENCE: An Approach Utilizing Variable-Valued Logic. (To appear in a published form.) A Proposal to NSF, 1976.

Morgan, C. G. Inductive Resolution. Master's Thesis, Department of Computer Science, University of Alberta, Edmonton Alberta, 1972.

Rychener, M. D. The Student Production System. A Study of Encoding Knowledge in Production Systems, Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, October 1975.

Shortliffe, E. H. A Rule Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection. Ph.D. Dissertation, Stanford Artificial Intelligence Laboratory, Memo AIM-251, 1974.

Schubert, L. K. Extending the Expression Power of Semantic Networks. Artificial Intelligence, 7, (1976), 163-198.

Waterman, D. A. Adaptive Production Systems, Working paper #285, Department of Psychology, Carnegie-Melon University, Pittsburgh, 1974.