VARIABLE-VALUED LOGIC AND ITS APPLICATIONS TO PATTERN RECOGNITION AND MACHINE LEARNING

by

R. S. Michalski

Chapter in the monograph, Computer Science and Multiple-Valued Logic Theory and Applications, D. C. Rine (Ed.), pp. 506-534, North-Holland Publishing Co., 1975.

Reprinted from:

computer science and multiple-valued logic

theory and applications

edited by

david c. rine

chapter 18

variable-valued logic and its applications to pattern recognition and machine learning

ryszard s. michalski

"... the purpose of science is to invent workable descriptions of the universe..."

S. S. Stevens, Psychology:
The Propaedeutic Science,
Phil. Sci. 3, 1936

1. Introduction

Most of the work done until now on multiple-valued logic in relation to computer science is oriented toward providing a theoretical and practical basis for constructing non-binary computer and switching systems. Very few attempts been made to investigate other applications of multiple-valued[5] logic. Among these few, there were attempts to apply multiple-valued logic to, for example, programming languages[24] or neural modelling[13]. This chapter presents some results on the application of an extended form of multi-valued logic called variable-valued logic to pattern recognition and artificial intelligence. The results described here come from various papers the author has published on this topic for the last several years. We will delineate here the concept of a variable-valued logic (VL) system, describe a particular VL system called VL_1, characterize briefly its properties and then illustrate by examples its application to pattern recognition and machine learning.

2. Variable-valued logic: Background and motivation

The concept of a variable-valued logic system (VLS) was first presented in 1972 at the IFIP Working Conference on Graphic Languages in Vancouver, Canada[17]. This concept extends known multiple-valued logic systems (MLS) in two directions:

(1) It permits the propositions and variables in the propositions to take values from different domains, which can vary in the kind and number of elements and also in the structure relating the elements.

(2) It generalizes some of the traditionally used operators and adds new operators which are 'most ortogonal' to the previous ones (the word 'ortogonal'

loosely means here that the expressions which are very long when using conventional operators become much shorter when new operators are used).

Direction (1) comes from the observation that if we want to apply logic statements as descriptions of reality which are usable by machines, it is inadequate to assume that every statement and all variables in it have to assume some arbitrary number of values, with no regard to the meaning of the statement or situation in which it is used (in *k*-valued logic everything is assumed to be *k*-valued).

The concept of truth is a measure we use to characterize a relationship between descriptions or models of some reality and the reality itself. A degree to which the descriptions suffice as representations of that reality is taken as a 'degree of truth' of statements.

In many situations it is possible to clearly state that a description suffices or that it does not suffice as a representation of the given reality. In such cases we operate in a binary logic environment, which is the simplest possible environment.

Let us consider the statement 'While I am writing this paper I am sitting at my desk'. This statement will be considered 'true' by someone who will agree to call my position 'sitting', to call the gray flat-topped construction 'a desk' and to call my location with regard to this construction by 'at'. If he does not agree with all this, then he will say 'false'. But if he has only some doubts about some of the conditions, he may say 'true in some sense', just introducing a new third possibility, besides 'true' and 'false', to the set of 'truth values' (output domain) of the relationship between the statement and the reality it describes.

As another example, let us consider the statement, 'Tomorrow I will be sitting at my desk'. This statement describes some reality which does not exist at the present time; it will exist in the future. This reality is yet unknown; therefore, the statement cannot be characterized at the moment as 'true' or 'false'. It could be characterized as 'undefined'. Thus, 'undefined' can be treated as a value in the output domain of this statement. Other values in the domain could be, e.g., 'likely, unlikely'. One could also think of an example where there may be a need for more than three values in the output domain of a statement to describe adequately a given situation. In the cases just described we operate in a multi-valued logic environment.

Suppose now, that together with the previous statement 'While I am writing this paper, I am sitting at my desk,' it is also said that 'The desk is gray'. It may be that a viewer of the scene who would treat the first statement as 3-valued, would consider the second statement as definitely 2-valued, because he has a clear way of testing it. Consequently, in this situation it would be proper to treat the first statement as 3-valued and the second statement as 2-valued, that is, to operate in a 'variable-valued' environment. (Of course, one can treat both statements as 3-valued. But in this case there would be a value in the domain of the second statement which would be superfluous and without any meaning. This would not only be intellectually unsatisfying and computationally costly (especially where

many such variables are involved) but also could cause difficulties in implementing learning processes.)

As we see from the above examples, the output domains (domains of 'truth values') of different statements may be different and they depend on the meaning of the statements or on the situation in which they are used. The output domains of different statements may have different values, different numbers of elements and, also, different structures relating the elements. To see the last point, note that the value 'true in some sense' in the domain of 'truth values' for the first statement is clearly in between 'true' and 'false', thus this domain is a linearly ordered set. But the value 'undefined' in the domain {likely, unlikely, undefined}, (used as a domain of the statement 'Tomorrow I will be sitting at my desk') has no relation to other elements and, therefore, this domain is only a partially ordered set.

The above considerations justify direction (1) from the viewpoint of logic. They also show that the concept of truth can be considered a property of relationships between statements and reality. This property, as we have shown, can have different domains with different structures just as properties we use in describing physical objects. E.g., properties such as color, volume, blood type, sex, height, etc. also can have different values related in different ways. The latter observation shows a direct applicability of variable-valued logic as a tool for creating descriptions of objects.

The direction (2) stated before ('new operators') is motivated by an observation that all human information processing activities seem to be governed by a law which can be called a 'law of maximum simplicity' (or 'law of minimum complexity'). This law can be informally interpreted as meaning that all information processing activities have an overall tendency to achieve given information processing goals in the simplest way. This 'simplest way' may mean, in a particular situation, the least amount of computation, time or effort, etc. In the case of describing various situations, this law means that the descriptions which we are using or producing, will tend to involve only necessary and most important information for our purpose. For example, when we address a letter to someone living in a foreign country, we will put the name of the country as a part of the address, but we would instinctively drop the name of the country, if the person lives in our country, because it is 'unnecessary'. Simplicity and efficiency of descriptions of objects is a fundamental problem in areas such as machine learning, pattern recognition, artificial intelligence, etc. Therefore, the problem of minimality of logical expressions used as object descriptions becomes very important. It is known, however, that certain classes of functions, e.g., symmetric functions, may have very long logical expressions when only traditional operators $\wedge$, $\vee$, $\neg$ are used. Consequently, there is a need for introducing into logical systems new operators, which would reduce expressions for such classes of functions. Thus, contrary to the usual tendency in logic, we are not interested in minimizing the number of operators, but in adding a variety of new operators. For example, the symmetric selector introduced in the logic system VL_1 described in the next

section, makes expressions of symmetric functions very short compared to expressions without this operator.

An important manifestation of the law of 'maximum simplicity' is that people have a very visible tendency to vary and minimize the 'number of quantization units' of all concepts they use in descriptions. By the 'number of quantization units' of a concept we mean the number of distinct values a given concept can accept in a description. For example, quantization units of the concept 'color' can be, e.g., {dark, light} or {blue, yellow, red} or {lengths of reflected light in Å}. In the first case, there are only two quantization units, while in the third case, there may be millions.[1]

To exemplify the above tendency, let us observe that if it is quite sufficient to distinguish someone from a group of people by saying 'this tall man', then one would more than likely say just this statement, rather than saying 'this fairly tall man' or this '6′1″ tall man'. Note that these statements imply successively larger number of quantization units of the concept 'height of a person'. Note also that the quantization units do not have to be of the same 'size', e.g., a basketball coach may quantize players' height as short, medium, 6′1″, 6′2″, 6′3″, ...

The tendency for minimization of the domains of concepts is one of the most important tools to combat the 'combinatorial explosion' inherent in any complex information processing activities, such as, e.g., creation of problem-oriented descriptions of objects or situations.

To conclude these general remarks, it should be noted that one of the major goals in developing VL systems is to create an adequate and efficient formal system for determining machine usable descriptions of objects or situations, for transforming such descriptions, for performing deductive and, in particular, inductive inferences about any objects or classes of objects, etc. This feature distinguishes VL systems from other non-classical logic systems such as Post algebra [27] or fuzzy logic [10, 28], whose basic motivation is different. The work in this direction, whose beginning [14, 15] goes back to 1969, belongs to a significant degree to the area which can now be called 'logical pattern recognition'. And as such, it is related to other works in this area, for example, work by Bruner [3], Hunt [8], Kochen [9], Banerji [1], Zhuravlev [29], Towster [26], Stoffel [25] and others.

3. Variable-valued logic system VL_1

3.1. Definition. Formally, a *variable-valued logic system* is defined as an ordered quintuple:

$$(X, Y, S, R_F, R_I) \tag{1}$$

where X is a finite non-empty (f.n.) set of *input* or *independent* variables whose

[1] There are estimated more than 7 million colors discriminable by a human eye.

domains D_i, $i = 1, 2, 3, \ldots$, are any non-empty sets; Y is a f.n. set of *output* or *dependent* variables, whose domains ${}^jD$, $j = 1, 2, \ldots$, are any non-empty linearly ordered sets; S is a f.n. set of *connecting symbols*, which denote operations on input and output variables; R_F is a f.n. set of *formation* or *syntactic* rules which define well-formed formulas (wffs) in a system (*VL formulas*) (a string of elements from X, D_i, Y, ${}^jD$ and S is a wff, if and only if it can be derived from a finite number of applications of the formation rules); and R_I is a f.n. set of *interpretation* or *semantic rules* which give an interpretation to VL formulas. Namely, they specify how, for any string of values taken by independent variables, to compute from the formula values of dependent variables.

In this paper we will discuss only one specific VL system called VL_1.

3.2. System VL_1: General outline. The VL_1 system is a simple VL system which uses conventional operations of multiple-valued logic such as $\vee$ (max), $\wedge$ (min), $\neg$ (inverse), and some new operators such as: a selector (which is a generalization of various known unary operators, e.g., the disjoint Post operators C_i, monotonic operators D_1)[5], arithmetic addition (which is used in a restricted context, namely to express symmetric properties of functions), exception and separation (a generalization of symmetric difference). One of the important features of VL_1 is that it constructs multi-valued formulas with multi-valued variables by using selectors which are two-valued functions. This has 2 significant advantages: (1) it makes the VL_1 formulas easy to interpret and comprehend and (2) it avoids enormous computational complexities arising in the synthesis and minimization of formulas when operators are allowed to be applied directly to multi-valued variables rather than to two-valued selectors.

Before giving a definition of VL_1 we will informally delinate some of its basic concepts.

A VL_1 formula is interpreted as an expression of a function which maps the cartesian product of the domains D_i, $i = 1, 2, 3, \ldots$ of input variables into the domain D of an output variable.

The formula is constructed by applying operators max ($\vee$), min ($\wedge$), inverse ($\neg$), exception (\) and separation (|) to selectors and elements of D. A simple form of a selector is a unary condition which tests whether a value of a given variable belongs to a certain subset of the domain of this variable. For example:

$$[x_3 = 2, 3, 5]$$

is a selector which is satisfied if a value of x_3 is 2, 3, or 5, otherwise is not satisfied. A satisfied selector accepts the highest and an unsatisfied the lowest value in the output domain D. Similarly,

$$[x_2 \leqslant 3]$$

is satisfied when value of x_2 is smaller or equal than 3, and

$$[x_1 \neq 2{:}4]$$

is satisfied when a value of x_i is not between 2 and 4, inclusively. A more complex form of a selector permits one to have an analogous condition on an arithmetic sum of variables or their inverses, or on a VL_1 formula itself.

3.3. Definition of VL_1. VL_1 is a variable-valued logic system (X, Y, S, R_F, R_I) where:

X is a f.n. set of input variables whose domains are finite non-empty (f.n.) sets (the elements of the sets are called *input semantic units*). To be specific, we will assume, without loss of generality, that variables are

$$x_1\ x_2\ x_3 \ldots x_n \tag{2}$$

and their domains $D(x_i)$ or D_i are sets of non-negative integers with $d_1, d_2, \ldots, d_n$ elements, respectively:

$$D_i = \{0, 1, 2, \ldots, d_i - 1\}, \quad i = 1, 2, \ldots, n. \tag{3}$$

Y is a set consisting of one output variable whose domain is a well-ordered f.n. set (the elements of the set are called *output semantic units*). To be specific, we will assume, without loss of generality, that the variable is y and its domain D is a set of non-negative integers:

$$D = \{0, 1, 2, \ldots, d - 1\}. \tag{4}$$

A possible way of interpreting elements of D is, e.g., to interpret them as truth-values ('degrees of truth') or 'degrees of preciseness' of statements describing relations among values of variables x_i.

S is the set of 16 *connecting symbols*:

$$= \neq \geqslant \leqslant | \vee \setminus \wedge + \neg ( ) [ ] , :$$

R_F is a set of the formation rules which define wff formulas in the system (VL_1 formulas):

(1) An element of D standing alone is a wff.

(2) A form $[L \# R]$ is a wff, iff:

$$\# \in \{=, \neq, \leqslant, \geqslant\}, \quad L \in \{x, V_1\}, \quad R \in \{c, V_2\}.$$

x is a literal $\tilde{x}_i$ or a sequence of *literals* $\tilde{x}_i$, $i \in I$, $I \subseteq \{1, 2, \ldots, n\}$, where $\tilde{x}_i$ is a variable x_i or a form $\neg x_i$ (written also $\bar{x}_i$, separated by the symbol + . x can also be a name of such a sequence.

c is a sequence of different non-negative integers in ascending order and separated by ',' or ':' or a name of such a sequence.

V_1, V_2 are wffs or names of wffs.

The form $[L \# R]$ is called a *selector*. L is called the *left part* or *referee*, and R is called the *right part* or *reference* of the selector. If L is x_i, then L is called a *simple referee*. If R is c, then R is called a *simple reference*. A selector with a

simple referee and simple reference, i.e. a form $[x_i \# c]$, is called a *simple selector.*

A simple reference, c, is said to be in an *extended form*, if it contains no ':'. If in an extended form, every maximal under inclusion[2] sequence of consecutive integers of length at least three is replaced by a form $c_1{:}c_2$, where c_1 is the first and c_2 the last element of the sequence, then the obtained reference is in a *compressed form.* If a referee is not simple, but reference is simple, then selector is called a *symmetric selector.*

(3) Forms (V), $\neg V$, $V_1 \wedge V_2$ (also written $V_1 V_2$), $V_1 \setminus V_2$, $V_1 \vee V_2$ and $V_1 | V_2$, where V, V_1 and V_2 are wffs or names of wffs, are wffs.

(3.1) $\neg V$ is called the *inverse* of V;
(3.2) $V_1 V_2$ is called the *product* (or *conjunction*) of V_1 and V_2;
(3.3) $V_1 \setminus V_2$ is called the *exception* V_2 *from* V_1 (or V_1 *except for* V_2);
(3.4) $V_1 \vee V_2$ is called the *sum* (or *disjunction*) of V_1 and V_2;
(3.5) $V_1 | V_2$ is called the *separation* of V_1 and V_2.

R_1 is a set of interpretation rules which assign to any VL_1 formula V a *value* $v(V) \in D$, depending on values of $x_1, \ldots, x_n$:

(1) The value $v(c)$ of an element $c \in D$, is c, which is denoted:

$$v(c) = c \tag{5}$$

(2) $$v([L \# R]) = \begin{cases} d-1, & \text{if } v(L) \# v(R) \\ 0, & \text{otherwise} \end{cases},$$

where $v(L)$, $L \in \{x, V_1\}$, is $v(V_1)$, i.e. value of wff V_1, if L is V_1, otherwise is $v(x)$, i.e. value of the sequence x. Assuming that x is a sequence of literals $\tilde{x}_i$, $i \in I$, separated by '+', $v(x)$ is defined as:

$$v(x) = \sum_{i \in I} v(\tilde{x}_i), \tag{6a}$$

where Σ denotes arithmetic sum

$$v(\tilde{x}_i) = \begin{cases} \dot{x}_i, & \text{if } \tilde{x}_i \text{ is } x_i, \\ d_i - 1 - \dot{x}_i, & \text{if } \tilde{x}_i \text{ is } \bar{x}_i. \end{cases} \tag{6b}$$

$\dot{x}_i$ denotes a value of variable x_i, $\dot{x}_i \in D_i$, $I \subseteq \{1, 2, 3, \ldots, n\}$.

Example: If values of variables x_1, x_2, x_3, x_4 are, respectively: 2, 0, 3, 4, and maximal elements in their domains: 4, 4, 5, 5, then:

$$v(x_2 + \bar{x}_3 + x_4) = 0 + (5 - 3) + 4 = 6.$$

$v(R)$, $R \in \{c, V_2\}$, is the sequence c, if R is c; otherwise $v(V_2)$. $v(L) \# v(R)$, $\# \in \{=, \neq, \leqslant, \geqslant\}$, is true if $v(L)$ is in relation $\#$ with $v(R)$. If R is c, then

[2] I.e., a sequence of consecutive integers which is not a part of another such sequence.

$v(L) = \boldsymbol{c}$ ($v(L) \neq \boldsymbol{c}|$) is true if $v(L)$ is (is not) one of the integers in $\boldsymbol{c}$, or is (is not) between any pair of integers in $\boldsymbol{c}$ separated by ':' and $v(L) \leqslant \boldsymbol{c}(L) \geqslant \boldsymbol{c}$) is true if $v(L)$ is smaller than or equal to (greater than or equal to) every integer in $\boldsymbol{c}$ (in normal use of these relations $\boldsymbol{c}$ will consist of just one element).

If $v(L) \# v(R)$, then the selector $[L \# R]$ is said to be *satisfied.*

(3)
$$v((V)) = v(V) \tag{7}$$
$$v(\neg V) = d - 1 - v(V) \tag{8}$$
$$v(V_1 \wedge V_2) = v(V_1 V_2) = \min\{v(V_1), v(V_2)\} \tag{9}$$
$$v(V_1 \setminus V_2) = \begin{cases} v(V_1), & \text{if } v(V_2) = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$
$$v(V_1 \vee V_2) = \max\{v(V_1), v(V_2)\}. \tag{11}$$
$$v(V_1 | V_2) = \begin{cases} v(V_1) & \text{if } v(V_2) = 0, \\ v(V_2) & \text{if } v(V_1) = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Parentheses have the usual meaning, i.e. they denote that a part of a formula is to be evaluated as a whole. Operations are order from one with the highest to one with the lowest priority as follows: $\neg \vee \setminus \wedge |$.

If there is more than one operation \, the priority goes from the most left one to the most right one unless it is changed by ( ).

In general, the elements of domain D_i, $i = 1, 2, \ldots, n$, and/or domain D may not be integers. In this case it is assumed that D_i (D) are linearly ordered according to some desired semantic or syntactic property (e.g., alphabetically) and then their elements are assigned *positions* 0, 1, 2, ... $d_i(d)$. In eq. 6b, $\dot{x}_i$ is then interpreted as the position of a value of variable x_i rather than the value itself. And as the *value* of a formula V is now taken the element of D whose position is $v(V)$ defined as above, rather than $v(V)$ itself.

3.4. Discussion of VL_t system

3.4.1. Examples of valid and invalid VL_t *formulas.* A string of symbols from sets X, D and S, specified in the definition of VL_t, is a VL_t formula if and only if it can be constructed by a finite number of applications of the formation rules R_F. Below are given some examples of VL_t formulas, and, for comparison, some strings which are not VL_t formulas.

VL_t formulas:

$$3[x_t = 0, 3{:}5][x_3 \neq 2, 4] \vee 2[x_2 \neq 3] \vee 1[x_4 \geqslant 2] \tag{13}$$
$$([x_2 = 1{:}3] \vee [x_3 \neq 4, 6])(1 \vee [x_1 = 0, 2, 4] \vee [x_3 = 3]) \tag{14}$$
$$\neg(2[x_1 = 3] \vee 1)[x_3 \neq 2] | 2([x_1 + x_3 = 0, 5]) \setminus [x_2 = 2] \vee 1[x_2 \neq 0] \tag{15}$$

Not VL_1 formulas:

$$([x_1 = 0, 2{:}5] = 3)[x_1 \neq 1] \vee 2[[[x_1 = 0] \vee 1 = [x = 1]]] = 2] \tag{16}$$

$$3[x_3 \neq 0{:}3] \vee \neg 2[[x_3 = 0, 1, 3, 5] = [x_3 = 2]] \vee x_3 \tag{17}$$

$$[2[x_1 + x_4 = 0] \vee 1 = x_1] \vee 1[2, 3 = 2[x_2 \neq 2]] | 3[x_2 + x_4 \leqslant 3] \tag{18}$$

String (16) is not a VL_1 formula because the form $[x_1 = 0, 2{:}5] = 3$ does not satisfy the definition of a selector (should be enclosed in [ ]). (17) is not a VL_1 formula because a variable standing alone is not a wff. (18) is not a VL_1 formula because x_1 is on the right-hand side of =, and also because the string '2, 3' is on the left-hand side of =.

3.4.2. Event space. The interpretation rules R_1 assign to any VL_1 formula a value (an element of a set D, depending on the values of $x_1, x_2, \ldots, x_n$) elements of sets $D_1, D_2, \ldots, D_n$. Thus, the interpretation rules interpret VL_1 formulas as expressions of a function

$$f: D_1 \times D_2 \times \cdots \times D_n \to D \tag{19}$$

where $\times$ denotes cartesian product and $\to$ denotes 'mapping into'. The set $D_1 \times D_2 \times \cdots \times D_n$, $D_i = \{0, 1, \ldots, d_i - 1\}$, $i = 1, 2, \ldots, n$, includes all possible sequences of values of input variables and is called the *universe of events* or the *event space.* The event space is denoted by $E(d_1, d_2, \ldots, d_n)$, where[3] $d_i = c(D_i)$, or, briefly, by E. The elements of an event space E, vectors $(\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n)$, where $\dot{x}_i$ is a value of the variable x_i, $\dot{x}_i \in D_i$ are called *events* and denoted by e^j, $j = 0, 1, 2, \ldots, d - 1$, where $d = c(E) = d_1 \cdot d_2 \ldots d_n$. Thus, we can write:

$$E = E(d_1, d_2, \ldots, d_n) = \{(\dot{x}_i, \dot{x}_2, \ldots, \dot{x}_n) | \dot{x}_i \in D_i, i = 1, 2, \ldots, n\} = \{e^j\}_{j=0}^{d-1} \tag{20}$$

We will assume that values of the index j are given by a function:

$$\gamma: E \to \{0, 1, \ldots, d - 1\} \tag{21}$$

specified by the expression:

$$j = \gamma(e) = x_n + \sum_{k=n-1}^{1} x_k \prod_{i=n}^{k+1} d_i \tag{22}$$

$\gamma(e)$ is called the *number of the event e.* For example, the number of the event $e = (3, 2, 1, 2)$ in the space E(4, 4, 2, 3) is:

$$\gamma(e) = 2 + 1 \cdot 3 + 2 \cdot 3 \cdot 2 + 3 \cdot 3 \cdot 2 \cdot 4 = 89.$$

It can be verified that each event e of a given event space has the unique number $\gamma(e)$, and, also, that given $\gamma(e)$ and $E(d_1, d_2, \ldots, d_n)$ one can retrieve[4] the event e.

[3] $c(S)$, where S is a set, denotes the cardinality of S.

[4] The retrieval of an event $e = (\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n)$ can be done by arithmetically dividing $\gamma(e)$ first by d_n, to obtain $\dot{x}_n$ as the remainder of this division, then by dividing the result by d_{n-1}, to obtain $\dot{x}_{n-1}$ as the remainder, and so on, until obtaining $\dot{x}_1$.

3.4.3. Examples of VL_1 *formulas' interpretation.* Recall the formula (13): $3[x_1 = 0, 3{:}5][x_3 \neq 2, 4] \vee 2[x_2 \neq 3] \vee 1[x_4 \geqslant 2]$. This formula can be interpreted as an expression of a function:

$$f\colon \mathrm{E}(6, 4, 5, 5) \to \{0, 1, 2, 3\} \tag{23}$$

assuming that $c(D_1) = 6$, $c(D_2) = 4$, $c(D_3) = 5$, $c(D_4) = 5$. The formula is assigned value 3 (briefly, *has value* 3), if selectors $[x_1 = 0, 3{:}5]$ and $[x_3 \neq 2, 4]$ are satisfied, i.e., if x_i accepts value 0, 3, 4 or 5 and x_3 value 0, 1 or 3. (Thus, the value 3 of the formula does not depend on values of x_2 and x_4.) The formula has value 2 if the previous condition does not hold and the selector $[x_2 \neq 3]$ is satisfied (recall that $v(V_1 \vee V_2) = \max\{v(V_1), v(V_2)\}$). The formula has value 1, if both of the previous conditions do not hold and the selector $[x_4 \geqslant 2]$ is satisfied. If none of the above conditions hold, the formula has value 0.

Consider a formula:

$$2[x_2 + x_3 + x_4 = 2, 3] \setminus [x_1 = 0] \tag{24}$$

The formula has value 2 for all events in which values of variables x_2, x_3 and x_4 sum up to 2 or 3, except for such events with this property in which the value of variable x_1 is 0. For all the other events the formula has value 0. Note that the function expressed by formula (24) is *symmetric* with regard to variables $\{x_2, x_3, x_4\}$, i.e. these variables can be exchanged one for another without changing value of the function.

3.4.4. VL *functions.* In describing objects (or processes) we usually deal with functions which may have an unspecified value for certain events, that is with functions:

$$f\colon \mathrm{E} \to D \cup \{*\} \tag{25}$$

where $*$ represents an unspecified value.

An incompletely specified function f can be considered as being equivalent to a set $\{f_i\}$ of completely specified functions f_i, each determined by a certain assignment of specified values (i.e. values from D) to events e for which $f(e) = *$. (We will call such events $*$-*events*.) If $*$-events can never occur (because of semantic constraints), or if they do occur, the value of f is not relevant, a formula VL_1 which expresses any function f_i can be accepted as an expression of f. If, however, we have to preserve the information which events are $*$-events, then value $*$ can be renamed into an additional element in D.

Functions of the type (25) will be called *variable-valued logic functions or* VL *functions.*

3.4.5. Multiple-output VL *functions.* As was previously mentioned, a VL_1 formula expresses a function

$$f\colon \mathrm{E} \to D$$

In practical applications we may be interested in expressing not just one function f

but a family of functions with the same domain E: (26)

$$f: \mathrm{E} \rightarrow \boldsymbol{D} \tag{27}$$

where

$$f = ({}^1f, {}^2f, \ldots, {}^mf),\ {}^kf: \mathrm{E} \rightarrow {}^kD,\ k = 1, 2, \ldots, m,$$
$${}^kD = \{0, 1, 2, \ldots, {}^kd - 1\},$$
$$\boldsymbol{D} = {}^1D \times {}^2D \times \cdots \times {}^mD.$$

Thus

$$f: D_1 \times D_2 \times \cdots \times D_m \rightarrow {}^1D \times {}^2D \times \cdots \times {}^mD \tag{28}$$

A function f is called a *multiple-output* VL function. It can be expressed by a single VL_1 expression by extending the set X with an additional variable y whose domain is $\{1, 2, \ldots, m\}$ and assuming that the output domain

$$D = \max ({}^1D, {}^2D, \ldots, {}^mD).$$

The role of the variable y is expressed by an additional interpretation rule:

(4) $v(V[y \# c])$ is interpreted as follows.

The value $v(V)$ is given to functions $\{{}^yf \mid y \# c\}$. For example, if V_1, V_2 and V_3 are VL_1 formulas which do not include the variable y, then

$$V_1[y = 1, 2, 3] \vee V_2[y = 3, 4] \vee V_3[y = 1, 3] \tag{29}$$

is interpreted as an expression of a function $f = ({}^1f, {}^2f, {}^3f, {}^4f)$, where expressions for function ${}^kf_1 k = 1, 2, 3, 4$ are:

$${}^1f: V_1 \vee V_3, \quad {}^2f: V_1, \quad {}^3f: V_1 \vee V_2 \vee V_3, \quad {}^4f: V_2 \tag{30}$$

The selector $[y = c]$ is called a *function selector.* A VL_1 formula which includes function selectors is called a *multiple-output* VL_1 *formula.* We extend a function f (27) to an incompletely specified function by assuming that:

$$f: \mathrm{E} \rightarrow ({}^1D \cup \{*\}) \times ({}^2D \cup \{*\}) \times \cdots \times ({}^mD \cup \{*\}) \tag{31}$$

3.4.6. Special classes of VL_1 *formulas.* We will define some special classes of VL_1 formulas (disjunctive and conjunctive simple VL_1 formulas, cyclic formulas and interval formulas) which are of special interest for applications, because of the simplicity of their interpretation, evaluation and synthesis. We will start with defining some auxiliary concepts.

Definition 1. A *component* of a VL_1 formula is defined as a selector or a constant from D, or a VL_1 formula in parentheses, or an inverse of a VL_1 formula.

Definition 2. A product of components is called a *term.* A term in which each of the components is either a simple selector or a constant is called a *simple term.*

Definition 3. A sequence of terms separated by \ is called a *phrase.*

Definition 4. A disjunction of phrases is called a *clause.* A clause in which each of the phrases is either a simple selector or a constant from D is called a *simple clause.*

Definition 5. A sequence of clauses separated by | is called a *combined* VL_1 formula.

Definition 6. A simple selector $[x_i \# c]$, $\# \in \{=, \neq\}$, whose reference c is $c_1{:}c_2$ or c, is called a *cyclic selector.* A cyclic selector in which '#' is just '=' is called an *interval selector.*

Interval and cyclic selectors are the simplest among selectors for evaluation. To evaluate an interval selector $[x_i = c_1{:}c_2]$ one needs only to check whether the value of x_i is between c_1 and c_2, i.e. within an interval; and to evaluate a cyclic selector $[x_i \# c_1{:}c_2]$ to check whether the value of x_i is in- or outside the interval.

Definition 7. A VL_1 formula which is a disjunction of simple terms is called a *disjunctive simple* (or *normal*[5]) VL_1 formula and denoted DVL_1. (For example, (13) is a DVL_1.)

Definition 8. A VL_1 formula which is a conjunction of simple clauses is called a *conjunctive simple* (or *normal*) VL_1 formula and denoted CVL_1. (For example, (14) is a CVL_1.)

Definition 9. A VL_1 formula which is a DVL_1 or CVL_1 is called a *simple* (or *normal*) VL_1 formula.

Theorem 1. *For any* VL *function there exists at least one* DVL_1 *and at least one* CVL_1 *formula, which are expressions of this function.*

Proof. We will give a construction algorithm. Let a given VL function f be defined by sets of events

$$F^k = \{e \,|\, f(e) = k\}, \quad k = 0, 1, 2, \ldots, d-1$$

Express all events of each F^k as a conjunction of selectors and the constant k. For example, event $(a_1, a_2, \ldots, a_n)$ of F^k would be expressed as

$$k[x_1 = a_1][x_2 = a_2] \ldots [x_n = a_n]$$

[5]The adjective 'normal' is given as an alternative for the sake of tradition as a disjunctive simple VL_1 formula reduces in the binary logic case (when $d_1 = d_2 = \cdots = d_n = d = 2$) or in the k-valued-logic case (when $d_1 = d_2 = \cdots = d_n = d = k$) into a form which directly corresponds to a form usually termed as disjunctive normal form.

Disjunction of all such obtained conjunctions (terms) is a DVL expression of f.

To construct a CVL_1, represent the inverse $\neg f$ (defined as $d-f-1$) by sets

$$F^{\bar{k}} = \{e \mid \neg f(e) = \bar{k}\}, \quad \bar{k} = 0, 1, 2, \ldots, d-1$$

Construct a DVL_1 expression for $\neg f$ in the way described above and then apply to it the De Morgan's law (40), to obtain a CVL_1 expression of f. □

3.4.7. Specification of formation rules as rewriting rules of a context-free grammar. Below are given rewriting rules of a contex-free grammar (together with a corresponding terminology), which are an alternative way of expressing formation rules R_F of the VL_1. They are equivalent to R_F with the exception for sequences x and c. In R_F, x and c were defined as sequences of *different* elements while here they can include identical elements (x and c, as defined in R_F, cannot be expressed by a contex-free rewriting rules). A bar '|' and '::=' are used as metalanguage symbols for describing rewriting rules.

Formula	$V ::= \mathscr{L} \mid V \mid \mathscr{L},$
Clause	$\mathscr{L} ::= \mathscr{P} \mid \mathscr{L} \vee \mathscr{P},$
Phrase	$\mathscr{P} ::= T \mid \mathscr{P} \setminus T,$
Term	$T ::= \mathscr{C} \mid T \wedge \mathscr{C} \mid T\mathscr{C},$
Component	$\mathscr{C} ::= c \mid S \mid \neg V \mid (V),$
Selector	$S ::= [L \# R],$
Left part (referee)	$L ::= x \mid V,$
Right part (reference)	$R ::= c \mid V,$
Selector relation	$\# ::= = \mid \neq \mid \geqslant \mid \leqslant,$
Output constants	$c ::= 0 \mid 1 \mid 2 \mid \ldots \mid d-1,$
Arithmetic sum of literals	$x ::= \tilde{x}_i \mid x + \tilde{x}_i,$
Literals	$\tilde{x}_i ::= x_i \mid \neg x_i,$
Input variables	$x_i ::= x_1 \mid x_2 \mid \ldots \mid x_n,$
Simple reference	$c ::= c_i \mid c, c_i \mid c : c_i,$
Non-negative integers	$c_i ::= 0_i \mid 1 \mid 2 \mid \ldots.$

3.4.8. Specification and equivalence of VL_1 formulas. When we say 'V is a VL_1 formula', by V we mean a *name* of a VL_1 formula. If we want to specify the formula whose name is V, we write: '$V =$' then write the formula. For example:

$$V = 3[x_1 = 3, 5] \vee 2[x_2 \neq 0] \vee 1 \tag{32}$$

Suppose that V_1 and V_2 are two VL_1 formulas which depend on the same set of variables $X = \{x_1, x_2, \ldots, x_n\}$ and take values from the same set D. Accepting notation $V(X, D)$ for the set of all VL_1 formulas which depend on X (or a subset of x) and take values from D (or a subset of D), we can write: $V_1, V_2 \in V(X, D)$. If for each event $e \in E(d_1, d_2, \ldots, d_n)$, $d_i = c(D_i)$, $i = 1, 2, \ldots, n$ and D_i–the domain of x_i:

$$v(V_1) = v(V_2) \tag{33}$$

then formulas V_1 and V_2 are called *equivalent with regard to interpretation*, or, briefly, *equivalent*, and we write:

$$V_1 \equiv V_2 \tag{34}$$

The connector $\equiv$ will also be used, with the meaning extended in the obvious way, when V_1 and/or V_2 in (40) are substituted by actual VL_1 formulas or by VL_1 formulas in which some parts are represented by names.

All operations on V_1 and/or V_2 which preserve relation (34) are called *equivalence-preserving operations.*

3.4.9. Examples of equivalence-preserving operations on VL_1 *formulas.* Let $V \in V(X, D)$.

$$V(d-1) \equiv V, \qquad V \vee 0 \equiv V \qquad \text{(identity elements)}, \tag{35}$$

$$V0 \equiv 0, \qquad V \vee (d-1) \equiv d-1 \qquad \text{(zero elements)}, \tag{36}$$

$$\neg(c) \equiv \bar{c} \qquad \text{where } v(\bar{c}) = d-c-1 \tag{37}$$

If $\not\#$ is $=$ or $\neq$, then

$$\neg[x_i \not\# c] = [x_i \bar{\not\#} c], \quad \text{where } \bar{\not\#} \text{ is } \begin{cases} \neq & \text{if } \not\# \text{ is } = \\ = & \text{if } \not\# \text{ is } \neq \end{cases}$$

For example, $\neg([x_2 = 3{:}5]) \equiv [x_2 \neq 3{:}5]$.

$$\neg[x_i \geqslant c] \equiv [x_i \leqslant c^-], \quad \text{where } c^- = c-1 \ (c > 0).$$

$$\neg[x_i \leqslant c] \equiv [x_i \geqslant c^+], \quad \text{where } c^+ = c+1 \ (c < d_i - 1).$$

Let V_1, V_2 and V_3 be elements of $V(X, D)$.

$$V_1(V_1 \vee V_2) \equiv V_1 \qquad V_1 \vee V_1 V_2 \equiv V_1 \qquad \text{(Absorption Laws)} \tag{38}$$

$$V_1(V_2 \vee V_3) \equiv V_1 V_2 \vee V_1 V_3 \qquad V_1 \vee V_2 V_3 \equiv (V_1 \vee V_2)(V_1 \vee V_3) \qquad \text{(Distributive Laws)} \tag{39}$$

Let $\{V_j\}_{j \in J}$ be a subset of $V(X, D)$.

$$\neg\left(\bigvee_{j \in J} V_j\right) \equiv \bigwedge_{j \in J} \neg(V_j) \tag{40}$$

$$\neg\left(\bigwedge_{j \in J} V_j\right) \equiv \bigvee_{j \in J} \neg(V_j) \tag{41}$$

De Morgan's Laws[6]

Examples: $\neg(2[x_1 = 1{:}3][x_3 \neq 2]) \equiv \bar{2} \vee [x_1 \neq 1{:}3] \vee [x_3 = 2]$ where $v(\bar{2}) = d-2-1$

$\neg(3[x_2 = 2, 4] \vee 1[x_2 = 0][x_3 \neq 1{:}3]) = (\bar{3} \vee [x_2 \neq 2, 4])(1 \vee [x_2 \neq 0] \vee [x_3 = 1{:}3])$

Let $c_1 \cup c_2$ (c_1 *merged with* c_2) and $c_1 \cap c_2$ (c_1 *intersected with* c_2) denote

[6] $\bigwedge$ and $\bigvee$ mean the conjunction and the disjunction of formulas, respectively.

compressed references obtained by set-theoretical summing and intersecting, respectively, the set of elements in extended forms of c_1 and c_2, then ordering the results by $<$ and finally transforming them into compressed forms. For example, if c_1 is 2, 4:6, 8 and c_2 is 0, 3, 5:7, then $c_1 \cup c_2$ is 0, 2:8 and $c_1 \cap c_2$ is 5, 6.

The following rules, called *merging rules*, describe when and how two terms of a VL_1 formula can be merged into one. Namely, if each of two terms T_1 and T_2 can be represented as a product of a term T with a simple selector involving the same variable:

$$T_1 = T[x_i \# c_1] \text{ and } T_2 = T[x_i \# c_2], \tag{42}$$

then in the case when '$\#$' is '$=$' or '$\geqslant$' or '$\leqslant$' in both T_1 and T_2:

$$T[x_i \# c_1] \vee T[x_i \# c_2] \equiv T[x_i \# c_1 \cup c_2] \tag{43}$$

(where $\#$ is the same relation in all three selectors), and when '$\#$' is '$\neq$' in both T_1 and T_2:

$$T[x_i \neq c_1] \vee T[x_i \neq c_2] \equiv T[x_i \neq c_1 \cap c_2]. \tag{44}$$

Any two terms which can be expressed in the form (42) are called *adjacent terms*. (They can always be merged into one. If '$\#$' is not the same in both selectors, then before applying (43) or (44), one of the selectors should be appropriately modified.)

If $c_1 \cup c_2$ is $0{:}d_i - 1$ (i.e., its extended form includes every element of D_i), or if $c_1 \cap c_2$ is empty, then, respectively: $[x_i = c_1 \cup c_2] \equiv d - 1$, $[x_i \neq c_1 \cap c_2] \equiv d - 1$, and according to (35), formulas (43) and (44) become:

$$T[x_i = c_1] \vee T[x_i = c_2] \equiv T, \tag{45}$$

$$T[x_i \neq c_1] \vee T[x_i \neq c_2] \equiv T. \tag{46}$$

Rules (51) and (52) are called *simplification rules*.

In applications, the following observation may be useful: any simple selector can be expressed as a product of cyclic selectors, or a sum of interval selectors. (To see this, consider, e.g., the selector $[x_i = 0, 2{:}4, 6]$. It can be expressed as a product $[x_i = 0{:}6][x_i \neq 1][x_2 \neq 5]$ or as a sum $[x_i = 0] \vee [x_i = 2{:}4] \vee [x_i = 6]$.)

With regard to operations $\setminus$ and $|$ it follows from (10) and (12) that:

$$V_1 \setminus V_2 \equiv V_1[V_2 = 0],$$

$$V_1 | V_2 \equiv V_1[V_2 = 0] \vee V_2[V_1 = 0].$$

4. Optimal VL_1 formulas

4.1. A planar geometrical representation of VL functions and VL_1 formulas. The merging and simplification rules (43–46) are examples of rules which allow us to modify (simplify) a given VL_1 formula without changing its interpretation (i.e., preserving the equivalence). There can be, in general, many different VL_1

formulas expressing the same VL function. Thus, a problem arises of how, for a given VL function, to construct the simplest (in some specified sense) VL_1 formula.

To illustrate this problem graphically, we will use a Generalized Logical Diagram (GLD) described in [16]. The GLD is a planar geometrical model of an event space $E(d_1, d_2, \ldots, d_n)$, and provides a convenient representation of VL functions and VL_1 formulas. For example, Fig. 1, shows a GLD representation of a VL function:

$$f: E(4, 2, 3, 3) \rightarrow \{0, 1, 2, 3, *\}$$

Cells of the diagram correspond to events in E(4, 2, 3, 3). The correspondence is self-explanatory, e.g., cell 25 corresponds to event $e^{25} = (1, 0, 2, 1)$. Numbers on the top and the right of the diagram are the numbers of the events to which the cells correspond (note a lexicographical order of the numbers). Cells are marked by values of the function f applied to the events to which the cells correspond. Empty cells denote events for which f equals $*$. (For a formal definition of GLD and a discussion of its properties, consult [16].)

An example of the VL_1 formula expressing the function f is

$$\begin{aligned} &3[x_1 = 2][x_2 = 0][x_4 = 0, 2] \vee 3[x_1 = 2][x_2 = 1][x_3 = 1] \vee \\ &\quad \vee 3[x_1 = 0, 3][x_3 = 2][x_4 = 0, 2] \vee 2[x_1 = 0, 1][x_2 = 0][x_3 = 0, 2][x_4 = 0, 2] \vee \\ &\quad \vee 2[x_3 = 1][x_4 = 0, 1] \vee 2[x_1 = 3][x_2 = 1][x_3 = 0][x_4 = 1, 2] \vee \\ &\quad \vee 1[x_1 = 0][x_2 = 1][x_3 = 0][x_4 = 1, 2] \vee 1[x_1 = 1, 3][x_2 = 1][x_3 = 0, 1] \vee \\ &\quad \vee 1[x_2 = 1][x_3 = 1, 2][x_4 = 0, 1]. \end{aligned} \tag{47}$$

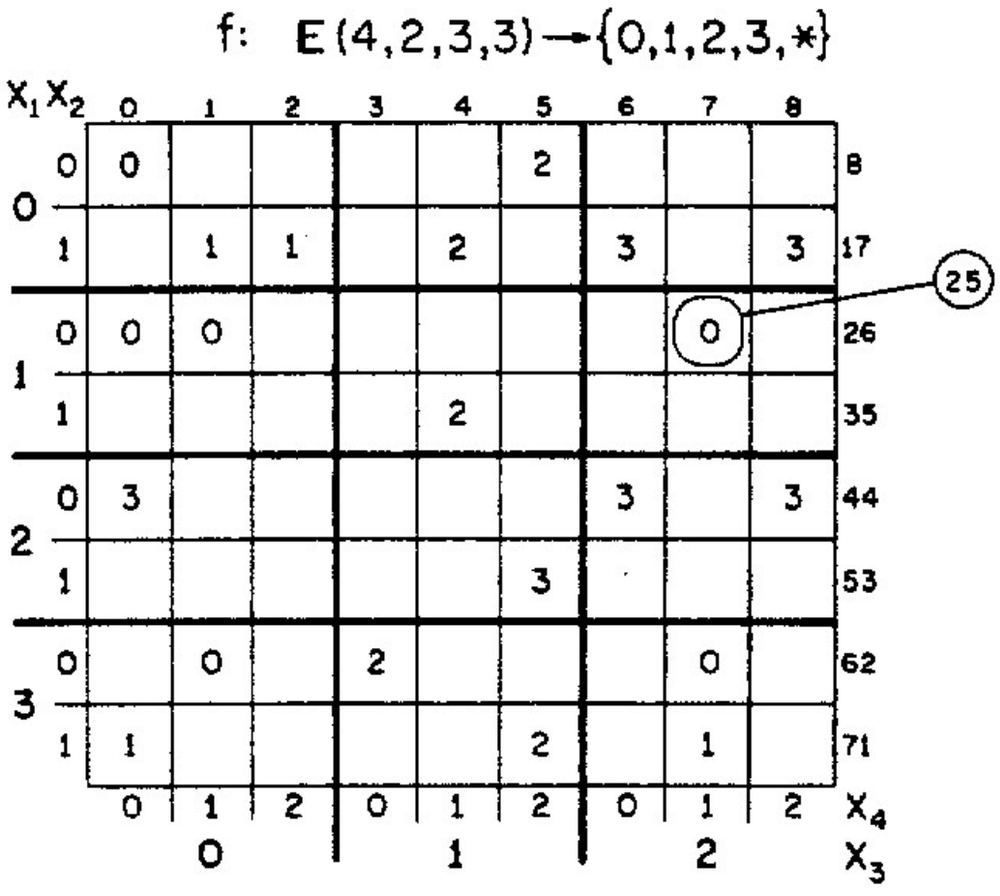


Fig. 1.

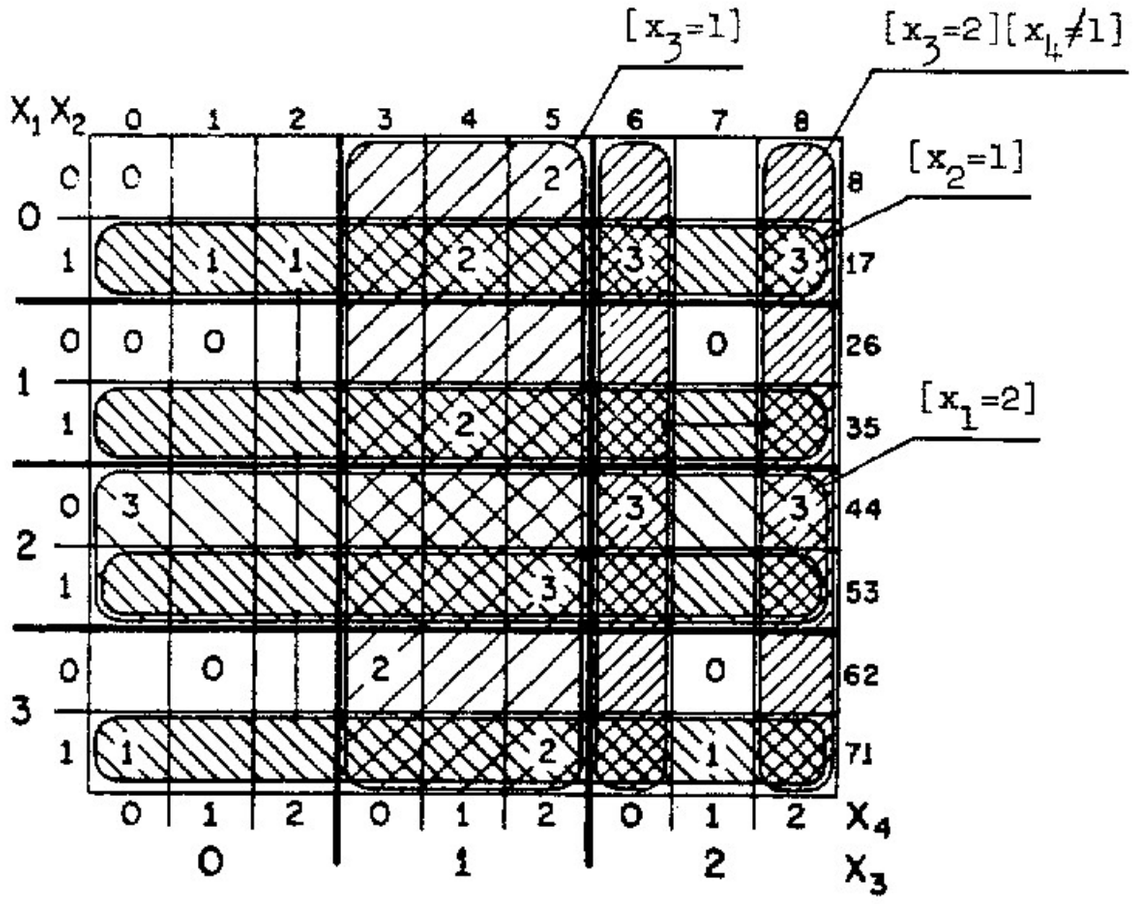


Fig. 2.

Another VL_1 expression for the same function f is

$$3[x_3 = 2][x_4 \neq 1] \vee 3[x_1 = 2] \vee 2[x_3 = 1] \vee 1[x_2 = 1]. \tag{48}$$

If the simplicity of VL_1 formulas is measured, e.g., by the number of selectors in them, then (48) is clearly much simpler than (47). Fig. 2 shows the sets of cells in the GLD representation of f which correspond to terms of (48).

4.2. Definition of an optimal VL_1 formula under a lexicographic functional. Depending on the application, different properties of VL_1 formulas may be desirable when expressing a given VL function, e.g., the minimal number of terms, of selectors, the minimal 'cost' of evaluation, etc. Therefore, in defining the concept of optimality of VL_1 formulas, we will not assume as the criterion of optimality any one arbitrary cost (or complexity) functional, but will make the definition explicitly dependent on an assumed functional.

We consider it theoretically important and practically useful that algorithms designed for the synthesis of VL_1 formulas should provide the availability of a number of different optimality functionals–from which the user may select the most appropriate for his type of application. For computational feasibility, however, it is desirable that the available functionals be expressed in one standard form. We found it convenient for computations and also useful in practice to assume that functionals are in the form:

$$A = \langle a\text{-list}, \tau\text{-list} \rangle, \tag{49}$$

where a-list, called *attribute* (or *criteria*) list, is a vector $a = (a_1, a_2, \ldots, a_l)$,

where the a_i denote single- or many-valued attributes used to characterize DVL_1 formulas, and τ-list, called *tolerance list*, is a vector $\tau = (\tau_1, \tau_2, \ldots, \tau_l)$, where $0 \leqslant \tau_i \leqslant 1$, $i = 1, 2, \ldots, l$, and the τ_i are called *tolerances for attributes* a_i. Functionals in the form (49) are called *lexicographic functionals.*

A DVL_1 formula V is said to be an *optimal* DVL_1 *expression for* f *under functional* A iff:

$$A(V) <^{\tau} A(V_j) \tag{50}$$

where

$$A(V) = (a_1(V), a_2(V), \ldots, a_l(V)),$$
$$A(V_j) = (a_1(V_j), a_2(V_j), \ldots, a_l(V_j)),$$

and $a_i(V)$, $a_i(V_j)$ denote the value of the attribute a_i for formula V and V_j, respectively. V_j, $j = 1, 2, 3, \ldots$–all irredundant DVL_1 expressions for f (a DVL_1 expression is called *irredundant*, if removing any term or selector from it makes it no longer an expression for f). $<^{\tau}$ denotes a relation, called the *lexicographic order with tolerance* τ, defined as:

$$A(V) <^{\tau} A(V_j) \text{ if } \begin{cases} a_1(V_j) - a_1(V) > T_1 \text{ or} \\ |a_1(V_j) - a_1(V)| \leqslant T_1 \text{ and } a_2(V_j) - a_2(V) > T_2 \text{ or} \\ \vdots \\ \cdots \text{ and } a_l(V_j) - a_l(V) \geqslant T_l, \end{cases}$$

$$T_i = \tau_i (a_{i\max} - a_{i\min}), \quad i = 1, 2, \ldots, l,$$

$$a_{i\max} = \max_j \{a_i(V_j)\}, \qquad a_{i\min} = \min_j \{a_i(V_j)\}.$$

Note that if $\tau = (0, 0, \ldots, 0)$ then $<^{\tau}$ denotes the lexicographic order in the usual sense. In this case, A is specified just as $A = (a\text{-list})$.

To specify a functional A one selects a set of attributes, puts them in the desirable order in the a-list, and sets values for tolerances in the τ-list.

AQVAL/1

The synthesis of optimal VL_1 formulas under a specified functional is a complex computational problem, especially when n, d_i and $c(F^1)$ are not small.

An efficient computer program, called AQVAL/1, has been developed at the University of Illinois for the purpose of such synthesis. An interested reader is referred to [11, 19, 22].

4. Applications to pattern recognition and machine learning

There exist a large number of problems which can be characterized as follows: given is some information about an object (e.g., a picture, outcomes of various

measurements, qualitative descriptions, etc.) and the problem is to assign to the object a name, or a class it belongs to, or, generally, to assign to it a decision from a certain set of predetermined decisions. For example, a decision can be: 'this is a picture of an aeroplane', or, 'the data indicate that the patient has a cancer of lungs', or 'this is Miss Elizabeth Wonderful'. An important characteristic of such problems is that the information about objects associated with the same decision can be in a great variety of forms and can consist of many different pieces of information, some of which may be relevant and some not relevant to the decision.

Also, in such problems there is no clear definition of the relationship which relates the decisions to objects. This relationship is given only by examples of objects with the associated decision. Thus, a problem arises of determining or "learning' a general formal expression characterizing this relationship from the given examples ('learning from examples'). Another similar class of problems is that there can be given a set of objects or data (e.g., animals living in certain environment, locations of planets in the solar system or data about people living in a given society) and the problem is to detect or 'discover' relationships ('patterns') which may be characteristic to the objects or data (e.g., to determine a hierarchical classification of animals, to 'discover' Keppler's laws or determine reasons for marriage crisis).

The problems delineated above are the core problems of the area of pattern recognition and machine learning. We will show now briefly how the VL_1 system can be useful for problems of this kind.

Let us first observe that many objects can be described by a list of values of some features or attributes which characterize these objects. For example, one can characterize a human being by statements such as:

$$\begin{aligned} \text{height} &= 6'7'', & \text{hair-color} &= \text{green}, \\ \text{sex} &= \text{male}, & \text{intelligence} &= \text{dumb}, \\ \text{blood type} &= \text{O}, & \text{age} &= 27, \\ \text{education} &= \text{M.D.}, & \text{character} &= \text{argumentative}, \\ \text{relation-to-money} &= \text{greedy}, & \text{status} &= \text{divorced}, \end{aligned} \tag{51}$$

where '=' stands loosely for "is'.

The above statements closely resemble selectors in VL_1

$$[L \# R] \tag{52}$$

in which L is a single variable, $\#$ is the equality sign '=' and R is a value of the variable. Since all the statements (51) characterize the same person, therefore these statements are equivalent to a conjunction of selectors:

$$[\text{height} = 6'7''][\text{hair-color} = \text{green}] \ldots [\text{status} = \text{divorced}] \tag{53}$$

assuming that the output domain of the selectors is $D = \{\text{false}, \text{true}\}$.

If the order of all attributes or characteristics is predefined, then an equivalent

description of a person can be just in a form of a list of values of these attributes:

$$(6'7'', \text{green, male}, \ldots, \text{divorced}) \tag{54}$$

that is it can be treated as an event in an event space E spanned by the cartesian product of the domains of all attributes. Thus, each description of a person in the form (51) will correspond to a point in the space E. Suppose now that a group of people is classified into m classes, each class being associated with a certain decision. Assume, however, that the rule of classification is unknown, the only thing we know are samples of people from each class. These samples give us event sets E_k, $k = 1, 2, \ldots, m$, where an E_k consists of events describing individuals in the sample from class k (assume here that E_k are disjoint). Suppose that the problem is to determine or discover, from the sets E_k, the rule which underlies the classification. It is easy to see, that such a problem, in general, is unsolvable. It is so, because for each family of sets, $\{E_k\}$, $k = 1, 2, \ldots, m$, there can be a very large or even infinite number of classification rules which will classify correctly people represented in sets E_k, but may classify differently people not represented in E_k (we will call such rules *sample-equivalent*). If we have no other information than sets E_k, it is impossible to tell which one of the above classification rules is the 'true' one. A reasonable approach to this problem is to seek among all the sample-equivalent rules a classification rule which is the simplest in some sense, or computationally most efficient. Such a rule will be easier to use and test than more complex rules, and, consequently, it will be easier to prove or disprove it, or to modify it.

To have a classification rule is equivalent to having descriptions of individual classes. If an object matches a description of class k, then it will be classified as a member of this class. In the case of application of VL_1 to a classification problem, the descriptions of classes will be expressed as VL_1 formulas. Among various methods of using VL_1 for solving classification problems we will consider two:

DCD method which produces the so-called *d*ependent *c*lass *d*escriptions.

ICD method which produces the so-called *i*ndependent *c*lass *d*escriptions.

Method DCD produces, based on sets $E_1, E_2, \ldots, E_m$, the simplest (according to some functional A), DVL_1 formula V which expresses the function

$$f: \mathrm{E} \rightarrow \{1, 2, \ldots, m\}, \tag{55}$$

defined by sets $E_1, E_2, \ldots, E_m$:

$$E_k = \{e \mid f(e) = k\}, \quad k = 1, 2, \ldots, m \tag{56}$$

(it is assumed that m is the largest and 1 is the smallest element of the output domain).

If a person has a description e, then that person is classified into class k if the value of the formula, $V(e)$, is k. The formula takes value k if a term with constant k is satisfied and no term with a higher constant is satisfied. The union of terms with constant k can then be treated as a description of class k. This description is

dependent on descriptions of class $i > k$, in the sense that satisfying this description is only a necessary condition for an object to be classified to class k. The sufficient condition is that, in addition, no description of class $i > k$ is satisfied.

Method ICD differs from DCD in that it assumes that each class is described by an independent VL_1 formula with the output domain

$$\{0, 1\}$$

where '1' represents a decision 'belongs to the class' and '0'–'does not belong to the class'.

Let us illustrate the above considerations by examples.

Example 1. First, we will consider a very simple two-class classification problem.

Fig. 3 shows two sets of objects F^1 and F^0 which are samples of objects taken from class 1 and 0, respectively. The problem is to find a 'simple' sample-equivalent classification rule based on sets F^1 and F^0.

When a person tries to solve this problem, that person usually looks for differences in the spatial relationships between elements in patterns of the two classes. In this case, however, there are no such differences, and therefore it seems to be difficult (for a person) to find a rule. On the other hand, for a computer program it is simpler, e.g., to count the numbers of elements of the same kind (e.g. circles, triangles, etc.) than to determine their relationships. In the example above, just such features happened to provide a sufficient representation of the objects (the problem of how to determine the initial representation of objects is a separate problem which goes beyond the scope of this paper). The appropriate rule which involves these features can be found very easily using the VL approach. Assume that the numbers of circles, ellipses, triangles, and squares in the 'pictures' in F^1 or F^0 are elements of the input domains D_1, D_2, D_3 and D_4, respectively. Thus, to describe any 'picture' in F^1 or F^0 it is sufficient to assume $D_1 = \{0, 1, 2\}$, $D_2 = \{0, 1, 2\}$, $D_3 = \{0, 1, 2, 3\}$ and $D_4 = \{0, 1, 2\}$. Thus, the event space is $E(3, 3, 4, 3) = D_1 \times D_2 \times D_3 \times D_2$. Sets F^1 and F^0 can now be represented by event sets $\{(x_1, \ldots, x_4)\}$, $x_i \in D_i$. Applying the DCD method, the problem becomes that of finding a VL_1 expression of the function

$$f\colon E(3, 4, 3, 3) \to \{0, 1\}$$

defined by event sets

$$F^1 = \{e \,|\, f(e) = 1\}$$

and

$$F^0 = \{e \,|\, f(e) = 0\}.$$

An optimal VL_1 expression of this function with regard to the functional $A = \langle t, s \rangle$

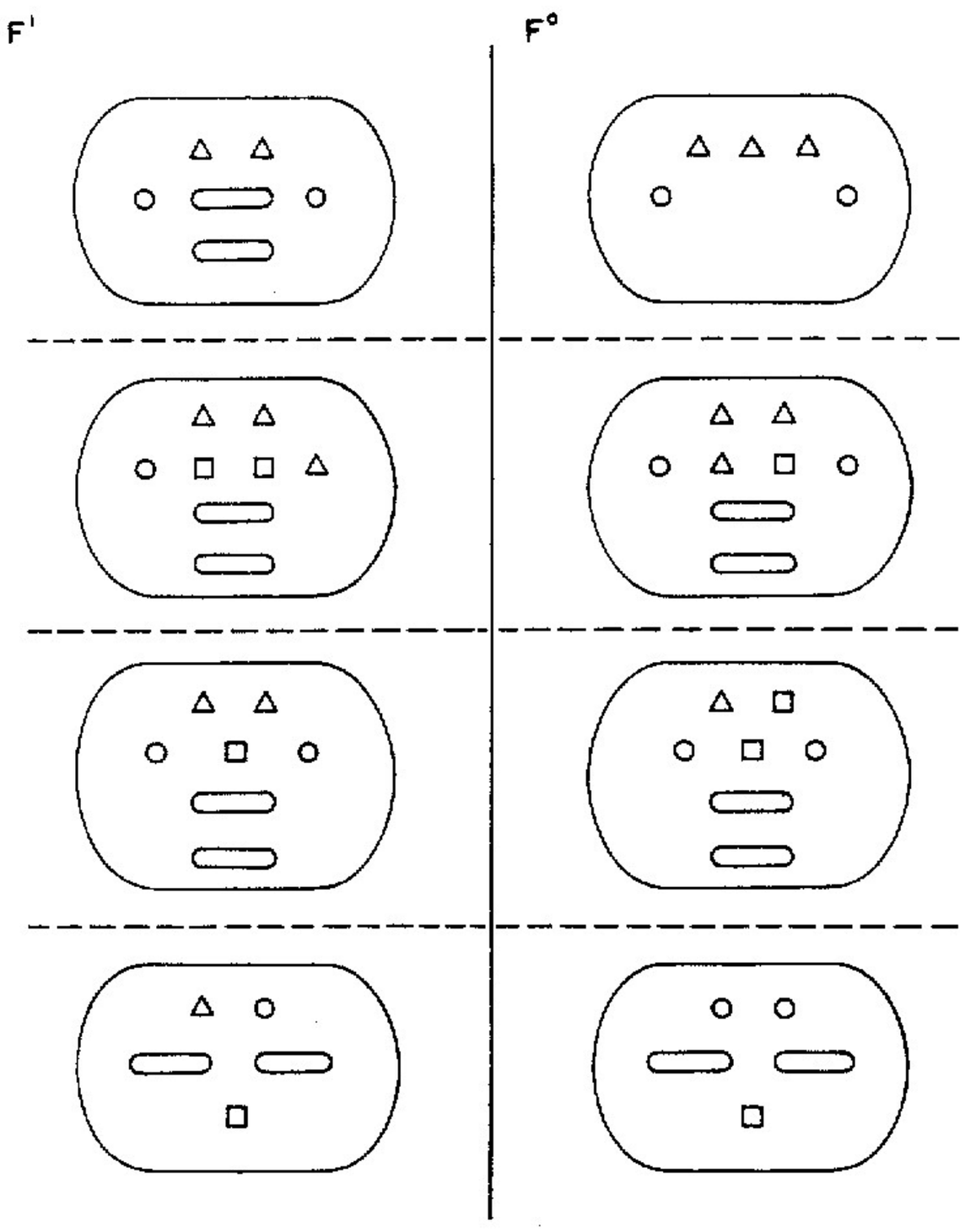


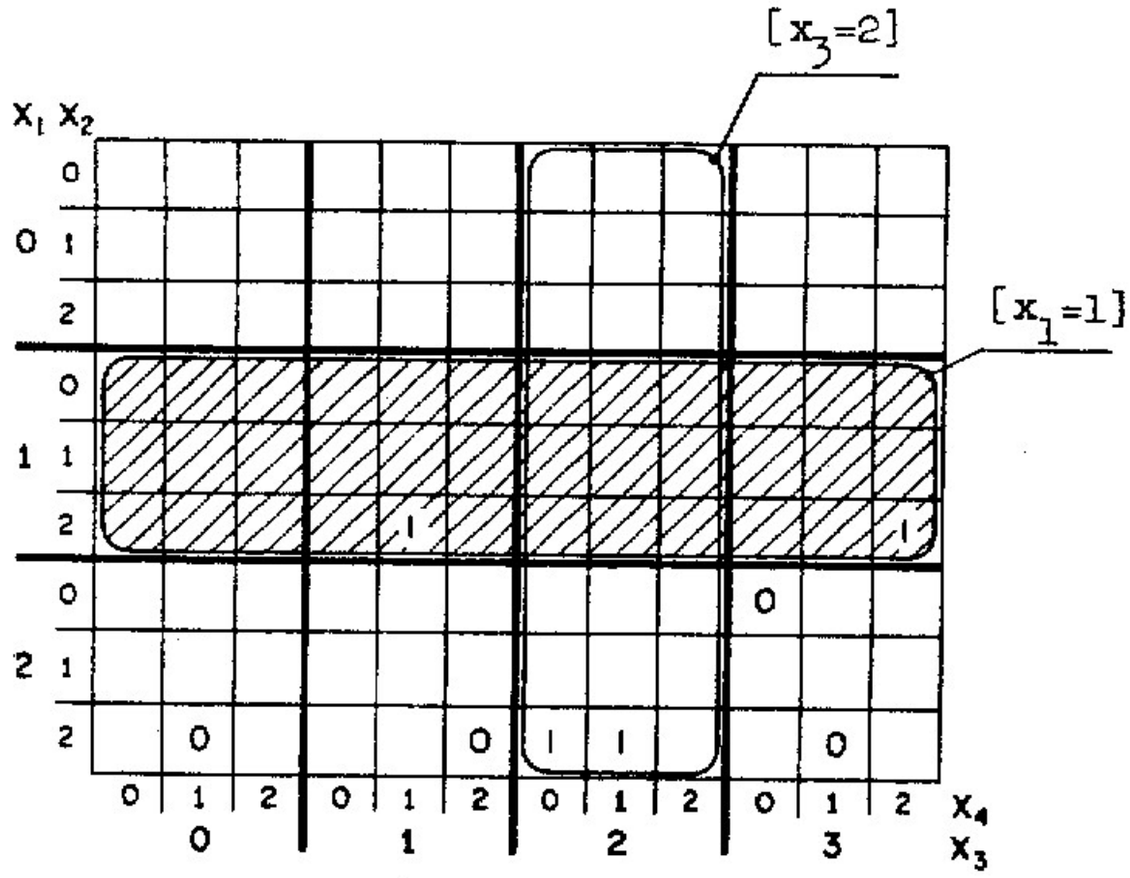
Fig. 3.

(t – number of terms, s – number of selectors) is:

$$V_1 = [x_1 = 1] \vee [x_3 = 2] \tag{57}$$

The formula V_1 can be interpreted: '*if an object has* 1 *circle or* 2 *triangles, then it belongs to the class* 1, *otherwise to class* 0'. Fig. 4 illustrates this formula graphically.

The ICD method requires that each class will have an independent description. The formula V_1 is already an independent description of class 1, so we only need



$f = [x_1 = 1] \vee [x_3 = 2]$

Fig. 4.

an independent description of class 0. To obtain it we seek the simplest VL_1 formula (with regard to the same functional $A = \langle t, s \rangle$) which takes value 1 (which represents now decision 'belongs to class 0') for events of F^0 and value 0 (which represents decision 'does not belong to class 0') for events of F^1. It is easy to see from diagram in Fig. 4 that this formula is:

$$V_0 = [x_1 = 2][x_3 \neq 2]$$

The formula says: 'an object belongs to class 0 if it has 2 circles and not 2 triangles'. It is worth noting that V_0 is not the inverse of V_1 and, therefore, using V_1 as a DCD classification rule is not equivalent to using the pair (V_1, V_0) as a ICD classification rule. These two rules are only sample-equivalent. For example, to event $e = (0, 1, 1, 2)$, the formula V_1 (used as DCD rule) will assign the decision 'class 0', while the pair (V_1, V_0) will assign to it no decision.

Example 2. In this example we will consider a more complex multi-class classification problem. Fig. 5 presents 14 groups of 'animals'. It is assumed that each group includes all different representants of 'animals' of the same one species.

The problem is to design an efficient classification system which for any given 'animal' from Fig. 5 will determine the species to which it belongs. The design of such a system involves a number of different steps: from determining an initial representation of objects to the hardware or software implementation of a classification rule. We will restrict ourselves here only to the problem of determining a classification rule which is a VL_1 formula optimal with regard to the functional $A = \langle t, s \rangle$.

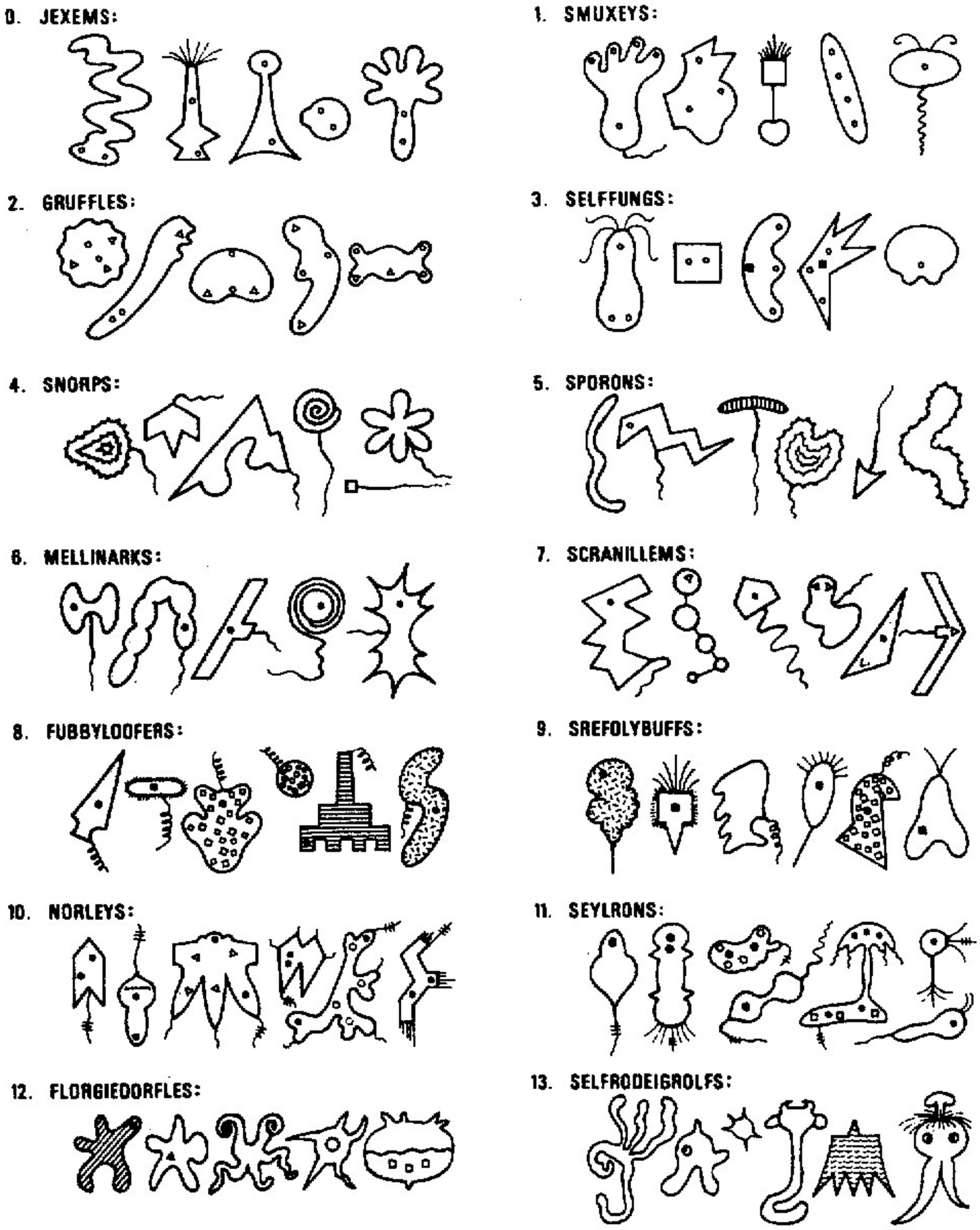


Fig. 5.

As an initial representation of 'animals' we accept a representation which uses the following descriptors (attributes) (by D_i is denoted the domain of the descriptor x_i):

x_1 is the number of black circles on the body, $D_1 = \{0, 1, 2\}$, where 0 denotes no black circles, 1 denotes 1 and 2, 2 or more;

x_2 is the number of tails $D_2 = \{0, 1\}$, where 0 is no tails and 1 is one or more;

x_3 is the number of crossmarks on tails, $D_3 = \{0, 1, 2\}$, where 0 means no crossmarks, 1 means one or two and 2, three;

x_4 is the number of easily distinguished extremities $D_4 = \{0, 1\}$, where 0 means none or one extremity, 1, two or more;

x_5 is the type of body texture, $D_5 = \{0, 1, 2, 3, 4, 5, 6\}$;

x_6 is the number of empty circles on the body, $D_6 = \{0, 1, 2\}$, where 0 is none or one circle, 1 is two and 2 three or more.

x_7 is the number of empty squares on the body, $D_7 = \{0, 1\}$, where 0 is no squares and 1 is one or more;

x_8 is the number of empty triangles on the body, $D_8 = \{0, 1\}$, where 0 means no triangle and 1 means one or more;

x_9 is the type of tail, $D_9 = \{0, 1, 2\}$, where 0 means no tail or more than one, 1 means straight tail and 2 spring tail;

x_{10} is the shape of the body, $D_{10} = \{0, 1, 2, 3\}$, where 0 is irregular (other than specified as 1, 2 or 3), 1 means ellipse, 2 circle and 3 triangle or square;

x_{11} is number of sharp or straight angles, $D_{11} = \{0, 1\}$, where 0 is no angles and 1 is one or more;

x_{12} is the number of 'eyes' (half-black circles), $D_{12} = \{0, 1\}$, where 0 means no eyes and 1, one or more;

x_{13} is the number of black squares on the body $D_{13} = \{0, 1\}$, where 0 means no black squares and 1 means one or more.

All animals are described in terms of the descriptors $x_1, \ldots, x_{13}$. Descriptions of the members of one species are combined into sets F^k,

$$F_k = \{e \mid e \text{ represents an 'animal' from the species } k\}, \quad k = 0, 1, \ldots, 13$$

The next step is to develop a classification rule with the help of the AQVAL/1 program.

Method DCD. We assume that the output domain consists of numbers $0, 1, \ldots, 13$ representing species are marked in Fig. 5.

Descriptions $C(k)$, $k = 0, 1, \ldots, 13$ of individual species (0, 1, . . . , 13, respectively) based on the VL_1 formula obtained from AQVAL/1 are:

$$C(0) = [x_6 = 1][x_8 = 0][x_{10} = 0],$$

$$C(1) = [x_3 = 0][x_4 = 1][x_6 = 2][x_{13} = 0] \vee [x_2 = 0][x_6 = 2][x_{13} = 0] \vee [x_1 = 0][x_2 = 1][x_6 = 0][x_9 = 0][x_{13} = 0],$$

$$C(2) = [x_6 = 1][x_{11} = 0],$$

$$C(3) = [x_1 = 0][x_6 = 1, 2] \vee [x_2 = 0][x_4 = 0][x_5 = 0],$$

$$C(4) = [x_1 = 0][x_5 = 1][x_8 = 0][x_9 = 1],$$

$$C(5) = [x_1 = 0][x_4 = 0][x_8 = 0][x_9 = 0, 1][x_{13} = 0],$$

$$C(6) = [x_3 = 0][x_5 = 1][x_8 = 0][x_9 = 1],$$

$$C(7) = [x_3 = 0][x_5 = 0, 1][x_9 = 1] \vee [x_4 = 0][x_8 = 1],$$
$$C(8) = [x_9 = 2][x_{13} = 1],$$
$$C(9) = [x_2 = 1][x_3 = 0][x_5 = 0, 5] \vee [x_1 = 0][x_2 = 1],$$
$$C(10) = [x_1 = 2][x_3 = 2][x_5 = 1],$$
$$C(11) = [x_2 = 1],$$
$$C(12) = [x_5 = 1, 4] \vee [x_8 = 1] \vee [x_5 = 0].$$

The class descriptions are dependent, therefore, in making a classification decision, the description $C(k)$, $k \in \{0, 1, \ldots, 12\}$, should be evaluated only if none of the descriptions $C(i)$, $i > k$, were satisfied. Class 13 is implied when none of the descriptions $C(k)$, $k = 0, 1, \ldots, 12$, is satisfied.

Let us interpret some of the descriptions, e.g., $C(0)$ and $C(8)$. An 'animal' will be classified as belonging to species 0 (Jexums), if variable x_6 equals 1 (i.e., if the 'animal' has 2 empty circles on the body), variable x_8 equals 0 (i.e., if the 'animal' has no triangles on the body) and variable x_{10} equals 0 (i.e., its body is 'irregular'). An animal is classified as belonging to species 8 (Fubbyloofers), if its description does not satisfy any of the descriptions $C(0)$, $C(1), \ldots, C(7)$, and if variable x_2 equals 2 (i.e., the animal has a 'spring' tail) and if variable x_{13} equals 1 (i.e., the animal has one black circle on the body). It is easy to see, that in this particular case, the description $C(8)$ is in fact independent of other descriptions, though for other descriptions this may not be true (e.g., description $C(11)$).

Method ICD. In this case each species is described by a separate binary-valued formula. The formulas $\hat{C}(k)$, $k = 0, 1, \ldots, 13$ constituting independent descriptions of species, which were obtained from program AQVAL/1 are:

$$\hat{C}(0) = [x_6 = 1][x_8 = 0][x_{10} = 0],$$
$$\hat{C}(1) = [x_2 = 1][x_4 = 1][x_5 = 0] \vee [x_2 = 0][x_6 = 2][x_{13} = 0]$$
$$\vee [x_1 = 0][x_2 = 1][x_6 = 0][x_9 = 0][x_{13} = 0],$$
$$\hat{C}(2) = [x_6 = 1][x_8 = 1],$$
$$\hat{C}(3) = [x_4 = 0][x_5 = 0][x_6 = 0, 2][x_7 = 0][x_9 = 0][x_{10} = 0][x_{11} = 0]$$
$$\vee [x_6 = 1][x_{10} = 3] \vee [x_2 = 0][x_{13} = 1],$$
$$\hat{C}(4) = [x_1 = 0][x_5 = 1][x_8 = 0][x_9 = 1],$$
$$\hat{C}(5) = [x_2 = 0][x_4 = 0][x_5 = 1][x_8 = 0] \vee [x_1 = 0][x_5 = 0, 6][x_9 = 1],$$
$$\hat{C}(6) = [x_1 = 1][x_3 = 0][x_5 = 1][x_9 = 1],$$
$$\hat{C}(7) = [x_1 = 1][x_5 = 0][x_9 = 1] \vee [x_4 = 0][x_6 = 0][x_8 = 1],$$
$$\hat{C}(8) = [x_9 = 2][x_{13} = 1],$$
$$\hat{C}(9) = [x_5 = 5][x_{13} = 0] \vee [x_2 = 1][x_9 = 0][x_{13} = 1] \vee [x_9 = 2][x_{13} = 0],$$
$$\hat{C}(10) = [x_1 = 2][x_3 = 2][x_5 = 1],$$
$$\hat{C}(11) = [x_1 = 1][x_3 = 2] \vee [x_1 = 2][x_5 = 0] \vee [x_3 = 1] \vee [x_1 = 1][x_2 = 1][x_9 = 0],$$

$$\hat{C}(12) = [x_2 = 0][x_4 = 1][x_5 = 1,4] \vee [x_5 = 0][x_6 = 0][x_8 = 1] \vee [x_2 = 0][x_5 = 0][x_6 = 0][x_{10} = 0][x_{11} = 1],$$

$$\hat{C}(13) = [x_4 = 1][x_5 = 0][x_6 = 0][x_8 = 0][x_{11} = 0] \vee [x_4 = 1][x_{10} = 2] \vee [x_5 = 3] \vee [x_2 = 2][x_4 = 1].$$

All these descriptions are independent, that is, if an 'animal' satisfies description $C(k)$, then it is classified as belonging to class k. It should be noted that for each 'animal' in Fig. 5, there will be only one description $C(k)$ which is satisfied (though, in the general case, there exist events, i.e., sequences of descriptor values, which satisfy more than one description).

The two examples above dealt with classification problems in which objects are described in terms of descriptors which are unary functions over the objects, that is, they are attributes characterizing objects as a whole. In the more general case, a description of an object may require descriptors which are multi-place inter-dependent functions relating to various parts of the objects. For example, a descriptor may be a relation 'above' which holds between some parts of an object being described. Such a problem can be, in a restricted sense, also handled within the VL_1 system. However, a more adequate formalism for such problems is provided by the system VL_2 which is an extension of system VL_1 and can be loosely characterized as a 'variable-valued first order predicated logic'. [20]

Concluding remarks

We described here a concept of a variable-valued logic system (a VL system) and motivated its development by a need for a formal system which would be more adequate than existing logic systems for problems of pattern recognition and artificial intelligence. We also described in detail one simple VL system called VL_1, discussed some of its formal properties and then showed how this system can be useful for some problems of pattern recognition and machine learning. We also pointed out that complex recognition problems, in particular, problems which require inter-dependent descriptors (e.g., descriptors which are relations over parts of objects) may require more general VL systems than VL_1. The work toward development of such more general systems is the subject for the current research on variable-valued logic. Let us summarize the kinds of recognition problems to which the VL_1 system can be especially useful. Namely, there are problems which:

involve a large number of classes and large number of variables, (because the class descriptions generated in this approach are computationally very efficient and involve usually a small subset of original variables),

are intrinsically non-linear, or 'multimodal',

involve nominal variables (i.e. variables whose values have no meaningful arithmetic relationships because they are labels of independent objects) and/or ordinal variables (values are linearly ordered),

assume that only a small number of representative learning samples is available,

make it desirable that classification procedures are easily comprehended (this is achieved because the classification procedure resembles the human inferential process).

The reader interest in more information about variable-valued logic and its applications is referred to [2,4,12,17–22]. [12,19,21] describe some results on the application of VL_1 to medical decision making, [19] also describes an application of VL_1 to the problem of discrimination of structural textures. [18] describes a comparison of VL_1 classification rules with some other types of classification rules. [20] gives current results on the development of system VL_2 and its application to general problems of inductive learning.

[2] reports a preliminary study on the relationship between variable-valued logic and syntactic approaches to pattern recognition. [22] describes an implementation of algorithms for the synthesis of optimal VL_1 formulas from non-optimal VL_1 formulas. [11] gives a user's guide and a description of the program AQVAL/1 (AQ7).

Acknowledgment

The author expresses his gratitude to A. B. Baskin and James Larson for their remarks, critical comments and proofreading of the paper. The research reported here was supported in part by the National Science Foundation under grant DCR 74-03514.

References

[1] R. B. Banerji, A language for the description of concepts, *General Systems* **9** (1964) 135–000.
[2] A. B. Baskin, A comparative discussion of variable-valued logic and grammatical inference, Report 663, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, July 1974.
[3] J. Bruner, et al., *A Study of Thinking*, (Wiley, New York, 1956).
[4] R. P. Cuneo, Selected problems of minimization of variable-valued logic formulas, Master's Thesis, Department of Computer Science, University of Illinois, Urbana, Illinois, May 1975.
[5] G. Epstein, G. Frieder, and D. C. Rine, The development of multiple-valued logic as related to computer science, *Computer*, 7(9) (1974) 20–32.
[6] C. Green, Application of theorem proving to problem solving, *Proc. First Int. Joint Artificial Intelligence Conf.*, Washington, D.C., 1969.
[7] F. Hayes-Roth, A structural approach to pattern learning and acquisition of classificatory power, *Proc. First Intern. Joint Conf. on Pattern Recognition*, (I.E.E.E., 1973).
[8] E. Hunt, et al., *Experiments in Induction*, (Academic Press, New York, 1966).
[9] M. Kochen, An experimental program for the selection of 'disjunctive hypothesis', *Proc. Spring Joint Computer Conference*, 1961.
[10] R. C. T. Lee, Fuzzy logic and the resolution principle. *J. ACM*, **19** (1972).
[11] J. Larson, R. S. Michalski, AQVAL/1: User's guide and program description, Department of Computer Science, University of Illinois, Urbana, May 1975.

[12] Lovby, Terje, Variable-valued logic applied to quality control in a clinical laboratory, Master's Thesis, Department of Computer Science, University of Illinois, Urbana, 1974.
[13] R. F. Mattrey, D. D. Givone, and C. M. Allen, Applying multiple-valued algebra concepts to neural modeling, *Proc. 1973 Intern. Symp. on Multiple-valued Logic*, Toronto, May 1973.
[14] R. S. Michalski, On the quasi-minimal solution of the general covering problem, *Proc. 5th Intern. Symp. on Information Processing* (FCIP 69), Vol. A3 (Switching Circuits), Yugoslavia, Bled, October 8–11, 1969 (in English).
[15] R. S. Michalski, Synteza wyrażeń minimalnych i rozpoznawanie symetrii funkcji logicznych, Praca doktorska, Politechnika Śląska 1969.
[16] R. S. Michalski, A geometrical model for the synthesis of interval covers, Report No. 461, Department of Computer Science, University of Illinois, Urbana, Illinois, June, 1971.
[17] R. S. Michalski, A variable-valued logic system as applied to picture description and recognition, *Graphic Languages*, Proc. IFIP Working Conf. on Graphic Languages, pp. 20–47, Vancouver, Canada, May 1972.
[18] R. S. Michalski, Discovering classification rules by the variable-valued logic system VL_1, *Proc. Third Intl. Joint Conf. on Artificial Intelligence*, Stanford, California, August, 1973.
[19] R. S. Michalski, AQVAL/1 – Computer implementation of a variable-valued logic systems VL_1 and examples of its application to pattern recognition, *Proc. First Intl. Joint Conf. on Pattern Recognition*, Washington, D.C., 1973 pp. 3–17.
[20] R. S. Michalski, Learning by inductive inference, *Proc. NATO Advanced Study Inst. on Computer Oriented Learning Processes*, Bonas, France, 1974.
[21] R. S. Michalski, Problems of designing an inferential medical consulting system, *First Illinois Conf. on Medical Information Systems*, Urbana, October 17–18, 1974.
[22] R. S. Michalski, Synthesis of optimal and quasi-optimal variable-valued logic formulas, *Proc. 5th Intern. Symp. on Multiple-valued Logic*, Bloomington, Indiana, May, 1975.
[23] R. S. Michalski, and B. H. McCormick, Interval Generalization of Switching Theory, *Proc. Third Annual Houston Conference on Computer and System Science*, Houston, Texas, April 26–27, 1971.
[24] D. C. Rine, A proposed multi-valued extension to ALGOL 68, *Kybernetics*, **2**, (1972) 107–111.
[25] J. C. Stoeffel, The theory of prime events, data analysis for sample vectors with inherently discrete variables, *Information Processing* 74, (North-Holland, Amsterdam, 1974).
[26] E. Towster, Several methods of concept formation by computer, Ph.D. Thesis, Department of Computer Science, University of Wisconsin, 1970.
[27] A. S. Wojcik, Relationships between Post and Boolean algebras with application to multi-valued switching theory, Rep. R-512, Coordinated Science Lab, Univ. Illinois, (June 1971).
[28] L. A. Zadeh, Fuzzy sets, *Information Control*, **8** (1965) 338–353.
[29] J. I. Zhuravlev, M. M. Kamilov and S. E. Tuliaganov, Algoritmy vichislenia ocienok i ich primienienie, Izd. FAN, Uzbekskoj SSR, Tashkent 1974.