74-1
R. J. [illegible]

PROCEEDINGS OF THE

1974 INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC

Morgantown, West Virginia

May 29, 30, 31, 1974

Participating and Sponsoring Agencies:

COMPUTER SOCIETY OF THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS

MATHEMATICAL AND INFORMATION SCIENCE DIVISION OF THE OFFICE OF NAVAL RESEARCH

WEST VIRGINIA UNIVERSITY

Assigned IEEE Catalog Number 74CHO845 - 8C

Assigned ONR Identifying Number NR 048 - 616

Distribution and Copies of This Proceedings
Can Be Obtained Through The IEEE COMPUTER SOCIETY

UNCLASSIFIED

Price and Copies are Available from:

IEEECS
Post Office Box 639
Silver Spring, Md.
20901

IEEECS
c/o Reseda Van and Storage
7112 Canby Avenue
Reseda, California
91335

I.E.E.E.
305 East 46th Street
9th Floor
New York, N.Y.
10017

VARIABLE-VALUED LOGIC:

System VL_1

by

R. S. Michalski
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ABSTRACT

The paper defines the concept of a variable-valued logic (VL) system and discusses in detail one specific VL system called VL_1.

The main motivation for the development of the VL system concept is to construct a well-defined, general formal system unifying various combinatorial problems and adequate, in particular, for problems of 'learning from examples' (problems which are of special interest to areas of pattern recognition and machine intelligence).

INDEX TERM:

Variable-valued logic, many-valued logic, switching functions, disjunctive normal forms, feature selection, classification rules, inductive systems.

INTRODUCTION

This paper describes a formal system, called VL_1, which is a specific system within a broader concept of a variable-valued logic (VL) system. One of the basic assumptions underlying the definition of a VL system is that every proposition (a VL formula) and each of the variables in the proposition, can accept its own, independent number of values ('truth-values'), which are selected on the basis of semantic or problem-oriented considerations.

The concept of 'truth', the fundamental concept of logic, is treated here as a property of our knowledge about nature in the broadest sense, similar to any other property characterizing nature. Therefore, the 'truth' is not considered as a concept which has an absolute number of 'truth-values' ('true - false', or 'true-false-possible', etc.) but as a concept which can take any number of values which is adequate in a given situation or appropriate for the meaning of the sentence. The main motivation for the development of the concept of a VL system is not theoretical, however, but practical, namely to construct a well-defined, general formal system, which would be adequate for solving various practical problems of combinatorial nature, in particular problems characterized as 'learning from examples'. Such problems are of special interest to the areas of pattern recognition and machine intelligence.

The VL_1 system, discussed here, though a very simple system, displays many of the desired features in the mentioned direction. It can be used as a formal basis for designing software or hardware systems able to infer ('learn') the simplest, in a well-defined sense, and also generalized, descriptions of complex objects (or object classes) from specific sample facts or examples characterizing the objects. These descriptions can then be used, e.g., as simple rules for classifying or recognizing the objects. Another application can be to infer the simplest expression for a deterministic relationship existing among various objects, from examples of this relationship.

The set of operations which VL_1 uses to create such descriptions is limited to only a few basic logical operations and one arithmetic operation -- addition (which is used only in a restricted context, namely to express symmetric properties of functions with regard to their variables or inversed variables).

The original facts or examples, from which system can 'learn', have to be expressed in a special format: as sequences of properties taken from various finite (ordered or unordered) domains (i.e. a 'questionnaire format') or, in the most general case, in the form of VL_1 formulas.

The concept of a VL system and the definition of VL_1 were first introduced[2] at the IFIP Working Conference on Graphic Languages, Vancouver, Canada, May 1972.

In this paper we present an extended definition of VL_1 which includes certain new concepts, such as a symmetric selector, exception and separation operations and a combined VL_1 formula. We also give here a grammatical description of the syntax of VL_1 and describe various concepts relevant to the system: equivalence-preserving operations, merging and simplication rules, concept of minimality of VL_1 formulas under a lexicographic functional, a geometric model of VL_1 formulas.

A Reader interested in applications of the system and its computer implementation is referred to other papers[1,2,3].

BASIC DEFINITIONS

Definition of a Variable-Valued Logic System

A variable-valued logic system (a VL system) is 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 variable, whose domains, denoted 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, denoted ${}^jD$, $j = 1, 2, \ldots$, are any non-empty sets.

S -- is a f.n. set of symbols, called connecting symbols,

R_F -- is a f.n. set of formation or syntactic rules which define well-formed formulas (wff) 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,

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 one specific VL system called VL_1. The definition of VL_1, given below, is an extension of the previous definition in paper[-]. It includes the following new concepts: a symmetric selector, an exception and separation operations and a combined VL_1 formula.

Definition of the VL_1 system

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 f.n. sets of elements called input semantic units. To be specific, we will assume, without loss of generality, that variables are

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

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

$$D_i = \{0,1,2,\ldots,\not{d}_i\}\ , \quad i = 1,2,\ldots,n \tag{3}$$

where $\not{d}_i = d_i - 1$.

Y -- is a set consisting of one output variable whose domain is a f.n. set of elements 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 d non-negative integers:

$$D = \{0,1,2,\ldots,\not{d}\} \tag{4}$$

where $\not{d} = d-1$.

A possible interpretation of elements of D is interpret them as truth-values ('degrees of truth') or 'degrees of preciseness' of statements describing relations among values of variables x_i. Another, more general, interpretation of elements of D can be to interpret them as decisions which are made depending on values of x_i. Note that in this case elements of D may not have any 'natural' order, unlike in the previous interpretation.

S -- is the set of the following connecting symbols:

$$= \quad \neq \quad \geq \quad \leq \quad \not\vee \quad \vee \quad \diagdown \quad \wedge \quad + \quad \neg \quad (\quad) \quad [\quad] \quad , \quad :$$

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:

   $\# \in \{=, \neq, \leq, \geq\}$ $\quad L \in \{\mathbf{x}, V_1\}$ $\quad R \in \{\mathbf{c}, V_2\}$

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

   $\mathbf{c}$ is a sequence of different non-negative integers ordered by the relation < (i.e., the smallest integer will be first and the largest integer will be last in the sequence) and separated by ',' or ':'. $\mathbf{c}$ also may be a name of a sequence such as defined above.

   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 $\mathbf{c}$, then R is called a simple reference. A selector with a simple referee and a simple reference, i.e. a form $[x_i \# \mathbf{c}]$, is called a simple selector.

   A simple reference, $\mathbf{c}$, is said to be in an extended form, if it contains no ':'. If in an extended form every maximal under inclusion** 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_1V_2), $V_1 \veebar V_2$, $V_1 \vee V_2$ and $V_1 | V_2$, where V, V_1 and V_2 are wffs or names of wffs, are wffs.

---

* when $\mathbf{x}$ denotes a variable whose values are such sequences.

** i.e., a sequence of consecutive integers which is not a part of another such sequence.

$\neg V$ is called the <u>inverse</u> of V (5)

$V_1 V_2$ is called the <u>product</u> (or <u>conjunction</u>) of V_1 and V_2 (6)

$V_1 \setminus V_2$ is called the <u>exception</u> V_2 <u>from</u> V_1 (or V_1 <u>except for</u> V_2) (7)

$V_1 \vee V_2$ is called the <u>sum</u> (or <u>disjunction</u>) of V_1 and V_2 (8)

$V_1 \mid V_2$ is called the <u>separation</u> of V_1 and V_2 (9)

R_I -- is a set of interpretation rules which assign to any VL_1 formula V a <u>value</u> $v(V) \in D$, depending on values of $x_1, \ldots, x_n$:

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

$$v(c) = c \qquad (10)$$

2. $v([L \# R]) = \begin{cases} \not{d}, & \text{if } v(L) \# v(R) \\ 0, & \text{otherwise} \end{cases}$,

where

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

$$v(x) = \sum_{i \in I} v(\tilde{x}_i), \qquad (11a)$$

where

$\sum$ denotes arithmetic sum

$$v(\tilde{x}_i) = \begin{cases} \dot{x}_i, & \text{if } \tilde{x}_i \text{ is } x_i \\ \not{d}_i - \dot{x}_i, & \text{if } \tilde{x}_i \text{ is } \neg x_i \end{cases} \qquad (11b)$$

$\dot{x}_i$ denotes a value of variable x_i, $\dot{x}_i \in D_i$

$I \subseteq \{1, 2, \ldots, n\}$

<u>Example</u>: 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:

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

$v(R)$, $R \in \{\mathbf{c}, V_2\}$, is the sequence $\mathbf{c}$, if R is $\mathbf{c}$; otherwise $v(V_2)$,

$v(L) \# v(R)$, $\# \in \{=, \neq, \leq, \geq\}$, is true if $v(L)$ is in relation $\#$ with $v(R)$. If R is **c**, then:

$v(L) = \mathbf{c}$ $(v(L) \neq \mathbf{c})$ is true if $v(L)$ is (is not) one of the integers in **c**, or is (is not) between any pair of integers in **c** separated by ':'

$v(L) \leq \mathbf{c}$ $(v(L) \geq \mathbf{c})$ is true if $v(L)$ is smaller than or equal to (greater than or equal to) every integer in **c** (in normal use of these relations **c** will consist of just one element).

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

3. $$v((V)) = v(V) \tag{12}$$

$$v(\neg V) = \not{d} - v(V) \tag{13}$$

$$v(V_1 \wedge V_2) = v(V_1V_2) = \min\{v(V_1), v(V_2)\} \tag{14}$$

$$v(V_1 \setminus V_2) = v(V_1), \text{ if } v(V_1V_2) = 0, \text{ otherwise } v(V_2) \tag{15}$$

$$v(V_1 \vee V_2) = \max\{v(V_1), v(V_2)\} \tag{16}$$

$$v(V_1 \mid V_2) = \begin{cases} v(V_1) & \text{if } v(V_2) = 0 \\ v(V_2) & \text{if } v(V_1) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

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

If there is more than one operation $\setminus$, 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 <u>positions</u> $0,1,2,\ldots d_i(d)$. In eq. 11b, $\dot{x}_i$ is then interpreted as the position of a value of variable x_i rather than the value itself. And as the <u>value</u> of a formula V is now taken the element of D whose position is $v(V)$ defined as above, rather than $v(V)$ itself.

DISCUSSION OF VL_1 FORMULAS

Examples of VL_1 and not VL_1 Formulas

A string of symbols from sets X, D and S, specified in the definition of VL_1, is a VL_1 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_1 formulas, and, for comparison, some strings which are not VL_1 formulas.

VL_1 formulas:

$$3[x_1=0,3:5][x_3\neq 2,4] \vee 2[x_2\neq 3] \vee 1[x_4\geq 2] \tag{18}$$

$$([x_2=1:3] \vee [x_3\neq 4,6])(1 \vee [x_1=0,2,4] \vee [x_3=3]) \tag{19}$$

$$\neg(2[x_1=3] \vee 1)[x_3\neq 2] \mid 2([x_1+x_3=0,5]) \vee [x_2=2] \vee 1[x_2\neq 0] \tag{20}$$

Not VL_1 formulas:

$$([x_1=0,2:5] = 3)[x_1\neq 1] \vee 2[[[x_1=0] \vee 1 = [x=1]]]=2] \tag{21}$$

$$3[x_3\neq 0:3] \vee \neg 2[[x_3=0,1,3,5] = [x_3=2]] \vee x_3 \tag{22}$$

$$[2[x_1+x_4=0] \vee 1 = x_1] \vee 1[2,3=2[x_2\neq 2]] \mid 3[x_2+x_4\leq 3] \tag{23}$$

String (21) 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 [ ]). (22) is not a VL_1 formula because a variable standing alone is not a wff. (23) 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 = .

Event Space

The interpretation rules R_I assign to any VL_1 formula a value -- an element of 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:

$$f: \; D_1 \times D_2 \times \ldots \times D_n \rightarrow D \tag{24}$$

where x denotes cartesian product and
→ 'mapping into'.

The set $D_1 \times D_2 \times \ldots \times D_n$, $D_i = \{0,1,\ldots,\not{d}_i\}$, $i = 1,2,\ldots,n$, includes all possible sequences of values of input variables and is called the <u>universe of events</u> or the <u>event space</u>. The event space is denoted by $\mathbf{E}(d_1, d_2, \ldots, d_n)$, where* $d_i = c(D_i)$, or, briefly, by $\mathbf{E}$. The elements of an event space $\mathbf{E}$, 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 <u>events</u> and denoted by e^j, $j = 0,1,2,\ldots,\not{d}$, where $\not{d}=d-1$, $d = c(\mathbf{E}) = d_1 d_2 \cdots d_n$. Thus, we can write:

$$\mathbf{E} = \mathbf{E}(d_1,d_2,\ldots,d_n)=\{(\dot{x}_1,\dot{x}_2,\ldots,\dot{x}_n)\,|\,\dot{x}_i \in D_i, i=1,2,\ldots,n\}=\{e^j\}_{j=0}^{\not{d}} \qquad (25)$$

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

$$\gamma: \mathbf{E} \rightarrow \{0, 1, \ldots, \not{d}\} \qquad (26)$$

specified by the expression:

$$j = \gamma(e) = x_n + \sum_{k=n-1}^{1} x_k \prod_{i=n}^{k+1} d_i \qquad (27)$$

$\gamma(e)$ is called the <u>number of the event</u> e. For example, the number of the event $e = (3, 2, 1, 2)$ in the space $\mathbf{E}(4, 4, 2, 3)$ 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 $\gamma(\dot{e})$ and $\mathbf{E}(d_1,d_2,\ldots,d_n)$ one can retrieve** the event e .

---

*c(S), where S is a set, denotes the cardinality of S.

**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 $\dot{x}_1$.

Examples of VL_1 Formulas Interpretation

Recall the formula (18): $3[x_1=0,3:5][x_3\neq 2,4] \vee 2[x_2\neq 3] \vee 1[x_4\geq 2]$. This formula can be interpreted as an expression of a function:

$$f: \; E(6, 4, 5, 5) \rightarrow \{0, 1, 2, 3\} \tag{28}$$

assuming that $c(D_1)=6$, $c(D_2)=4$, $c(D_3)=5$, $c(D_4)=4$. 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 $[x_4\geq 2]$ is satisfied. If none of the above conditions hold, the formula has value 0.

Consider a formula:

$$2[x_2+x_3+x_4=2,3] \vee 1[x_1=0] \tag{29}$$

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 the latter events, and only for them, the formula has value 1. For all the other events the formula has value 0. Note that the function expressed by formula (29) 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 (assuming that domains of these variables are of the same cordinality).

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 :

$$f: \ \mathbf{E} \rightarrow D \cup \{*\} \tag{30}$$

where * represents an unspecified value.

An incompletely specified function f can be considered as 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 of the functions 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 (30) will be called variable-valued logic functions or VL functions.

Multiple-output VL functions

As was previously mentioned, a VL_1 formula expresses a function

$$f: \mathbf{E} \longrightarrow D \tag{31}$$

In practical applications we may be interested in expressing not just one function f but a family of functions with the same domain **E**: (32)

$$\mathbf{f}: \mathbf{E} \longrightarrow \mathbf{D} \tag{33}$$

where

$$\mathbf{f} = ({}^1f, {}^2f, \ldots, {}^mf), \ {}^kf: \ \mathbf{E} \rightarrow {}^kD, \ k = 1, 2, \ldots, m,$$

$${}^kD = \{0, 1, 2, \ldots, {}^k\not{d}\}$$

$$\mathbf{D} = {}^1D \times {}^2D \times \ldots \times {}^mD$$

Thus:

$$\mathbf{f}: \ D_1 \times D_2 \times \ldots \times D_m \longrightarrow {}^1D \times {}^2D \times \ldots \times {}^mD \tag{34}$$

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, ..., m} and assuming that

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

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

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

   the value $v(V)$ is given to functions $\{{}^yf \mid y \# \mathbf{c}\}$.

For example, if V_1, V_2 and V_3 are VL_1 formulas which do not include the variable y, then

$$V_1[y = 1,2,3] \vee V_2[y = 3,4] \vee V_3[y = 1,3] \tag{35}$$

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

$${}^1f\colon\ V_1 \vee V_3,\quad {}^2f\colon\ V_1,\quad {}^3f\colon\ V_1 \vee V_2 \vee V_3,\quad {}^4f\colon\ V_2 \tag{36}$$

The selector $[y = \mathbf{c}]$ is called a <u>function selector</u>. A VL_1 formula which includes function selectors is called a <u>multiple-output VL_1 formula</u>. We extend a function $\mathbf{f}$ (33) to an incompletely specified function by assuming that:

$$f\colon\ \mathbf{E} \longrightarrow ({}^1D \cup \{*\}) \times ({}^2D \cup \{*\}) \times \ldots \times ({}^mD \cup \{*\}) \tag{37}$$

<u>Special Classes of VL_1 Formulas</u>

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 $\vee$ 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 within or outside the interval. Interval and cyclic selectors have direct correspondence to interval and cyclic literals discussed in papers[3,4].

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

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

<u>Definition 8.</u> A VL_1 formula which is a conjunction of simple clauses is called a <u>conjunctive</u> <u>simple</u> (or <u>normal</u>) VL_1 formula and denoted CVL_1. (For example, (19) is a CVL_1.)

<u>Definition 9.</u> A VL_1 formula which is a DVL_1 or CVL_1 is called a <u>simple</u> (or <u>normal</u>) VL_1 formula.

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

<u>Proof.</u> See paper[7].

<u>Specification of formation rules as rewriting rules of a contex-free grammar</u>

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 <u>different</u> 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.

<u>Formula</u>	$V ::= \mathcal{L} \mid V \mid \mathcal{L}$
<u>Clause</u>	$\mathcal{L} ::= \mathcal{P} \mid \mathcal{L} \vee \mathcal{P}$
<u>Phrase</u>	$\mathcal{P} ::= T \mid \mathcal{P} \veebar T$
<u>Term</u>	$T ::= \mathcal{C} \mid T \wedge \mathcal{C} \mid T\mathcal{C}$
<u>Component</u>	$\mathcal{C} ::= c \mid S \mid \neg V \mid (V)$
<u>Selector</u>	$S ::= [L \# R]$
<u>Left part (referee)</u>	$L ::= \mathbf{x} \mid V$
<u>Right part (reference)</u>	$R ::= \mathbf{c} \mid V$
<u>Selector relation</u>	$\# ::= = \mid \neq \mid \geq \mid \leq$

Output constants (values of dependent variable)	$c ::= 0 \mid 1 \mid 2 \mid \dots \mid \not{d}$
Arithmetic sum of literals	$\mathbf{x} ::= \tilde{x}_i \mid \mathbf{x} + \tilde{x}_i$
Literals	$\tilde{x}_i ::= x_i \mid \neg x_i$
Input variables	$x_i ::= x_1 \mid x_2 \mid \dots \mid x_n$
Simple reference	$\mathbf{c} ::= c_i \mid \mathbf{c}, c_i \mid \mathbf{c} : c_i$
Non-negative integers	$c_i ::= 0 \mid 1 \mid 2 \mid \dots$

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=' and then write the formula. For example:

$$V = 3[x_1=3,5] \vee 2[x_2 \neq 0] \vee 1 \tag{38}$$

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

$$v(V_1) = v(V_2) \tag{39}$$

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

$$V_1 \equiv V_2 \tag{40}$$

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

All operations on V_1 and/or V_2 which preserve relation (40) are called <u>equivalence-preserving operations</u>.

<u>Examples of Equivalence-Preserving Operations on VL_1 Formulas</u>

Let $V \in \mathbf{V}(X,D)$.

$V\phi \equiv V$ $\quad V \vee 0 \equiv V$ $\quad$ (identity elements) (41)

$V0 \equiv 0$ $\quad V \wedge \phi \equiv \phi$ $\quad$ (zero elements) (42)

$\neg(c) \equiv \overline{c}$ $\quad$ where $v(\overline{c}) = \phi - c$ (43)

If $\#$ is $=$ or $\neq$ then:

$\neg[x_i \# c] \equiv [x_i \overline{\#} c]$, $\quad$ where $'\overline{\#}'$ is $\begin{cases} '\neq' \text{ if } '\#' \text{ is } '=' \\ '=' \text{ if } '\#' \text{ is } '\neq' \end{cases}$

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

$\neg[x_i \geq c] \equiv [x_i \leq c^-]$, where c^- is equal $c-1$.

$\neg[x_i \leq c] \equiv [x_i \geq c^+]$, where c^+ is equal $c+1$.

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

$V_1(V_1 \vee V_2) \equiv V_1$ $\quad V_1 \vee V_1V_2 \equiv V_1$ $\quad$ (Absorption Laws) (44)

$V_1(V_2 \vee V_3) \equiv V_1V_2 \vee V_1V_3$ $\quad V_1 \vee V_2V_3 \equiv (V_1 \vee V_2)(V_1 \vee V_3)$ $\quad$ (Distributive Laws) (45)

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

$$\neg(\bigvee_{j\in J} V_j) \equiv \bigwedge_{j\in J} \neg(V_j) \qquad (46)$$

$$\neg(\bigwedge_{j\in J} V_i) \equiv \bigvee_{j\in J} \neg(V_j) \qquad \text{De Morgan's Laws*} \qquad (47)$$

<u>Examples</u>: $\neg(2[x_1=1:3][x_3\neq 2]) \equiv \overline{2} \vee [x_1\neq 1:3] \vee [x_3=2]$

where $v(\overline{2}) = \phi - 2$

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

Let $c_1 \cup c_2$ (c_1 <u>merged with</u> c_2) and $c_1 \cap c_2$ (c_1 <u>intersected with</u> c_2) denote compressed references obtained by set-theoretical summing and intersecting, respectively, the sets of elements in extended forms

* $\bigwedge$ and $\bigvee$ mean a conjunction and a disjunction of formulas, respectively.

of $\mathbf{c}_1$ and $\mathbf{c}_2$, then ordering the results by $<$ and finally transforming them into compressed forms. For example, if $\mathbf{c}_1$ is 2,4:6,8 and $\mathbf{c}_2$ is 0,3,5:7, then $\mathbf{c}_1 \cup \mathbf{c}_2$ is 0,2:8 and $\mathbf{c}_1 \cap \mathbf{c}_2$ 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:

$$T_1 = T[x_i \# \mathbf{c}_1] \text{ and } T_2 = T[x_i \# \mathbf{c}_2] \tag{48}$$

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

$$T[x_i \# \mathbf{c}_1] \vee T[x_i \# \mathbf{c}_2] \equiv T[x_i \# \mathbf{c}_1 \cup \mathbf{c}_2] \tag{49}$$

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

$$T[x_i \neq \mathbf{c}_1] \vee T[x_i \neq \mathbf{c}_2] \equiv T[x_i \neq \mathbf{c}_1 \cap \mathbf{c}_2] \tag{50}$$

Any two terms which can be expressed in the form (48) are called adjacent terms. (They can always be merged into one. If '#' is not the same in both selectors, then before applying (49)or (50), one of the selectors should be appropriately modified.)

If $\mathbf{c}_1 \cup \mathbf{c}_2$ is $0{:}d_i$ (i.e., its extended form includes every element of D_i), or if $\mathbf{c}_1 \cap \mathbf{c}_2$ is empty, then, respectively: $[x_i = \mathbf{c}_1 \cup \mathbf{c}_2] \equiv \phi$, $[x_i \neq \mathbf{c}_1 \cap \mathbf{c}_2] \equiv \phi$, and according to (41), formulas (49) and (50) become:

$$T[x_i = \mathbf{c}_1] \vee T[x_i = \mathbf{c}_2] \equiv T \tag{51}$$

$$T[x_i \neq \mathbf{c}_1] \vee T[x_i \neq \mathbf{c}_2] \equiv T \tag{52}$$

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 $\diagdown$ and $|$ it follows from (15) and (17) that:

$$V_1 \diagdown V_2 \equiv V_1[V_1V_2 = 0] \vee V_2$$

$$V_1 \mid V_2 \equiv V_1[V_2 = 0] \vee V_2[V_1 = 0]$$

MINIMAL VL_1 FORMULAS

A Planar Geometrical Representation of VL Functions and VL_1 Formulas

The merging and simplification rules (49, 50, 51, 52) 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 paper[4]. 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, Figure 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 paper[4].)

An example of the VL_1 formula expressing the function f is:

$$3[x_1=2][x_2=0][x_4=0,2] \vee 3[x_1=2][x_2=1][x_3=1] \vee$$
$$\vee 3[x_1=0,3][x_3=2][x_4=0,2] \vee 2[x_1=0,1][x_2=0][x_3=0,2][x_4=0,2] \vee$$
$$\vee 2[x_3=1][x_4=0,1] \vee 2[x_1=3][x_2=1][x_3=0][x_4=1,2] \vee$$
$$\vee 1[x_1=0][x_2=1][x_3=0][x_4=1,2] \vee 1[x_1=1,3][x_2=1][x_3=0,1] \vee$$
$$\vee 1[x_2=1][x_3=1,2][x_4=0,1] \quad , \qquad (53)$$

Another VL_1 expression for the same function f is:

$$3[x_3=2][x_4\neq 1] \vee 3[x_1=2] \vee 2[x_3=1] \vee 1[x_2=1] \qquad (54)$$

If the simplicity of VL_1 formulas is measured, e.g., by the number of selectors in them, then (54) is clearly much simpler than (53). Figure 2 shows the sets of cells in the GLD representation of f which correspond to terms of (54).

<u>Definition of the Minimal VL_1 Formula under a Lexicographic Functional</u>

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 minimality of VL_1 formulas, we will not assume as the criterion of minimality 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 cost functionals--

from which the user may select the most appropriate for his type of application. For computational feasibility, it is desirable, however, 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:

$$A = \langle \mathbf{a}\text{-list}, \boldsymbol{\tau}\text{-list} \rangle \tag{55}$$

where

a-list, called attribute (or criteria) list, is a vector $\mathbf{a} = (a_1, a_2, \dots, a_\ell)$, where the a_i denote single- or many-valued attributes used to characterize DVL_1 formulas,

$\boldsymbol{\tau}$-list, called tolerance list, is a vector $\boldsymbol{\tau} = (\tau_1, \tau_2, \dots, \tau_\ell)$, where $0 \leqslant \tau_i \leqslant 1$, $i=1,2,\dots,\ell$, and the τ_i are called tolerances for attributes a_i.

Functionals in the form (55) are called lexicographic functionals.

A DVL_1 formula V is said to be a minimal DVL_1 expression for f under functional A iff:

$$A(V) \overset{\tau}{\leqslant} A(V_j) \tag{56}$$

where

$A(V) = (a_1(V), a_2(V), \dots, a_\ell(V))$

$A(V_j) = (a_1(V_j), a_2(V_j), \dots, a_\ell(V_j))$

$a_i(V)$, $a_i(V_j)$ denote the value of the attribute a_i for formula V and V_j, respectively. V_j, $j=1,2,3,\dots$--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).

$\overset{\tau}{\leqslant}$ denotes a relation, called the lexicographic order with tolerance $\boldsymbol{\tau}$, defined as:

$$A(V) \overset{\tau}{<} A(V_j) \text{ if } \begin{cases} a_1(V_j)-a_1(V)>T_1 \\ \text{or} \quad a_1(V_j)-a_1(V)\leq T_1 \text{ and } a_2(V_j)-a_2(V)>T_2 \\ \text{or} \ldots\ldots \\ \vdots \\ \text{or} \ldots\ldots\ldots\ldots \text{ and } a_\ell(V_j)-a_\ell(V)\geq T_\ell \end{cases}$$

$$T_i = \tau_i \ (a_{imax} - a_{imin}),\ i=1,2,\ldots,\ell$$

$$a_{imax} = \max_j\{a_i(V_j)\},\ a_{imin} = \min_j\{a_i(V_j)\}$$

Note that if $\tau = (0,0,\ldots,0)$ then $\overset{\tau}{<}$ denotes the lexicographic order in the usual sense. In this case, A is specified just as A = <a-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 minimal VL_1 formulas under a specified functional is a complex computational problem, especially when n, d_i and d are not small.

A very 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 paper[1].

CONCLUDING REMARKS

The concept of VL_1 system is an extension of the concept of a many-valued logic system. It can be shown that VL_1 includes as special cases, for example, two-valued Boolean algebras (when $d_1=d_2=\ldots=d_n=d=2$ and the selectors $[x_1=1]$ and $[x_i=0]$ are interpreted as x_i and $\bar{x}_i$), a multi-valued logic system described by Givone[5], a multi-valued disjoint [illegible] system[6], etc.

The system can be applied to various practical problems of 'combinatorial nature', especially problems in the areas of pattern recognition, concept formation, inductive systems, etc. Program AQVAL/1, implementing a synthesis of minimal DVL_1 formulas, has already been successfully applied to a number of pattern recognition problems, such as synthesis of certain medical diagnostic procedures[1], design of a set of spacial filters for discrimination of non-uniform textures[1], 'discovering' simple classification rules[2], determination of simple rules for carbonate rocks classification in geology, and some others.

ACKNOWLEDGMENT

The author gratefully acknowledges the help he received from Professor Dennis Cudia in developing rewriting rules specifying the syntax of VL_1.

REFERENCES

1. Michalski, R. S., "AQVAL/1--Computer Implementation of a Variable-Valued Logic System and the Application to Pattern Recognition," Proceedings of the First International Joint Conference on Pattern Recognition, pp. 3-17, Washington, D.C., October 30-November 1, 1973.

2. ________, "A Variable-Valued Logic System as Applied to Picture Description and Recognition," GRAPHIC LANGUAGES, Proceedings of the IFIP Working Conference on Graphic Languages, Vancouver, Canada, May 1972.

3. ________, "Discovering Classification Rules by the Variable-Valued Logic System VL_1," Proceedings of the Third International Joint Conference on Artificial Intelligence, Theorem Proving and Logic, pp. 162-172, Stanford, California, August 20-24, 1973.

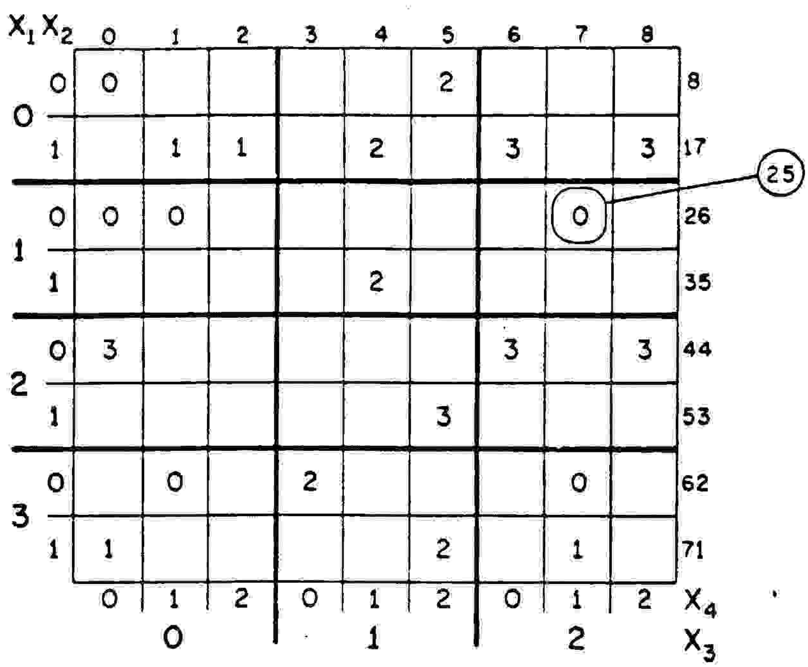
4. ________, "A Geometrical Model for the Synthesis of Interval Covers," Report No. 461, Department of Computer Science, University of Illinois, Urbana, Illinois, June 24, 1971.

5. Givone, P. D. and Snelsire, R. W., The design of multiple-valued logic systems, Rep. Department of Electrical Engineering, State University of New York at Buffalo, 1968.

6. Nutter, R. S., "Function simplification techniques for Postian multi-valued logic systems", Dissertation West Virginia University, 1971.

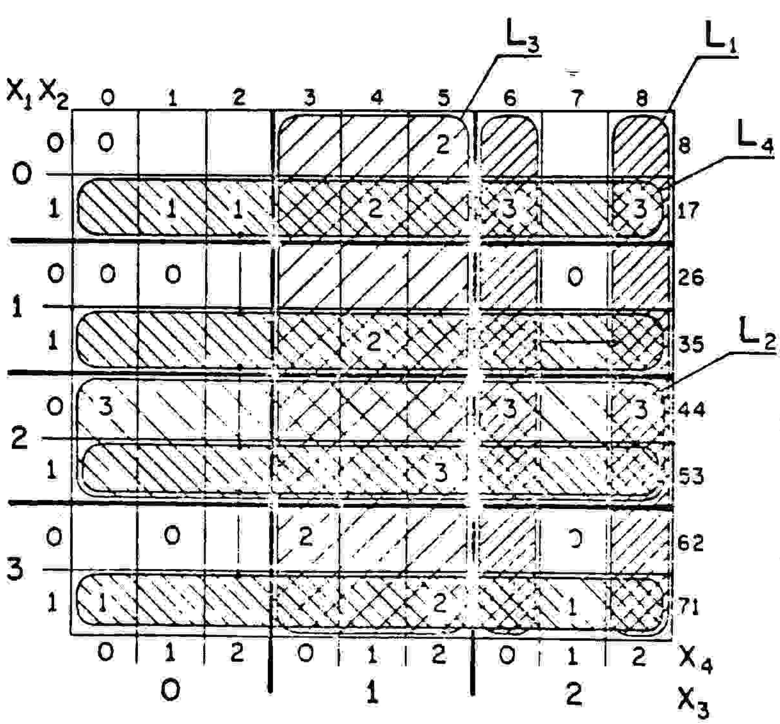
7. Michalski, R. S., Variable-valued logic system VL_1: I--elements of theory, to appear as a report of the Department of Computer Science, University of Illinois, Urbana, Illinois.

8. ________ and McCormick, B. H., "Interval generalization of switching theory", Proceedings of the Third Annual Huston Conference on Computer and System Science, Houston, Texas, April 26-27, 1971.



GLD representation of a VL Function E(4,2,3,3) $\rightarrow$ {0,1,2,3,*}

Fig.1.



L_1 =: $[x_3=2][x_4\neq 1]$
L_2 =: $[x_1=2]$
L_3 =: $[x_3=1]$
L_4 =: $[x_2=1]$

(L_i =: T_i, where T_i is a term,means that the set of cells L_i corresponds to the term T_i)

Sets of cells in the GLD corresponding to terms in formula (55)

Fig.2.