A PROGRAMMING LANGUAGE

Kenneth E. Iverson

Research Division, IBM Corporation

Yorktown Heights, New York

The paper describes a succinct problem-oriented programming language. The language is broad in scope, having been developed for, and applied effectively in, such diverse areas as microprogramming, switching theory, operations research, information retrieval, sorting theory, structure of compilers, search procedures, and language translation. The language permits a high degree of useful formalism. It relies heavily on a systematic extension of a small set of basic operations to vectors, matrices, and trees, and on a family of flexible selection operations controlled by logical vectors. Illustrations are drawn from a variety of applications.

The programming language outlined[1] here has been designed for the succinct description of algorithms. The intended range of algorithms is broad; effective applications include microprogramming, switching theory, operations research, information retrieval, sorting theory, structure of compilers, search procedures, and language translation. The symbols used have been chosen so as to be concise and mnemonic, and their choice has not been restricted to the character sets provided in contemporary printers and computers. A high degree of formalism is provided.

Basic Operations

The language is based on a consistent unification and extension of existing mathematical notations, and upon a systematic extension of a small set of basic arithmetic and logical operations to vectors, matrices, and trees. The arithmetic operations include the four familiar arithmetic operations and the absolute value (denoted by the usual symbols) as well as the floor, ceiling, and residue functions denoted and defined as follows:

Name	Symbol	Definition
floor	$\lfloor x \rfloor$	$\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$
ceiling	$\lceil x \rceil$	$\lceil x \rceil \geq x > \lceil x \rceil - 1$
residue mod m	$m \mid n$	$n = mq + m \mid n; 0 \leq m \mid n < m,$

where $\lfloor x \rfloor$, $\lceil x \rceil$, m, n, and q are integers.

The logical operations and, or, and not are defined upon the logical variables 0 and 1 and are denoted by $\wedge$, $\vee$, and an overbar. They are augmented by the relational statement (proposition) (x R y) defined as follows. If x and y are any entities (e.g., numerals, alphabetic literals, or logical variables) and R is any relation defined upon them, then (x R y) is equal to 1 or 0 according to whether the relation R does or does not hold between x and y. Thus (i=j) is the familiar Kronecker delta δ_{ij}, $(u \neq v)$ is the exclusive or function of the logical variables u and v, and

$$\text{sgn } x = (x > 0) - (x < 0)$$

is the familiar sign function, defined as +1, 0, or -1 according as x is strictly positive, zero, or strictly negative.

Vectors and Matrices

A vector is denoted by an underlined lowercase letter (e.g., $\underline{x}$), its i-th component by $\underline{x}_i$, and its dimension by $\nu(\underline{x})$. A matrix is denoted by an underlined uppercase letter (e.g., $\underline{X}$), its i-th row by $\underline{X}^i$, its j-th column by $\underline{X}_j$, its ij-th element by $\underline{X}^i_j$, its row dimension (i.e., the common dimension of its row vectors) by $\nu(\underline{X})$, and its column dimension by $\mu(\underline{X})$.

All of the basic operations are extended component-by-component to vectors and matrices. Thus,

$$\underline{z} = \underline{x} + \underline{y} \iff \underline{z}_i = \underline{x}_i + \underline{y}_i,$$

$$\underline{Z} = \underline{X} \times \underline{Y} \iff \underline{Z}^i_j = \underline{X}^i_j \times \underline{Y}^i_j$$

$$\underline{w} = \underline{u} \wedge \underline{v} \iff \underline{w}_i = \underline{u}_i \wedge \underline{v}_i,$$

$$\underline{w} = (\underline{x} < \underline{y}) \iff \underline{w}_i = (\underline{x}_i < \underline{y}_i)$$

The symbol $\underline{\epsilon}(n)$ will denote a logical vector of dimension n whose components are all unity, and $\underline{\epsilon}(n)$ therefore denotes the zero vector. The

dimension n will be elided whenever it is clear from context.

The cyclic left shift of a vector $\underline{x}$ is called left rotation. It is denoted by $k \uparrow \underline{x}$ and defined by the relation:

$$\underline{y} = (k \uparrow \underline{x}) \Longleftrightarrow \underline{y}_i = \underline{x}_j,$$

where $j = 1 + \nu(\underline{x}) \mid (i + k - 1)$. Right rotation is denoted by $k \downarrow \underline{x}$ and is defined analogously.

Reduction

For each basic binary operator $\odot$, the $\odot$-reduction of a vector $\underline{x}$ is denoted by $\odot/\underline{x}$ and defined by

$$\odot/\underline{x} = (\cdots((\underline{x}_1 \odot \underline{x}_2) \odot \underline{x}_3) \odot \cdots \odot \underline{x}_{\nu(\underline{x})}).$$

Thus $+/\underline{x}$ is the sum, and $\times/\underline{x}$ is the product of all components of $\underline{x}$. Moreover, if $\underline{u} = (1, 0, 1)$, then $+/\underline{u} = 2$, $+/\overline{\underline{u}} = 1$, $\wedge/\underline{u} = 0$, and $\vee/\underline{u} = 1$.

Reduction is extended to matrices row-by-row:

$$\underline{z} = \odot/\underline{X} \Longleftrightarrow \underline{z}_i = \odot/\underline{X}^i,$$

and column-by-column:

$$\underline{z} = \odot//\underline{X} \Longleftrightarrow \underline{z}_i = \odot/\underline{X}_i,$$

the column reduction being distinguished by the double slash.

For example, if $\underline{u} = (\underline{u}_1, \underline{u}_2)$ is a logical vector of dimension two, then De Morgan's law may be expressed as:

$$\wedge/\underline{u} = \overline{\vee/\overline{\underline{u}}}.$$

Moreover, this is the valid generalization of De Morgan's law to a vector $\underline{u}$ of arbitrary dimension.

The reduction operation can be extended to any relation R by substituting R for $\odot$ in the formal definition above. The parentheses in the definition now signify relational statements as well as grouping, and thus the expression $\not=/\underline{u}$ denotes the application of the exclusive-or over all components of the logical vector $\underline{u}$. For example, $\not=/(1, 0, 1) = ((1 \not= 0) \not= 1) = (1 \not= 1) = 0$. Induction can be used to show that

$$\not=/\underline{u} = 2 \mid +/\underline{u}, \text{ and } =/\underline{u} = \overline{2 \mid +/\overline{\underline{u}}}.$$

Hence, $\not=/\underline{u} = \overline{=/\overline{\underline{u}}}$, a useful companion to De Morgan's law.

Matrix Product

The conventional matrix product $\underline{A}\ \underline{B}$ may be defined as:

$$(\underline{A}\ \underline{B})^i_j = +/(\underline{A}^i \times \underline{B}_j).$$

Adopting the notation $\underline{A} \overset{+}{\times} \underline{B}$ for this product makes explicit the roles of the basic operations + and $\times$, and suggests the following useful generalization:

$$\underline{A} \overset{\odot_1}{\odot_2} \underline{B} = \odot_1/(\underline{A}^i \odot_2 \underline{B}_j),$$

where $\odot_1$ and $\odot_2$ are any operators or relations with suitable domains.

Thus $\underline{C} = \underline{M} \overset{\wedge}{=} \underline{X}$ is an "incidence matrix" such that $C^i_j = 1$ if and only if $\underline{M}^i$ and $\underline{X}_j$ agree in all components. If $\underline{M}$ is the memory of a binary computer (i.e., $\underline{M}$ is a logical matrix and row $\underline{M}^i$ is the i-th word of memory), and if $\underline{x}$ is an "argument register", then (treating $\underline{x}$ as a one-column matrix)

$$\underline{s} = \underline{M} \overset{\wedge}{=} \underline{x}$$

defines the sense vector $\underline{s}$ of an associative memory such that $\underline{s}_i = 1$ if and only if word $\underline{M}^i$ agrees with $\underline{x}$.

As further examples, $\underline{R} = \underline{P} \overset{+}{>} \underline{Q}$ gives the number of places in which a component of $\underline{P}^i$ exceeds the corresponding component of $\underline{Q}_j$, and $\underline{K} = \underline{P} \overset{\wedge}{>} \underline{Q}$ is a "covering matrix" which indicates which rows of $\underline{P}$ cover (exceed in every component) which rows of $\underline{Q}$.

De Morgan's law and the identity

$$\not=/\underline{u} = \overline{=/\overline{\underline{u}}}$$

establish a duality with respect to negation between $\wedge$ and $\vee$, and between $\not=$ and $=$. This duality is easily extended to matrices. For example,

$$\underline{A} \overset{\wedge}{=} \underline{B} = \overline{\overline{\underline{A}} \overset{\vee}{\not=} \overline{\underline{B}}}.$$

Selection

The formation of a vector $\underline{y}$ from a vector $\underline{x}$ by deleting certain components of $\underline{x}$ will be denoted by

$$\underline{y} = \underline{u}/\underline{x},$$

where $\underline{u}$ is a logical vector (of dimension $\nu(\underline{x})$ and $\underline{x}_i$ is deleted if and only if $\underline{u}_i = 0$. Thus $\underline{u}/\underline{x}$ and $\overline{\underline{u}}/\underline{x}$ provide a disjoint decomposition of $\underline{x}$, and $(\underline{x} > \epsilon)/\underline{x}$ is the vector of all strictly positive components of $\underline{x}$.

The operation $\underline{u}/\underline{x}$ is called compression and is extended to matrices by row and by column as follows:

$$\underline{Z} = \underline{u}/\underline{X} \Longleftrightarrow \underline{Z}^i = \underline{u}/\underline{X}^i,$$
$$\underline{Z} = \underline{u}//\underline{X} \Longleftrightarrow \underline{Z}_i = \underline{u}/\underline{X}_i.$$

The familiar identity concerning partitioned matrices can now be generalized as follows:

$$\underline{X} \overset{+}{\times} \underline{Y} = (u/\underline{X}) \overset{+}{\times} (u//\underline{Y}) + (\underline{u}/\underline{X}) \overset{+}{\times} (\underline{u}//\underline{Y}).$$

Since the identity depends only on the associativity and commutativity of the operators + and $\times$, it holds also for all operators (and relations) possessing these properties. Moreover,

$$\underline{u}/(\underline{X} \overset{+}{\times} \underline{Y}) = \underline{X} \overset{+}{\times} (\underline{u}/\underline{Y}),$$

and

$$\underline{u}//(\underline{X} \overset{+}{\times} \underline{Y}) = (\underline{u}//\underline{X}) \overset{+}{\times} \underline{Y}.$$

To illustrate other uses of compression, consider a bank ledger $\underline{L}$ so arranged that $\underline{L}^i$ represents the i-th account, and $\underline{L}_1$, $\underline{L}_2$, and $\underline{L}_3$ represent the vector of names, of account numbers, and of balances, respectively. Then the preparation of a list $\underline{P}$ (in the same format) of all accounts having a balance less than two dollars is described by the statement:

$$\underline{P} \leftarrow (\underline{L}_3 < 2\underline{\epsilon})//\underline{L},$$

where the arrow denotes specification (in the sense of the symbol = in Fortran and the symbol := in Algol).

Three useful operations converse to compression are defined as follows:

Mesh: $\underline{z} = \backslash \underline{a}, \underline{u}, \underline{b} \backslash \Leftrightarrow \overline{\underline{u}}/\underline{z} = \underline{a}; \quad \underline{u}/\underline{z} = \underline{b}$

Mask: $\underline{z} = /\underline{a}, \underline{u}, \underline{b}/ \Leftrightarrow \overline{\underline{u}}/\underline{z} = \overline{\underline{u}}/\underline{a}; \ \underline{u}/\underline{z} = \underline{u}/\underline{b}$

Expansion: $\underline{z} = \underline{u} \backslash \underline{b} \Leftrightarrow \underline{z} = \backslash \overline{\underline{\epsilon}}, \ \underline{u}, \underline{b} \backslash$.

They may be extended to matrices in the established manner and are related by the obvious identities:

$$\backslash \underline{a}, \underline{u}, \underline{b} \backslash = /\overline{\underline{u}} \backslash \underline{a}, \ \underline{u}, \ \underline{u} \backslash \underline{b} /,$$

$$/\underline{a}, \underline{u}, \underline{b}/ = \backslash \overline{\underline{u}} / \underline{a}, \ \underline{u}, \ \underline{u} / \underline{b} \backslash .$$

Special Vectors

In addition to the full vector $\underline{\epsilon}(n)$ already defined, it is convenient to define the prefix vector $\underline{a}^j(n)$ as a logical vector of dimension n whose first j components are unity. The suffix vector $\underline{\omega}^j(n)$ is defined analogously. An infix vector having i leading 0's followed by j 1's can be denoted by $i \downarrow \underline{a}^j$. When used in compression operations, these special vectors are very useful in specifying fixed formats.

Mixed Base Value

If the components of the vector $\underline{y}$ specify the radices of a mixed base number system and if $\underline{x}$ is any numerical vector of the same dimension, then the value of $\underline{x}$ in that number system is called the base y value of x, is denoted by $\underline{y} \perp \underline{x}$, and is defined formally by

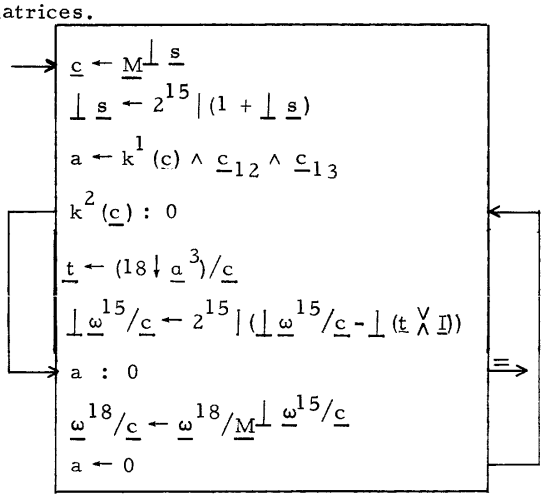
$$\underline{y} \perp \underline{x} = \underline{w} \overset{+}{\times} \underline{x},$$

where $\underline{w}$ is the weighting vector defined by $\underline{w}_{\nu(\underline{w})} = 1$ and $\underline{w}_{i-1} = \underline{w}_i \times \underline{y}_i$. For example, if $\underline{y} = (24, 60, 60)$ and if $\underline{x} = (1, 2, 5)$ is the elapsed time in hours, minutes, and seconds, then $\underline{y} \perp \underline{x} = 3725$ is the elapsed time in seconds.

The value of $\underline{x}$ in a decimal system is denoted by $(1\ 0\ \underline{\epsilon}) \perp \underline{x}$, and in a binary system by either $(2\ \underline{\epsilon}) \perp \underline{x}$, or $\perp \underline{x}$. Moreover, if y is any real number, then $(y\ \underline{\epsilon}) \perp \underline{x}$ denotes the polynomial in y with coefficients $\underline{x}$.

Application to Microprogramming

To illustrate the use of the notation in describing the operation of a computer, consider the IBM 7090 with a memory $\underline{M}$ of dimension $2^{15} \times 36$, a command vector $\underline{c}$ of dimension 36 representing the instruction next to be executed, a sequence vector $\underline{s}$ of dimension 15 representing the instruction counter, and a 3×15 index matrix $\underline{I}$ representing the three index registers. The instruction fetch phase of operation (excluding the channel trap) can then be described as in Figure 1. 0-origin indexing will be used for all vectors and matrices.



Instruction Fetch of IBM 7090

Figure 1

Step 1 shows the selection of the next instruction from the memory word specified by the instruction counter $\underline{s}$ and its transfer to the command register $\underline{c}$. Step 2 shows the incrementation (reduced modulo 2^{15}) of the counter $\underline{s}$. The logical function $k^1(\underline{c})$ of step 3 determines whether the instruction just fetched belongs to the class of instructions which are subject to indirect addressing, the bits $\underline{c}_{12}$ and $\underline{c}_{13}$ determine whether this particular instruction is to be indexed, and a is therefore set to unity only if indirect addressing is to be performed.

The function $k^2(\underline{c})$ determines whether the instruction is indexable. If $k^2(\underline{c}) = 0$, the branch from step 4 to step 7 skips the potential indexing of steps 5 and 6. As shown by step 6, indexing proceeds by oring together the index registers (i.e., rows of $\underline{I}$) selected by the three tag bits $\underline{t} = (18 \downarrow \underline{a}^3)/\underline{c}$, subtracting the base two value of the resulting vector from the address portion of the instruction $\underline{c}$, reducing the result modulo 2^{15}, and respecifying the address portion of $\underline{c}$ accordingly.

The fetch terminates immediately from step 7 if no indirect addressing is indicated; otherwise step 8 performs the indirect addressing and the indexing phase is repeated. Step 9 limits the indirect addressing to a single level. It may be noted that all format information is presented directly in the microprogram.

The description of the execution phase of computer operation will be illustrated by the family of instructions for logical functions in the 7090. Representing the 38-bit accumulator by the logical vector $\underline{u}$ (with $\nu(\underline{u}) = 38$, and with $\underline{u}_0$, $\underline{u}_1$, and $\underline{u}_2$ representing the sign, q and p bits, respectively), these instructions may be described as in Figure 2.

Or to storage	ORS	$\underline{M}^{\perp}\underline{\omega}^{15}/\underline{c} \leftarrow (\underline{a}^2/\underline{u}) \vee \underline{M}^{\perp}\underline{\omega}^{15}/\underline{c}$
Or to accumulator	ORA	$\underline{u} \leftarrow \underline{u} \vee (\underline{a}^2 \backslash \underline{M}^{\perp}\underline{\omega}^{15}/\underline{c})$
And to storage	ANS	$\underline{M}^{\perp}\underline{\omega}^{15}/\underline{c} \leftarrow (\underline{a}^2/\underline{u}) \wedge \underline{M}^{\perp}\underline{\omega}^{15}/\underline{c}$
And to accumulator	ANA	$\underline{u} \leftarrow \underline{u} \wedge (\underline{a}^2 \backslash \underline{M}^{\perp}\underline{\omega}^{15}/\underline{c})$
Exclusive or to accumulator	ERA	$\underline{u} \leftarrow (\underline{u} \neq (\underline{a}^2 \backslash \underline{M}^{\perp}\underline{\omega}^{15}/\underline{c}))$
		$\underline{a}^2/\underline{u} \leftarrow \underline{\epsilon}$
Complement magnitude	COM	$\underline{a}^1/\underline{u} \leftarrow \underline{a}^1/\underline{u}$
Clear magnitude	CLM	$\underline{a}^1/\underline{u} \leftarrow \underline{\epsilon}$
Change sign	CHS	$\underline{u}_0 \leftarrow \overline{\underline{u}}_0$
Set sign plus	SSP	$\underline{u}_0 \leftarrow 0$
Set sign minus	SSM	$\underline{u}_0 \leftarrow 1$
Store logical word	SLW	$\underline{M}^{\perp}\underline{\omega}^{15}/\underline{c} \leftarrow \underline{a}^2/\underline{u}$
Clear and add logical	CAL	$\underline{u} \leftarrow \underline{a}^2 \backslash \underline{M}^{\perp}\underline{\omega}^{15}/\underline{c}$
Add and carry logical	ACL	$y \leftarrow \perp(\underline{a}^2/\underline{u}) + \perp \underline{M}^{\perp}\underline{\omega}^{15}/\underline{c}$
		$\perp \underline{a}^2/\underline{u} \leftarrow (2^{36} \mid y) + (y > 2^{36})$

Instructions in IBM 7090

Figure 2

Ordered Trees

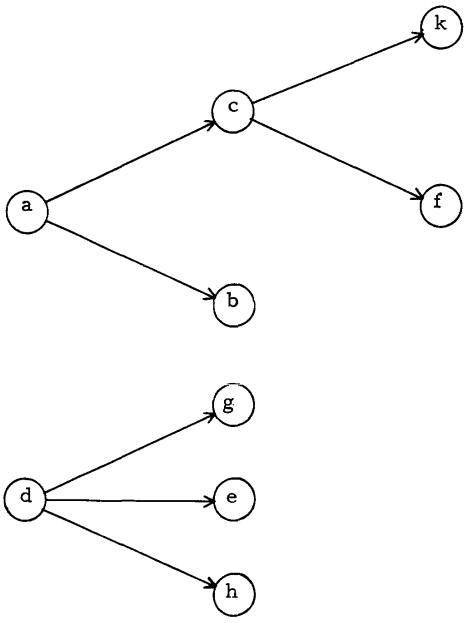
A tree can be represented graphically as in Figure 3(a) and, since it is a special case of a directed graph, it can also be represented by a node vector $\underline{n}$ and connection matrix $\underline{C}$ as in Figure 3(b).

Defining an ordered tree as a tree in which the set of branches emanating from each node has a specified ordering, a simple indexing system for ordered trees can be defined in the manner indicated by Figure 4(a). The dimension of the vector index $\underline{i}$ assigned to each node is equal to the number of the level on which it occurs. If T is an ordered tree, then $T_{\underline{i}}$ will denote the subtree rooted in the node with index $\underline{i}$. Thus $T_{(1,1)}$ is the subtree enclosed in broken lines in Figure 4(a). Moreover, $T^{\underline{i}}$ will denote the (unique) path vector comprising all nodes from the root to node $\underline{i}$. Thus $T^{(1,2)} = (a, b)$. The maximum dimension occurring among the index vectors is called the height of the tree and is denoted by $\nu(T)$.

The index vector notation proves very convenient in describing algorithms involving tree operands. Moreover, it provides a simple formalization of the two important representations of a tree, the Lukasiewicz representation and the level-by-level list.

If each index vector is augmented by sufficient null components (denoted by 0) to bring all to a common dimension equal to the height of the tree, then they may all be arrayed in an index matrix $\underline{I}$ as in Figure 4(b). If, as in Figure 4(b), the node vector $\underline{n}$ is defined such that $\underline{n}_j$ is the value of the node indicated by (the significant part of) the index vector $\underline{I}^j$, then the matrix obtained by appending $\underline{n}$ to $\underline{I}$ provides an unambiguous representation of the tree T. It is convenient to also append the degree vector $\underline{d}$ such that $\underline{d}_j$ is the degree of (i.e., the number of branches emanating from) node $\underline{I}^j$. The resulting matrix of $\nu(T) + 2$ columns is called a full list matrix of T.

Any matrix obtained by interchanging rows of a full list matrix is also a full list matrix and is again an unequivocal representation of the tree. If, as in Figure 4(c), the index vectors are right justified and arranged in increasing order on their value as integers (in a positional number system), the resulting list matrix is called a full right list matrix and is denoted by $\rceil T$. If, as in Figure 4(b), the index vectors are left justified and arranged in increasing order as fractions, the list matrix is called a full left list matrix and is denoted by $\lceil T$.



(a)

$\underline{n}$ = (a, c, f, k, e, d, h, g, b,)

$$\underline{C} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)

FIGURE 3

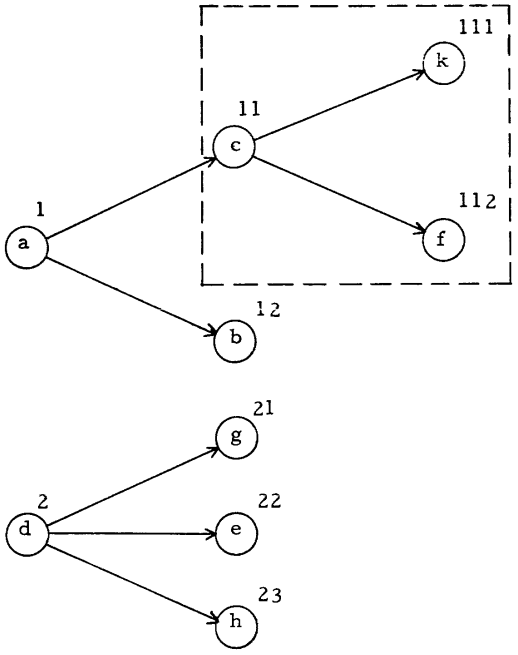
The right list groups nodes by levels, and the left list by subtrees. Moreover, in both cases the degree and node vector together (that is, $\underline{a}^2/(\rceil T)$ or $\underline{a}^2/(\lceil T)$) can be shown to provide an unequivocal representation of the tree without the index matrix. If, as in the case of a tree representing a compound statement involving operators of known degree, the degree vector $\underline{d}$ is a known function of the node vector $\underline{n}$, then the tree may be represented by the node vector $\underline{n}$ alone. In the case of the left list, this leads to the familiar Lukasiewicz* notation for compound statements.

Applications of Tree Notation

The tree notation and the right and left list matrices are useful in many areas**. These include sorting algorithms such as the repeated selection sort[4], the analysis of compound statements and their transformation to optimal form as required in a compiler, and the construction of an optimal variable-length code of the Huffman[5] prefix type. The sorting algorithm proceeds level by level and the right list representation of the tree is therefore most suitable. The analysis of a compound statement proceeds by subtrees and the left list (i.e., Lukasiewicz) form is therefore appropriate. The index vectors of the leaves of any tree clearly form a legitimate Huffman prefix code and the construction of such a code proceeds by combining subtrees in a manner dictated by the frequencies of the characters to be encoded, and therefore employs a left list. However, the characters are finally assigned in order of decreasing frequency to the leaves in right list order, and the index matrix produced must therefore be brought to right list order.

* First introduced by J. Lukasiewicz[2], and first analyzed by Burks et al[3].

** For detailed treatments of the applications mentioned, see Reference 1.



T

(a)

$\underline{d}$	$\underline{n}$	$\underline{I}$		
2	a	1	0	0
2	c	1	1	0
0	k	1	1	1
0	f	1	1	2
0	b	1	2	0
3	d	2	0	0
0	g	2	1	0
0	e	2	2	0
0	h	2	3	0

$\lceil T$

(b)

$\underline{d}'$	$\underline{n}'$	$\underline{I}'$		
2	a	0	0	1
3	d	0	0	2
2	c	0	1	1
0	b	0	1	2
0	g	0	2	1
0	e	0	2	2
0	h	0	2	3
0	k	1	1	1
0	f	1	1	2

$\rceil T$

(c)

FIGURE 4

REFERENCES

1. Iverson, K. E., "A Programming Language", Wiley, 1962.

2. Lukasiewicz, Jan, Aristotle's Syllogistic From the Standpoint of Modern Formal Logic, Clarendon Press, Oxford, 1951, p. 78

3. Burks, A. W., D. W. Warren, and J. B. Wright, An Analysis of a Logical Machine Using Parenthesis-free Notation", Mathematical Tables and Other Aids to Computation, Vol. VIII (1954), pp 53 - 57

4. Friend, E. H., "Sorting on Electronic Computer Systems", J. Assoc. Comp. Mach. 3, pp 134 - 168 (March 1956)

5. Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes", Proc. IRE, Vol. 40 (1952) pp 1098 - 1101