R. W. Murphy

A Positive-Integer Arithmetic for Data Processing

Abstract: It is hypothesized that positive numbers suffice for the expression of quantities in accounting. New arithmetic operations are devised that yield non-negative results in computation, and the applicability of these operations to data processing is studied. These operations permit a wide variety of functions to be computed with fewer and less complex steps and imply the feasibility of constructing less complex data-processing machines.

Introduction

This report describes some exploratory work on an arithmetic intended to be more useful for accounting and data processing. The report deals with the question, "What would be the effects of using only positive numbers in accounting?" Effects that would interest us especially are reductions in the number or in the complexity of steps of computation and, possibly, a closer correlation between data processing and that which it represents.

Certain operations are performed by accountants and bookkeepers in order to avoid having to count the objects of their interest. These are the common arithmetic operations of addition, subtraction, multiplication and division. The unrestricted use of subtraction gives rise to negative numbers and necessitates the further operations of testing for negative balances.

It can be contended that negative numbers do not occur "naturally" in accounting: that quantities can be expressed only by means of the counting numbers and zero. If "credit" is taken as a positive quantity of something, then "debit" ought to be thought of as a positive quantity of something else. The practice of maintaining separate totals of debit and credit is an instance of this concept.

In postulating a positive-integer arithmetic for bookkeeping,* new operations must be devised to work within this number system. We will employ an operation, called *diminish,* instead of subtract. In terms of diminish, we can define *add, take the lesser,* and *take the greater.* The body of this report discusses some properties and uses of these operations.

*The author has not been able to discover any published work dealing with a non-negative system of arithmetic in data processing. Some of the computation techniques described in this paper, or variations such as those using absolute-value operations, are known to programmers, but, to the author's knowledge, have not been developed extensively or reported.

As this report shows, a wide variety of functions can be computed with the diminish operation. In particular, the kind of function that is ordinarily described by a number of conditional statements (e.g.—if x is positive, multiply by r) becomes a single, non-conditional arithmetic expression. This observation has some important implications with respect to the design and use of data-processing machines:

1. Data-processing-machine programs are straight-line rather than branched. This fact implies that:
   a Fewer selectors are required for machines with control panels.
   b Stored programs can be retained in sequential-access memories rather than in random-access memories because program control transfers are not required.
   c The construction and assembly of stored programs is simplified.
2. The total number of program steps is reduced, thus conserving space.
3. The number of program steps executed in performing a computation is sometimes reduced, sometimes increased.
4. Equipment for the storage and control of number signs can be eliminated from data-processing machines.

Definition and elementary properties of diminish

Diminish is a primitive recursive function, and has been studied as a topic of mathematical logic.* In that context the operation has not been given a specific name, but is designated by the symbol " $\dot{-}$ ". Because this symbol might be confused with the divide symbol, we prefer to

*See, e.g., S. C. Kleene, *Introduction to Metamathematics,* D. Van Nostrand and Co., New York, 1952, p. 217, Chap. IX.

use "$\ominus$" to stand for the diminish operation, which we define* to be:

$$x \ominus y = x - y \quad \text{if } x \geqslant y$$
$$= 0 \quad \text{if } x < y$$

Diminish, therefore, produces the quantity of objects remaining after as many as y have been removed from x.

Diminish can be associated with other operations:

The sum of x and y:

$$x + y = G \ominus [(G \ominus x) \ominus y]$$
$$= G \ominus [(G \ominus y) \ominus x]$$

G is any number large enough so that making it larger does not alter the value of $x + y$.

The lesser of x and y:

$$x \cap y = x \ominus (x \ominus y) = y \ominus (y \ominus x)$$

The greater of x and y:

$$x \cup y = x + (y \ominus x) = y + (x \ominus y)$$

In the same manner that we use the phrase "the add operation," we will use the phrases "the lesser operation" and "the greater operation."

The four operations, *diminish, add, greater,* and *lesser,* represent directly certain simple data processes. For example, suppose we have two sets of objects, X and Y, and x and y are the counts of the two sets. If we withdraw one of X for each of Y, certain derivative quantities appear:

$x \ominus y \equiv$ number of unmatched X objects

$y \ominus x \equiv$ number of unmatched Y objects

$x \cap y \equiv$ number of matched X or Y objects

Interpretations like these are helpful in working out expressions of more complicated processes because diminish, unlike subtract, does not have the cancellation properties which allow the reduction of long expressions.

Miscellaneous identities

$$x \ominus y \leqslant x \tag{1}$$

$$(x \ominus y) \ominus x = 0 \tag{2}$$

$$(x \ominus y) \ominus (y \ominus x) = x \ominus y \tag{3}$$

$$(x \ominus y) \ominus z = (x \ominus z) \ominus y = x \ominus (y + z) \tag{4}$$

$$(x + y) \ominus z = (x \ominus z) + y \quad \text{if } x \geqslant z \tag{5a}$$

$$= y \ominus (z \ominus x) \quad \text{if } x < z \tag{5b}$$

$$= (x \ominus z) + [y \ominus (z \ominus x)] \tag{5c}$$

$$= [(x \ominus z) + y] \ominus (z \ominus x) \tag{5d}$$

$$x \ominus (z \ominus y) = (x + y) \ominus z \quad \text{if } z \geqslant y \tag{6a}$$

$$= [(x + y) \ominus z] \ominus (y \ominus z) \tag{6b}$$

$$= (x + y) \ominus (y \cup z) \tag{6c}$$

An instance of the use of diminish

The computation of income-tax and FICA deductions in payroll processing illustrates the use of the diminish operation. These are two separate computations which usually are done together in practice.

(1) An income-tax deduction, D, is made only if the gross pay, G_n, exceeds the exemption allowance, E. The tax is computed on the difference between gross pay and exemption.

Written in conventional arithmetic, this relation is:

$$D = r(G_n - E) \quad \text{if } G_n > E \quad (\text{r} \equiv \text{tax rate})$$
$$= 0 \quad \text{if } G_n \leqslant E$$

The diminish operation permits the same function to be written as:

$$D = r(G_n \ominus E)$$

(2) The FICA deduction, F, is based on gross pay if gross earnings-to-date, $\Sigma^n G_i$, are less than \$4200, based on the difference, $4200 - \Sigma^{n-1} G_i$, if gross earnings-to-date go over \$4200 in this pay period; and zero if \$4200 has already been exceeded.

More briefly, these conditional relations are written in conventional arithmetic as:

$$F = sG_n \quad \text{if } \Sigma^n G_i \leqslant 4200$$
$$= s(4200 - \Sigma^{n-1} G_i) \quad \text{if } \Sigma^{n-1} G_i < 4200 < \Sigma^n G_i$$
$$= 0 \quad \text{if } \Sigma^{n-1} G_i > 4200$$

Using diminish and lesser, these relations become:

$$F = s[G_n \cap (4200 \ominus \Sigma^{n-1} G_i)] \text{ or}$$
$$F = s\{G_n \ominus [G_n \ominus (4200 \ominus \Sigma^{n-1} G_i)]\}$$

If we assume an instruction set, such as that of the IBM 705 computer, plus the additional instruction to perform the diminish operation, we can contrast the programs performing these functions in the two ways. It is convenient to define the diminish instruction as, "Diminish the quantity specified by the address part of the diminish instruction by the quantity in the arithmetic register." A comparison of the two types of programming is shown in Program 1.

---

*Dr. R. R. Seeber Jr. of IBM has pointed out in a note that calculators with absolute-value instructions can perform the diminish operation as follows:

$$x \ominus y = \tfrac{1}{2}[(x - y) + |x - y|]$$

Comparable relations hold for $x \cup y$ and $x \cap y$. Thus the procedures for evaluating various functions, worked out in this paper in terms of diminish, can be translated into procedures using absolute-value operations.

Program 1 **Comparison of conventional** *programming* steps with use of "diminish" operation.*

Conventional			*Using Diminish*		
1.1	R ADD	(G_n)	1.1	R ADD	(E)
.2	SUB	(E)	.2	DIM	(G)
.3	TR +	2.1	.3	MPY	(r)
.4	R ADD	(0)	.4	STORE	(D)
.5	TR	2.2			
2.1	MPY	(r)			
.2	STORE	(D)			
3.1	R ADD	(4200)	2.1	R ADD	$(\Sigma^{n-1}G_i)$
.2	SUB	$(\Sigma^{n-1}G_i)$	.2	DIM	(4200)
.3	SUB	(G_n)	.3	DIM	(G_n)
.4	TR +	4.1	.4	DIM	(G_n)
.5	ADD	(G_n)	.5	MPY	(s)
.6	TR +	4.2	.6	STORE	(F)
.7	R ADD	(0)			
.8	TR	4.3			
4.1	R ADD	(G_n)			
.2	MPY	(s)			
.3	STORE	(F)			

*R ADD (G_n) means "reset the arithmetic register to contain the number G_n"
SUB (E) means "subtract from the number in the arithmetic register the number E"
TR 2.2 means "perform next the operation 2.2"
MPY (r) means "multiply the number in the arithmetic register by the number r"
STORE (D) means "put the number in the arithmetic register into storage and refer to it as D"
TR + 4.1 means "perform next the operation 4.1 only if the number in the arithmetic register is positive, otherwise perform the next operation in sequence"

In the instance of Program 1, diminish provides:
(1) A saving of eight out of eighteen instruction spaces;
(2) A saving of an average of three out of thirteen instructions executed;
(3) A straight-line program

Characteristics of the operations greater and lesser

Quite often in data processes, it is necessary to obtain a quantity which is in a certain relation to other quantities; for example, the least of three quantities. The operations, greater and lesser, will be used exclusively in expressions for performing such processes and therefore are of special interest. Moreover, these two operations can be associated directly with the theory of sets. There are a number of useful relations involving these two operations that can be presented, either by proving them from the definitions of greater and lesser in terms of diminish, or by making the association with set theory and using that theory to provide us with ready-made relations. This second way is briefer and will be given here.

The quantities, denoted by $x, y, \ldots$, express the count of certain physical or conceptual objects. The objects can be represented by any tokens, such as points on a plane, for arithmetic purposes. A boundary, drawn around x points, segregates a set of points. The set, designated X, is isomorphic to the quantity x. Another boundary, drawn either *wholly within* or *wholly without* the boundary around x, produces another set, Y, isomorphic to the quantity y. X or Y will be void sets if the corresponding quantities happen to be zero (see Fig. 1).

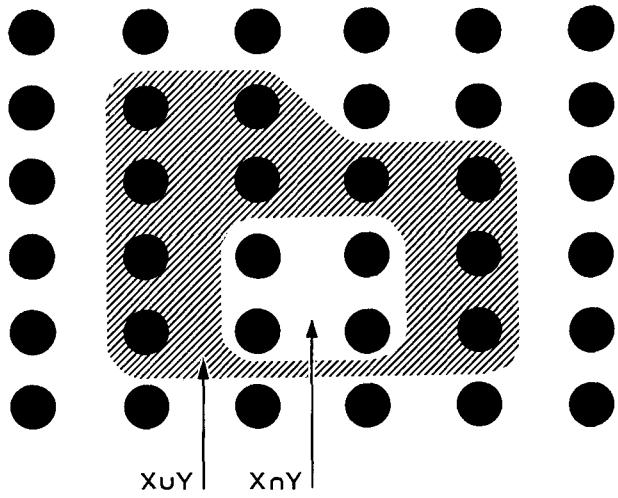

Figure 1 **Graphic representation** *of arithmetic boundaries between sets of points in a given plane.*

The union of two sets, $X \cup Y$, is defined as the set having elements in either X or Y. Therefore the set-theoretic union for the special construction of sets just defined is isomorphic to the value, the larger of the quantities x and y. Similarly, the intersection of two sets, $X \cap Y$, is defined as the set having only the elements common to X and Y. Thus the set-theoretic intersection is isomorphic to the value, the lesser of x and y. These isomorphisms account for the use of the standard symbols $\cup$ (cup) and $\cap$ (cap) for the greater and lesser operations.

Having established these isomorphisms, we can translate certain relations, proved in set theory, into the following:

$$x \cup y = y \cup x$$
$$x \cap y = y \cap x \qquad \text{(commutative relations)}$$

$$x \cup (y \cup z) = (x \cup y) \cup z$$
$$x \cap (y \cap z) = (x \cap y) \cap z \qquad \text{(associative relations)}$$

$$x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$$
$$x \cap (y \cup z) = (x \cap y) \cup (x \cap z) \qquad \text{(distributive relations)}$$

$$x \cup (x \cup y) = x \cup y$$
$$x \cap (x \cap y) = x \cap y$$

$$x \cup (x \cap y) = x$$
$$x \cap (x \cup y) = x$$

The calculability of any ordinal of a set of quantities

Suppose we are given an arbitrarily-ordered set of numbers $\langle a, b, c, \ldots n\rangle$. There will be another set $\langle \alpha, \beta, \gamma, \ldots \nu\rangle$ containing the same numbers but ordered in increasing magnitude.

$\langle a, b, c, \ldots\rangle$ could be numbers stored in memory, in order of memory address. $\langle \alpha, \beta, \gamma, \ldots\rangle$ are the ordinals of this set: i.e., α is least, β is the next, or second least, etc.

Every ordinal, $\alpha, \beta, \gamma, \ldots \lambda, \mu, \nu$, from either end is calculable from $\langle a, b, c, \ldots\rangle$ using only the operations greater ($\cup$) and lesser ($\cap$). This implies that the ordinal is also calculable with only diminish. Using greater and lesser, the calculability is shown in the following manner:

(1) The least and the greatest quantities, α and ν, are:

$$\alpha = a \cap b \cap c \cap \ldots$$

$$\nu = a \cup b \cup c \cup \ldots$$

since $\cup$ and $\cap$ are transitive.

(2) Consider the n different subsets of $\langle a, b, c, \ldots\rangle$, each containing $n-1$ elements. Each subset will contain all but one of $a, b, c, \ldots$, and, by (1) will have a calculable least element $\alpha'_1\ \alpha'_2\ \alpha'_3\ \alpha'_n$. One subset will not contain α, and will therefore contain β as a least element. All other subsets will contain α which will be the least element. Therefore, the next to the least element of the original set will be the greatest of the leasts of these subsets:

$$\beta = \alpha'_1 \cup \alpha'_2 \cup \alpha'_3 \cup \cdots \cup \alpha'_n$$

$$= (b \cap c \cap d \cap \ldots) \cup (a \cap c \cap d \cap \ldots)$$

$$\cup (a \cap b \cap d \cap \ldots) \cup \ldots$$

(3) By (2) the next leasts of the same subsets are calculable. The subset not containing α, and the subset not containing β will contain γ as the next least. All other subsets contain both α and β, and β will be the next least in these. The next-next least, γ, of the original set is therefore:

$$\gamma = \beta'_1 \cup \beta'_2 \cup \beta'_3 \cup \cdots \cup \beta'_n.$$

(4) By induction, it can be seen that any ordinal from least on up can be calculated. This is done by taking the subsets, each not containing one of $a, b, c, \ldots$ and therefore not containing one of $\alpha, \beta, \gamma, \ldots$ and calculating the previous ordinal for each, then calculating the greater of these previous ordinals.

(5) In exactly similar fashion, the second greatest, third greatest, etc. are calculable.

As an example, suppose we want the second least of the four quantities a, b, c, d. By the above rule, this will be the greatest of the leasts of the four subsets (abc), (abd), (acd), (bcd):

$$\beta = (a \cap b \cap c) \cup (a \cap b \cap d) \cup (a \cap c \cap d)$$

$$\cup (b \cap c \cap d).$$

Applying the formulas of the preceding section to the first and second parenthetical terms and to the third and fourth:

$$\beta = ((a \cap b) \cap (c \cup d)) \cup ((c \cap d) \cap (a \cup b)).$$

The machine program for evaluating this expression, or others of the type described in this section, would be straight-line, and would not require the use of comparison operations to determine one of several alternate programs.

When dealing with the inner ordinals (second least, next-to-greatest, etc.) there is an important distinction that must be made to avoid misinterpreting the verbal description of an operation. An operation such as "Take the second least of the quantities $a, b, c, \ldots n$," means that if all the quantities are rearranged such that

$$\alpha \leqslant \beta \leqslant \gamma \leqslant \ldots \leqslant \nu,$$

the operation gives β as a result, regardless if $\alpha = \beta$, $\beta = \gamma, \ldots$

A different function might be defined as "Take the second least, not counting equals." That is, if we have

$$\alpha = \beta = \ldots = \delta < \varepsilon = \xi = \ldots \iota < \kappa = \ldots$$

the result of this operation is a quantity between the first and second $<$ symbols. Such a function can not be constructed with the diminish operation or its derivatives for reasons that will be considered in connection with the continuity properties of expressions using diminish and its derivative operations.

Functions constructable with diminish

Diminish is being discussed as it applies to accounting arithmetic. In this application, the counting numbers serve as measures of quantity. We wish now to find out what functions can be formulated as expressions using diminish. This can be done most easily by permitting the quantities to take on all real positive values, rather than just integral values, and analyzing the continuity properties of these expressions. Later on, it will be shown that we have more variety in the functions that can be calculated by machine if we restrict quantities to integers, or at least to numbers that can change only in discrete steps.

We use the conventional operation of subtract only as a convenient means of defining diminish. If we set up the function

$$f(x, y) = x \ominus y = x - y \qquad \text{if } x \geqslant y$$

$$= 0 \qquad \text{if } x < y$$

$f(x, y)$ is single-valued and continuous for all values

$x < y$, and $x > y$, because both $x - y$ and 0 are continuous. If $x = y$, $f(x, y) = 0$; moreover, this is the limit of $f(x, y)$ as x and y approach each other from either direction. Thus $f(x, y)$ is a single-valued and continuous function of both variables.

Terms like $(x \ominus y)$ will appear in more complex expressions. As all the variables in such expressions change, each term of the form $(x \ominus y)$ will vary continuously, and thus can be considered to be a single dependent variable. That is, in an expression like

$$g(x, y, z) = (x \ominus y) \ominus z,$$

where x, y, and z may be varied simultaneously, we can treat $(x \ominus y)$ as a single variable, recognizing that $g(x, y, z)$ is a continuous function of all variables.

Assuming that any finite assortment of variables, diminish operation, and parentheses in an expression can be decomposed by stages into terms like $(x \ominus y)$, any such expression must be a continuous function of all the variables. Because the operations add, greater, and lesser are all definable in terms of diminish, these operations also provide continuous functions.

We may be given any function in the form of a table, or graph, or statement in words. In order to determine if the stated form of the function can be replaced with an expression of terms like those just discussed, it is necessary first of all, to determine if the function is continuous. If it is not, then some other method of computation must be used. Two methods will be considered later, one using the additional operation of multiply, the other being an algorithm for computing functions given as tables.

If the stated function is continuous, however, we can conjecture that the function can be replaced by an expression using the diminish operation. In the next section, the continuity properties of certain functions of data processing will be examined.

Configuration functions

An important class of functions includes such relations as:

$f_1(x, y, z)$ = the second least of x, y, z. (1)

$f_2(x, y, z)$ = the lesser of x and y which is greater than z, or else z. (2)

These are representative of a class which we call *configuration functions*, and characterize as follows:

(1) They are single-valued functions of n variables.
(2) The function value is the same as the value of one of the variables.
(3) The function value depends on the relative values of the n variables.

The nature of (1) and (2) suggests that the functional statement, in words, might be replaced by an expression employing the operations greater and lesser. However, this possibility does not exist if it can be determined that the function is not continuous. The process of finding out if a given function is continuous is the primary concern of this section.

The notion or concept "relative values of variables" is fundamental to the definition of configuration functions and requires elaboration before proceeding to the continuity properties of these functions. What will be done is to classify the domains of the variables into sub-domains, in each of which the relative values of the variables are different. Therefore, by definition, a configuration function assumes a single value which is also the value of one or more of the variables, but which of the variables it is, depends on the sub-domain where the function is evaluated.

The classification of the domains will be done in two stages, the first stage resulting in sub-domains called *configurations* which are characterized by such relations as

$$x \leqslant y \leqslant z \leqslant \ldots$$

The second stage takes apart the configurations and results in the *sub-configurations*

$$x < y < z < \ldots$$
$$x = y < z < \ldots$$
$$x < y = z < \ldots$$
$$x = y = z < \ldots \quad \text{etc.}$$

In general, it is necessary to evaluate a configuration function for each sub-configuration of the variables in order that there may not be an ambiguity as to the value of the function in the sub-domains where two or more variables assume equal values. However, it is only with the discontinuous functions that an ambiguity can occur, as will be seen.

A function of n independent variables is an explicit expression of certain variable symbols . . . , x, y, z. To evaluate the function, a schedule of substitutions is needed:

For x substitute 5
For y substitute 4 (3)
For z substitute 3

The numbers (5, 4, 3) can be put into the relation

$$3 \leqslant 4 \leqslant 5 \quad (4)$$

by the method described previously. Now, if we interpret the substitution relations (3) as reflexive, we have from (4)

$$z \leqslant y \leqslant x \quad (5)$$

We define the relation (5) as a *configuration* of the variables x, y, z. The configuration defines a closed region, or domain, of the variables x, y, z, for which the relative

values of the variables are fixed. It is obvious that there are portions of the total domain of x, y, z that belong to more than one configuration. For example, if

$$\langle x, y, z\rangle = \langle 5, 3, 3\rangle,$$

then both of the following are true:

$$z \leqslant y \leqslant x$$

$$y \leqslant z \leqslant x.$$

The generic configuration

$$\xi \leqslant \eta \leqslant \zeta \leqslant \ldots \tag{6}$$

is used to indicate any permutation of the variables $x, y, z, \ldots$ in the configuration relation. Since all real numbers have the properties:

(1) Either $a \leqslant b$ or $b \leqslant a$, or both; and

(2) If $a \leqslant b$ and $b \leqslant c$, then $a \leqslant c$,

we require all permutations of $x, y, z, \ldots$ in (6) to cover the domains of $x, y, z, \ldots$.

Within the configuration there are 2^{n-1} conditions, termed *sub-configurations,* that can be established by appropriate choices of values for the variables. There are $n-1$ symbols $\leqslant$, and each may have either of two meanings, $=$ or $<$. The sub-configuration

$$\xi < \eta < \zeta < \ldots \tag{7}$$

belongs to one and only one configuration (6), while of the sub-configurations,

$$\begin{gathered}\xi = \eta < \zeta < \ldots \\ \xi < \eta = \zeta < \ldots \\ \xi = \eta = \zeta < \ldots, \text{ etc.,}\end{gathered} \tag{8}$$

each belongs to more than one configuration.

Sets of continuous functions of a parameter t can be used to provide values for the variables, $\xi, \eta, \zeta, \ldots$ Thus, if $f(x, y, z)$ is continuous everywhere, it must be continuous along the curve described by the parametric functions. In particular, a typical set of parametric functions is in the relation:

$$\ldots\alpha < [\beta + (\varepsilon \ominus \beta)\, t] < [\gamma + (\varepsilon \ominus \gamma)\, t] < \ldots < \varepsilon < \ldots \tag{9}$$

and gives values in the sub-configuration

$$\xi < \eta < \zeta < \ldots < \omega < \ldots$$

for $t < 1$, and in the sub-configuration

$$\xi < \eta = \zeta = \ldots = \omega < \ldots$$

when $t = 1$. The existence of (9) thus proves that, if $f(x, y, z, \ldots)$ is continuous, it is continuous within a configuration.

This theorem has the important implication that a continuous configuration function can be described, in full, by listing, for each configuration, the variable whose value is taken by the function in that configuration. In other words, it is unnecessary to consider conditions of equality if the function is continuous.

It is also true that if a function is discontinuous at some point, it must be discontinuous within at least one configuration. This is because every point of the domains of $x, y, z, \ldots$ has all its neighboring points in the same configuration, although the neighboring points may themselves belong to more than one configuration.

In testing the continuity of specific configuration functions, one way of proceeding is to list all $n!$ configurations, and for each configuration, all 2^{n-1} sub-configurations. The function in question is then analyzed, and the variable whose value is assumed by the function is put in correspondence with the sub-configuration. If the same variable appears in each sub-configuration of each configuration, the function is continuous. For example, the two examples given earlier are put in this form in Tables 1 and 2.

Examination of the tabulation for $f_1(x, y, z)$ shows that the same variable appears throughout each configuration and differs only in different configurations. On the other hand, $f_2(x, y, z)$ has the value of y in the sub-configuration $z < y < x$, and x in the sub-configuration $z = y < x$. Thus f_1 is continuous while f_2 is not.

This method of analysis is interesting in that it suggests a way of defining continuity, or rather a property analogous to continuity, for functions whose variables are discrete rather than continuous. That is, we might say that a configuration function is "connected," if the variable

Table 1 **Function f_1 (x, y, z) $\equiv$ the second least of x, y, z.**

	$\xi < \eta < \zeta$	$\xi = \eta < \zeta$	$\xi < \eta = \zeta$	$\xi = \eta = \zeta$
$x \leqslant y \leqslant z$	y	$x = y$	$y = z$	$x = y = z$
$y \leqslant x \leqslant z$	x	$x = y$	$x = z$	$x = y = z$
$y \leqslant z \leqslant x$	z	$y = z$	$x = z$	$x = y = z$
$z \leqslant y \leqslant x$	y	$y = z$	$x = y$	$x = y = z$
$z \leqslant x \leqslant y$	x	$x = z$	$x = y$	$x = y = z$
$x \leqslant z \leqslant y$	z	$x = z$	$y = z$	$x = y = z$

Table 2 **Function f_2 (x, y, z) $=$ the lesser of x and y which is greater than z, or else z.**

	$\xi < \eta < \zeta$	$\xi = \eta < \zeta$	$\xi < \eta = \zeta$	$\xi = \eta = \zeta$
$x \leqslant y \leqslant z$	z	z	$y = z$	$x = y = z$
$y \leqslant x \leqslant z$	z	z	$x = z$	$x = y = z$
$y \leqslant z \leqslant x$	x	x	$x = z$	$x = y = z$
$z \leqslant y \leqslant x$	y	x	$x = y$	$x = y = z$
$z \leqslant x \leqslant y$	x	y	$x = y$	$x = y = z$
$x \leqslant z \leqslant y$	y	y	$y = z$	$x = y = z$

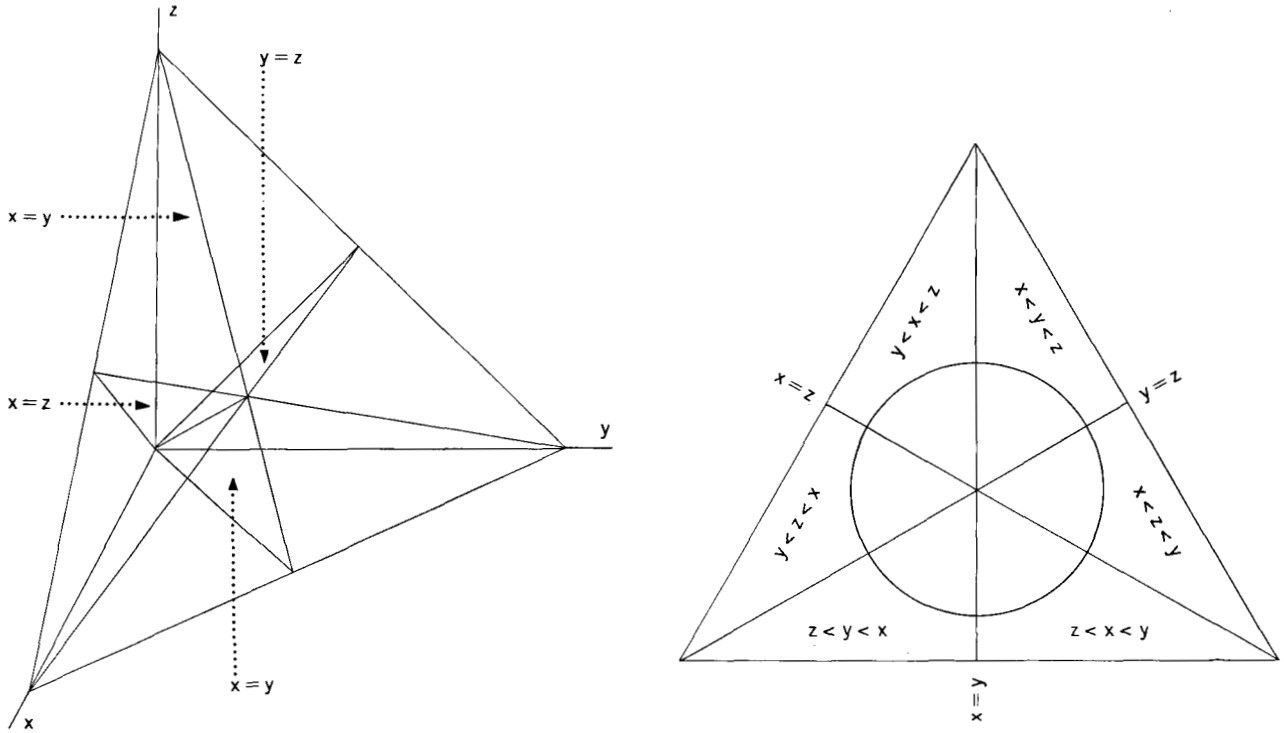

Figure 2 **Values of three independent variables** *plotted as a point in three-dimensional space.*

whose value is assumed is the same for every sub-configuration of a configuration. This concept might be extended to a larger class of functions by letting the variable be dependent, in the sense that a dependent variable is an expression using only the elementary operations. However, this conjecture is presented as interesting, but is apart from the major objective of exploring the properties of diminish and its derivative operations.

Instead of going through the tedious process of evaluating a configuration function for every sub-configuration, it is easier to discover a lack of continuity by using some test function. The test function provides a set of n values as a function of a parameter t, each value being applied to one of the n variables of the configuration function being tested. Each value of the set is a continuous function of t, and as t is varied, the test function places the variables in every configuration. In the next section we will describe a suitable test function graphically.

A test function for determining continuity configuration functions

Any set of values of three independent variables can be plotted as a point in three-dimensional space. The positive octant, to which the values of the variables are restricted, is divided into six regions by the planes $x = y$, $x = z$, $y = z$, as shown in the left diagram of Fig. 2. Each region corresponds to a configuration of the variables.

Looking down into the octant along the line $x = y = z$, the planes of equality would appear as lines at 60° to one another. This line of view establishes a plane of projection on which we can trace out, in projection, the path described by any variation of the three independent variables. The right diagram of Fig. 2 shows this projection of the planes of equality and of an arbitrary path (the circle) described by the set of the three independent variables.

We can put a two-dimensional figure, which shows the variation of the individual variables, on the above projection plane, using angular position along the projected path as a parameter. In Fig. 3 the radial distance from the intersection $x = y = z$ is the value of the individual variable.

It is evident that the only situation which can not be plotted is $x = y = z \neq 0$. Since we are concerned especially with the lack of continuity of functions, any arbitrary path over which the function is discontinuous is sufficient to show the discontinuity of the function. Therefore, we can choose a path not including the condition $x = y = z$ and avoid the difficulty in representation.

The configuration functions, by definition, will consist of segments of the above figure; that is, if we choose one line segment in each sector, we have a representation of a configuration function. It is evident that most of $3^{3!} = 729$ combinations of segments of three variables will be discontinuous, in that they contain a jump, for example, from x to z at the plane $x = y$.

The number of continuous paths in this figure can be counted, and gives us an upper limit to the configuration

functions that might be written as expressions using diminish. This number is 18 in the case of three variables, and, in fact, these can all be written as expressions using only the operations greater and lesser, as shown in Table 3. The method of counting continuous paths is to establish a starting point on some line segment, then pass to the next sector and count the number of ways of getting to each line segment from the previous sector in a continuous manner. This process is continued around, and the number of ways that provide a return to the starting point is the number of continuous configuration functions containing the starting line segment. The numbers on the above diagram are the different ways in which each segment can be traversed from the indicated starting point. The total number of continuous configuration functions is obtained by counting paths for each line segment in the starting sector.

Table 3 **Continuous configuration functions** *of three variables.*

	xyz	yxz	yzx	zyx	zxy	xzy
$x \cup y$	y	x	x	x	y	y
$x \cup z$	z	z	x	x	x	z
$y \cup z$	z	z	z	y	y	y
$x \cap y$	x	y	y	y	x	x
$x \cap z$	x	x	z	z	z	x
$y \cap z$	y	y	y	z	z	z
$x \cup (y \cap z)$	y	x	x	x	x	z
$x \cap (y \cup z)$	x	x	z	y	x	x
$y \cup (x \cap z)$	y	x	z	y	y	y
$y \cap (x \cup z)$	y	y	y	y	x	z
$z \cup (x \cap y)$	z	z	z	y	x	z
$z \cap (x \cup y)$	y	x	z	z	z	z
$x \cap (x \cup y) = x \cap (x \cup z)$ $= x \cup (x \cap y)$ etc.	x	x	x	x	x	x
$y \cap (x \cup y)$	y	y	y	y	y	y
$z \cap (y \cup z)$	z	z	z	z	z	z
$x \cup y \cup z$	z	z	x	x	y	y
$x \cap y \cap z$	x	y	y	z	z	x
$(x \cup y) \cap (x \cup z) \cap (y \cup z)$	y	x	z	y	x	z

The advantage of this graphical construction is that it provides an easy visual method of testing if a function is continuous. The two functions, given previously as examples, can be presented as in Figs. 4(a) and 4(b).

The figures show that f_1 is continuous, while f_2 is not, and therefore f_2 can not be expressed only in terms of diminish.

The method of testing continuity just described can be extended to any number of variables and to functions which take on values which are derivative from the independent variables. To do this, we simply draw on the same figure the additional variables which the function might assume. In the event that there are more than three independent variables, we assume that the two-dimensional projection plane can also be interpreted as a projection of multi-dimensional space. Figure 5 represents the configuration functions of four variables, and is drawn by superimposing the fourth variable, w, letting it assume all possible configurations with respect to the six configurations of x, y, z. The continuous configurations have been counted on this type of diagram and found to be 6993.

Several observations can be made about these diagrams:

(1) Regardless of the number of variables, a test function exists which passes through every configuration.

(2) Within each configuration, the test function passes through the sub-configuration $\xi < \eta < \zeta < \ldots$ and through only two of the degenerate sub-configurations, each containing one equality. Thus, the test function can be used only with configuration functions in which equality conditions are not singled out.

Multiplication

In addition to the operations so far treated (diminish, add, greater, lesser), most data processing requires the use of multiplication. Multiplication is distributive with respect to diminish and its derivative operations:

$$x(y \ominus z) = xy \ominus xz$$

$$x(y + z) = xy + xz$$

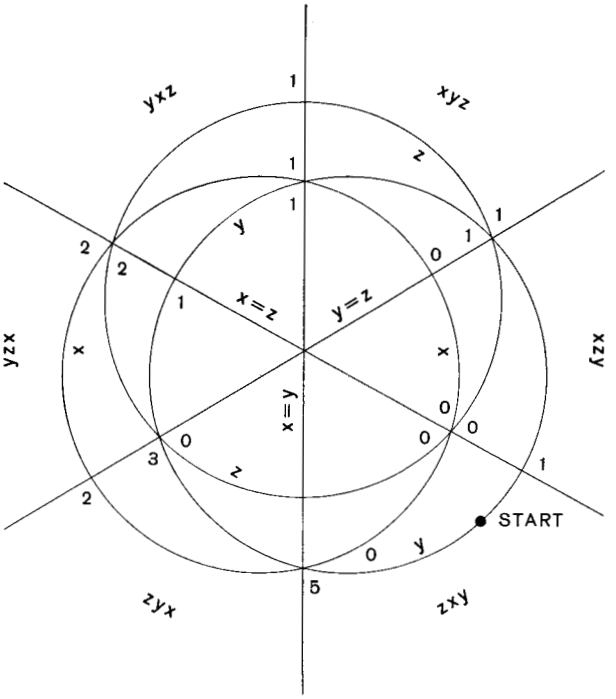

Figure 3 **Two-dimensional plot** *on the projection plane of the right diagram of Fig. 2 showing variation of individual variables.*

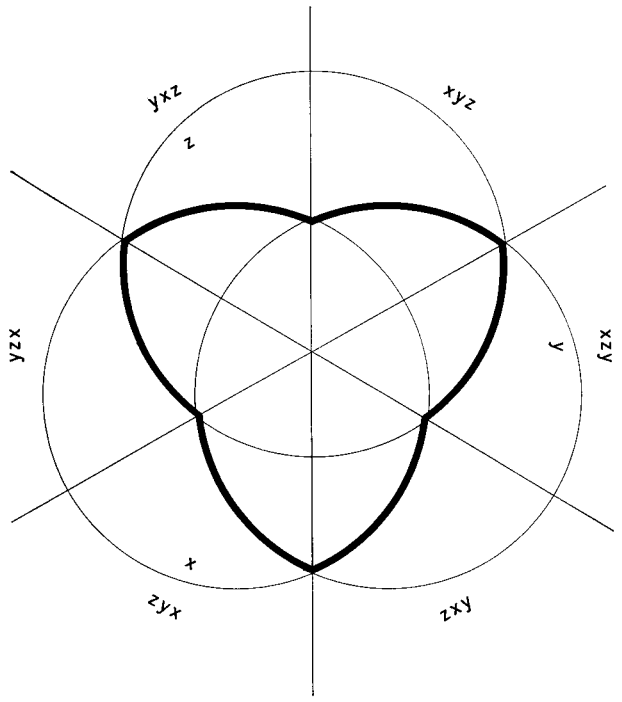

Figure 4(a) **Function f_1, the second least of x, y, z**

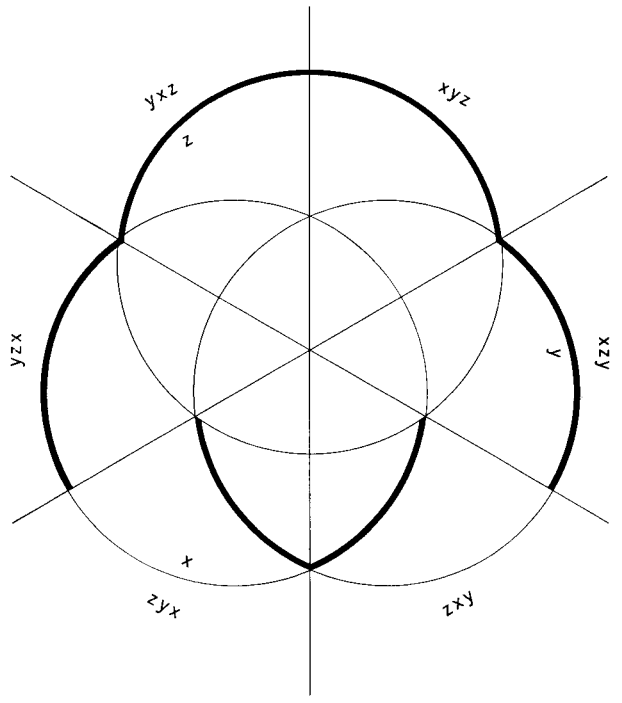

Figure 4(b) **Function f_2, the lesser of x and y,** *which is greater than z, or else z. The figures show that f_1 is continuous, while f_2 is not.*

$$x(y \cap z) = xy \cap xz$$

$$x(y \cup z) = xy \cup xz.$$

The operation of multiplication, along with diminish, permits expressions to be written which represent functions containing discontinuously-varying rates. For example, a public-utility rate schedule is:

\$1.90 for the first 24 kwh or less

5.3¢/kwh for the next 120 kwh

3.7¢/kwh for the next 136 kwh

2.2¢/kwh for more than 280 kwh.

This function is represented by the expressions

$$f(x) = 1.900 + .053[(x \ominus 24) \ominus (x \ominus 144)] + .037[(x \ominus 144) \ominus (x \ominus 280)] + .022(x \ominus 280)$$

$$= 1.900 + \{.053x \ominus [1.272 + .016(x \ominus 144) + .015(x \ominus 280)]\}.$$

The single-address stored programs, compared on the same basis as the example titled "Program 1" are shown in Program 2.

Program 2 **Comparison of single-address stored programs.**

Conventional			*Using Diminish*		
1.1	R ADD	(x)	1.1	R ADD	(x)
.2	SUB	(24)	.2	MPY	(.053)
.3	TR +	2.1	.3	STORE	99.1
.4	R ADD	(1.900)			
.5	TR	4.3			
2.1	SUB	(120)	2.1	R ADD	(280)
.2	TR +	3.1	.2	DIM	(x)
.3	ADD	(120)	.3	MPY	(.015)
.4	MPY	(.053)	.4	STORE	99.2
.5	ADD	(1.900)			
.6	TR	4.3			
3.1	SUB	(136)	3.1	R ADD	(144)
.2	TR +	4.1	.2	DIM	(x)
.3	ADD	(136)	.3	MPY	(.016)
.4	MPY	(.037)	.4	ADD	99.2
.5	ADD	(8.260)	.5	ADD	(1.272)
.6	TR	4.3	.6	DIM	99.1
4.1	MPY	(.022)	.7	ADD	(1.900)
.2	ADD	(13.292)			

In the instance of Program 2, diminish provides:
(1) A saving of five out of nineteen instruction spaces;
(2) A loss of approximately four in additional instructions executed;
(3) A straight-line program.

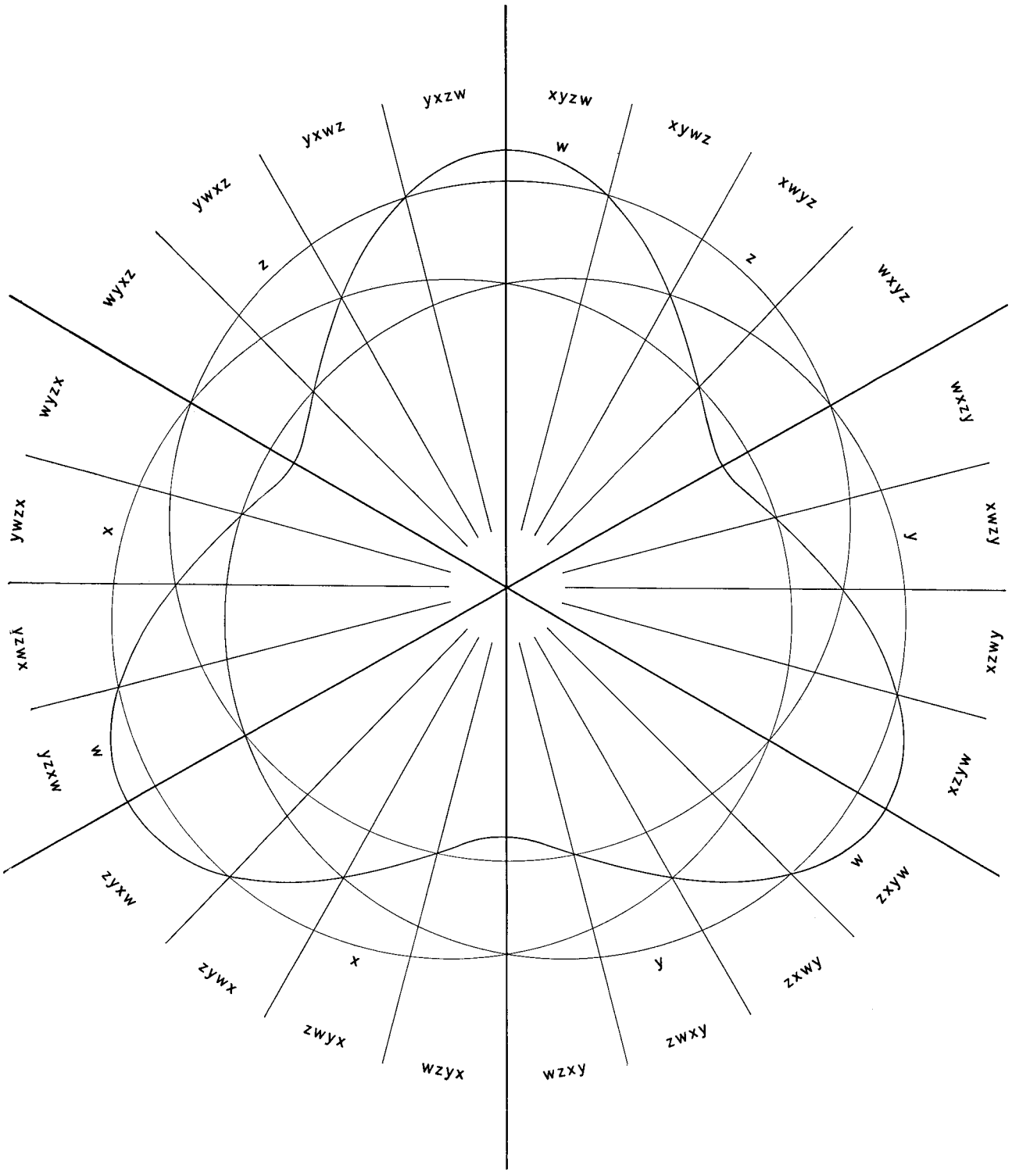

Figure 5 **Configuration functions of four variables** *The fourth variable, w, is superimposed, and assumes all possible configurations with respect to the configurations of x, y, z.*

The simulation of discontinuous functions in digital computation

Diminish, by itself, only permits the construction of continuous functions, or functions that would be continuous if each quantity could assume any real positive value. The use of multiplication makes possible expressions which simulate discontinuous functions in digital data-processing machines. Consider the function

$$f(x) = 1 \ominus (a \ominus x) = 0 \quad \text{if } 1 + x \leqslant a$$

$$= 1 \quad \text{if } x > a\,.$$

If x can vary continuously, the graph of $f(x)$ consists of the two line segments $f = 0$ and $f = 1$ joined by a segment $(x + 1) \ominus a$. To a digital machine computing this function, the joining segment will not be accessible when the quantities are scaled such that the incremental change in x is unity, so $f(x)$ can be considered to simulate the unit step function.

The function

$$g(x) = 1 \ominus [(a \ominus x) + (x \ominus b)]$$

has the value 1 only if $a \leqslant x \leqslant b$. A more general function, $h(x)$, can be set up to be 1 only if any number of conditions are met:

$$h(x) = 1 \cap (w \ominus x) \cap (y \ominus z) \cap \ldots .$$

$$= 1 \text{ if } w \leqslant x \text{ and } y \leqslant z, \text{ and } \ldots .$$

$$= 0 \text{ otherwise.}$$

Having a function which is unity only when a prescribed set of order relations are met, and zero otherwise, it is possible to simulate, on a digital computer, any function with a finite number of discontinuities. This is done by multiplying one of the above types of unity functions by the appropriate value, and taking the sum of a number of such products.

Calculation of the ordinal number of a number in a set

Previously it was shown that the first, second, ... nth, of a set of numbers could be calculated. This calculation might be described as: "Given the ordinal (first, second, etc.), which number of the set is it?" The preceding section now provides a method for the inverse operation: "Given a number which is one of a set, which ordinal is it?"

If the original set is $\langle x_1, x_2, x_3, \ldots \rangle$ which contains n distinct integers, there will be a corresponding set $\langle \alpha, \beta, \gamma, \ldots \rangle$ in which

$$\alpha < \beta < \gamma < \ldots$$

To the set $\langle \alpha, \beta, \gamma, \ldots \rangle$ there corresponds a third set $\langle 1, 2, 3, \ldots \rangle$ which we call the ordinal number set. Any number, x, among the original numbers, $x_1, x_2, x_3, \ldots$ is the same as one of $\alpha, \beta, \gamma, \ldots$ and therefore corresponds to some i of $1, 2, 3, \ldots$ The correspondence is unique provided that all the x_i are different.

Let x be the number in the set $\langle x_j \rangle$ whose ordinal, i, is to be found. Then

$$i = \sum_{j=1}^{n} [1 \ominus (x_j \ominus x)],$$

because for each $x_k < x$,

$$1 \ominus (x_k \ominus x) = 1$$

while for the $x_1 = x$,

$$1 \ominus (x_1 \ominus x) = 1$$

and for each $x_m > x$,

$$1 \ominus (x_m \ominus x) = 0.$$

The numbers in the set $\langle x_i \rangle$ need not be integers, provided that there is some number, s, such that

$$x_i s = \text{integer.}$$

If this is so, then

$$i = \sum_{j=1}^{n} [1 \ominus s\,(x_j \ominus x)\,].$$

As an additional point, if equalities are permitted among the x_j, there arises the question as to what is meant by the ordinal number of x if $x = x_k$ where $\{x_k\}$ is a subset of the original set of numbers $\langle x_j \rangle$.

We can define the following:

$$i_{\max} = \sum_{j=1}^{n} [1 \ominus (x_j \ominus x)]$$

$$i_{\min} = n + 1 \ominus \sum_{j=1}^{n} [1 \ominus (x \ominus x_j)],$$

which amounts to defining the ordinal of a number in terms of an upper and lower bound.

It is interesting to note that computing the number whose ordinal is given is performed by expressions which are continuous, while the inverse process of finding the ordinal number of a given number is performed by simulating a discontinuous process.

Computation of tabular functions

The basic problem of computation might be described in this way:

(1) Given once, a statement of a functional relationship, and,

(2) Given repeatedly, various numbers to apply in the function, determine, repeatedly, a new number in accordance with both (1) and (2).

Up to now, we have been treating statements which employ such words as "the lesser of," "if x is less than y" to express the functional relationship. We have shown that there exist mathematical expressions which are equivalent functional relationships, and that these expressions are translatable into very simple, unconditional steps of computation.

Perhaps the most specific way of describing a functional relationship is to use a succession of statements:

If $x_{i-1} < x \leqslant x_i$, take $f_i (i = 2, 3, \ldots n)$,

where x_{i-1}, x_i, f_i are all numbers, given once as the functional relationship, while x is one of the numbers given on repeated occasions. The shorthand way of representing this functional relationship is by means of a table:

$x =$	x_1	x_2	$\ldots$	x_i	$\ldots$	x_n
$f =$	f_1	f_2	$\ldots$	f_i	$\ldots$	f_n

The table is arranged so that $x_i < x_{i+1}$, but f need not be a monotonic function of x.

There are several ways in which a computer can get f_i, if given x_i using well-known table-look-up procedures. The purpose of this section is to describe another method for computing tabular functions. Basically, the method consists of two stages: converting the tabular entries to coefficients, and performing an algorithm, which uses the coefficients to build up the value $f(x)$. The first stage is, of course, a one-time operation: once determined, the coefficients are available for any future determination of $f(x)$. The algorithm, which is performed for each value of the argument x, will be described first, in order to show what coefficients are obtained.

We first define an adjoined number $x_i|a_i$ to be the representations of x_i and a_i placed side by side and manipulated as a single number. For example, if $x_i = 212$ and $a_i = 1907$, the adjoined number $x_i|a_i = 2121907$. If the radix is r, and the number of digits of a_i is s, then

$$x_i|a_i = x_i r^s + a_i$$

Operations and calculations can be performed on adjoined numbers, just as on ordinary numbers. Thus

$$x_i|a_i \ominus x|0 = x_i \ominus x|a_i \qquad \text{if } x_i \geqslant x$$

$$= \quad 0|0 \qquad \text{if } x_i < x$$

The purpose of this maneuver is to use the diminish operation to wipe out a_i if $x_i < x$, but leave a_i available if $x_i \geqslant x$.

With this definition as a basis, we can now set up a sequence of functions, P_i, which will be the procedure for computing tabular functions. The functions are shown with the value which they assume under the indicated conditions:

	$x \leqslant x_1$	$x \leqslant x_2$	$x \leqslant x_3$
$P_1 = (x_1\|a_1 \ominus x\|0) + x\|0$	$x_1\|a_1$	$x\|0$	$x\|0$
$P_2 = (x_2\|a_2 \ominus P_1) + x\|0$	$x + x_2 \ominus x_1\|a_2 \ominus a_1$	$x_2\|a_2$	$x\|0$
$P_3 = (x_3\|a_3 \ominus P_2) + x\|0$	$x_3 \ominus x_2 + x_1\|a_3 \ominus a_2 + a_1$	$x + x_3 \ominus x_2\|a_3 \ominus a_2$	$x_3\|a_3$

where $a_1 < a_2 < a_3 < \ldots < a_n$.

This process is carried on for n steps (until x_n is reached), resulting in a quantity $x^1|g_n$. Disregarding the x^1 part, which can be readily removed in a machine, we have for g_n:

If $x \leqslant x_1$,
$g_n{}^1 = a_n \ominus (\ldots \ominus (a_4 \ominus (a_3 \ominus (a_2 \ominus a_1))) \ldots)$.

If $x \leqslant x_2$, $\quad g_n{}^2 = a_n \ominus (\ldots \ominus (a_4 \ominus (a_3 \ominus a_2)) \ldots)$.

If $x \leqslant x_3$, $\quad g_n{}^3 = a_n \ominus (\ldots \ominus (a_4 \ominus a_3) \ldots)$.

If $x > x_n$, $\quad g_n{}^n = a_n$.

If $x \leqslant x_n$, $\quad g_n{}^{n+1} = 0$.

Since

$a_1 < a_2 < a_3 < \ldots < a_n$,

$g_n{}^1 = a_n - \ldots \pm a_4 \pm a_3 \pm a_2 \pm a_1$

$g_n{}^2 = a_n - \ldots \pm a_4 \pm a_3 \pm a_2$

$g_n{}^3 = a_n - \ldots \pm a_4 \pm a_3$

$g_n{}^n = a_n$.

The upper signs are taken if n is even.

Because we get a different value of $g_n{}^i$ for different ranges, $x_{i-1} < x \leqslant x_i$, and because the coefficients a_y are

arbitrary (except for the condition $a_y < a_{y+1}$), we can set ${g_n}^i$ equal to the table entries f_i. There is a preliminary step, however, because

$${g_n}^{n-1} < {g_n}^{n-3} < \ldots < {g_n}^1 < \ldots < {g_n}^{n-2} < g^n.$$

If f_i should conform to this order, we can set $f_i = {g_n}^i$ and solve for the coefficients a_y. Most of the time, f_i will be in different order, but we can set

$$f_i = {g_n}^i = f_i \pmod m.$$

In general, m is conveniently a power of the number radix (10 for decimal numbers), so that after $P_n = x^1|g_n$ is computed, both x^1 and the excess digits of g_n can be removed in one operation.

To summarize, we have a computational procedure, as shown in Program 3.

Program 3 **Computing procedure.**

1.1	R ADD	$(x\|0)$
.2	DIM	$(x_1\|a_1)$
.3	ADD	$(x\|0)$
.4	DIM	$(x_2\|a_2)$
.5	ADD	$(x\|0)$
.6	|	|
|	|	|
|	|	|
|	|	|
|	|	|
1.2n	DIM	$(x_n\|a_n)$
1.2n+1	SET L	....

The procedure indicated in Program 3 leaves in the accumulator the value f_i corresponding to $x_{i-1} < x \leqslant x_i$.

The method, as described, requires two program steps for each entry of the table plus some preparatory steps. In general, it is not as rapid as direct look-up or search methods of obtaining table values, but will require smaller storage space, particularly for small tables. A typical example is that of an insurance company which retains in each policy record the month and year to which the premium is paid. Premiums may be paid annually, semi-annually, quarterly, or monthly, the mode of payment being coded with a single digit in the policy record. As each new premium payment is made, the premium-paid-to-date must be revised. In this instance the code is adjoined to the month to which the premium is paid to give a number which will be used as the argument to determine the year-month increment.

Detailed programs have been written, and show that this method, in comparison to conventional table-look-up procedures, resulted in (1) a saving of 10 out of 32 instruction spaces; (2) a loss of approximately 10 in additional program steps executed; and (3) a straight-line program.

Conclusions

Positive-integer arithmetic, with its appropriate operations, provides a basically different method of computation. Conventionally, tests on the signs of numbers are used to determine the later course of computations and are stated explicitly in the computation. In positive-integer arithmetic, the tests are contained implicitly in the fundamental operations, and the course of computation is fixed.

A very wide range of functions can be written as simple expressions. This includes functions which otherwise require verbal descriptions or tables to express, as well as non-analytic functions.

In the application of positive-integer arithmetic to data-processing and computing machines, this study discloses the following:

(1) It is possible to build efficient machines with less equipment than is now used. The reduction in equipment appears in the storage requirements for machine programs; the elimination of equipment for the control of signs; and the use of sequential, instead of random, program storages.

(2) The addition of the diminish operation to existing equipment offers the programmer means of reducing requirements on machine capacity, either in program storage or in control panel functions.

(3) The programmer, in thinking through a complex computation, may find it easier to follow a single line of reasoning rather than to keep in mind the branchings of alternate procedures. Besides this subjective consideration, it appears that automatic programming and program assembly are simpler when only linear procedures occur.

(4) In some types of computations, the disadvantage of this system of arithmetic is that it requires more program steps to be executed and therefore would be a slower machine operation.

Received January 8, 1957