where the symbol $\cap$ denotes an AND operation and the symbol $*$ denotes a multiplication operation.

(2) The result of an OR operation with any number of Boolean variables is the same as the (arithmetic) addition of the x, y, z integer variables after the following test is made:

(a) If the sum is equal to zero, the result is correct;

(b) If the sum is larger than zero, the answer is a 1; i.e.

$$\begin{aligned} x + y + z &= A \cup B \cup C \quad \text{if } x + y + z = 0 \\ x + y + z &= 1 \quad \text{if } (x + y + z) \geqq 1 \end{aligned} \tag{2}$$

where the symbol $\cup$ denotes an OR operation and the symbol $+$ denotes an addition operation.

(3) The result of a NOT operation with a Boolean variable is the same as subtracting an integer variable x from 1; i.e.

$$\bar{A} = (1 - x) \tag{3}$$

because if $A = x = 1$, then $\bar{A} = 1 - 1 = 0$; and if $A = x = 0$, then $\bar{A} = 1 - 0 = 1$.

The FORTRAN program in Figure 1 illustrates the method presented. It simulates the logic of a full-adder as described by the following two Boolean functions:

$$L = K_1K_2 + K_1K_3 + K_2K_3 \tag{4}$$

$$M = \bar{L}(K_1 + K_2 + K_3) + K_1K_2K_3 \tag{5}$$

where K_1, K_2 and K_3 are the two input bits and previous carry to be added, L is the output carry, and M is the output sum. Integer variables were chosen for compatibility with the FORTRAN language.

```
C     EXAMPLE OF BOOLEAN SIMULATION
C     SIMULATION OF A FULL-ADDER
      DIMENSION  K(3)
C     INITIALIZE THE INPUT TRUTH TABLE TO ZERO
      DO 10 I=1,3
   10 K(I)=0
C     DERIVE THE TRUTH TABLE FOR THE SUM M, AND THE CARRY L.
      DO 110 I=1,8
      L = K(1)*K(2)+K(1)*K(3)+K(2)*K(3)
      IF(L) 20,30,20
   20 L = 1
   30 LT=K(1)+K(2)+K(3)
      IF(LT) 40,50,40
   40 LT=1
   50 M = (1-L)*LT + K(1)*K(2)*K(3)
      IF(M) 60,70,60
   60 M = 1
   70 PRINT 75,K(3),K(2),K(1),M,L
C     GENERATE THE NEXT INPUT COMBINATION
      K(3)=K(1)*K(2)*(1-K(3)) + (1-K(1)*K(2))*K(3)
      IF(K(3)) 80,90,80
   80 K(3)=1
   90 K(2) =K(1)*(1-K(2)) + (1-K(1))*K(2)
      IF(K(2)) 100,110,100
  100 K(2)=1
  110 K(1)=(1-K(1))
   75 FORMAT(10X,I3,I3,I3,4X,I3,I3)
      PAUSE
      END
```

FIG. 1

The AND and NOT operations are transformed to multiplication and subtraction operations as described in (1) and (3). The OR operation needs a control IF statement after the arithmetic addition is performed in order to restore the value of the variable to unity. This may be simplified by using a Function subprogram to calculate the result of the OR operation, thus eliminating the need for repetition of the IF statements. It was not done in this example because of the limitation of the FORTRAN compiler in the 1620 Model 1 computer where this program was checked out, and where the use of subprograms is not permitted.

M. MORRIS MANO
California State College at Los Angeles
Los Angeles, California

RECEIVED FEBRUARY, 1964

FURTHER REMARKS ON REDUCING TRUNCATION ERRORS

Recently Jack M. Wolfe [1] proposed the use of cascaded accumulators to evaluate a sum of the form $S = \sum_{i=1}^{N} y_i$ when N is large and all the y's are of roughly the same order of magnitude. His intention was to alleviate the accumulation of rounding or truncation errors which otherwise occurs when S is evaluated in the straightforward way illustrated by the following FORTRAN program.

```
1  S = 0·0
2  DO 4 I = 1, N
3  YI = ···
4  S = S + YI
5  ····
```

The rounding or truncation in statement 4 could contribute to a loss of almost $\log_{10} N$ significant decimals in S. This would be important in those cases where the values of YI computed in statement 3 were correct to nearly full machine precision; otherwise the uncertainty in the YI's would swamp any additional error introduced in statement 4.

Of course, the simplest and fastest way to prevent such figure-loss is to accumulate S to double-precision. For example, in a FORTRAN IV program it would suffice to precede statement 1 above by the TYPE statement DOUBLE PRECISION S. The convenient accessibility of double-precision in many FORTRAN and some ALGOL compilers indicates that double-precision will soon be universally acceptable as a substitute for ingenuity in the solution of numerical problems.

In the meantime, programmers without easy access to double-precision arithmetic may be able to simulate it in the program above by a method far simpler than Wolfe's, provided they are using one of the electronic computers which normalize floating-point sums before rounding or truncating them. Among such machines are, for example, the I.B.M. 704, 709, 7090, 7094, 7040, 7044 and 360 (short word arithmetic).

The trick to be described below does not work on machines such as the I.B.M. 650, 1620, Univac 1107 and the Control Data 3600 which round or truncate floating-point sums to single precision before normalizing them.

In the following program S2 is an estimate of the error caused when S = T was last rounded or truncated, and is used in statement 13 to compensate for that error. The parentheses in statement 23 must not be omitted; they cause the difference (S−T) to be evaluated first and hence, in most cases, without error because the difference is normalized before it is rounded or truncated.

```
1  S = 0.0
   S2 = 0.0
2  DO 4 I = 1, N
3  YI = ···
13 S2 = S2 + YI
   T = S + S2
23 S2 = (S−T) + S2
4  S = T
5  ····
```

Until double-precision arithmetic was made a standard feature of the FORTRAN language, the author and his students used this trick on a 7090 in FORTRAN II programs to perform quadrature, solve differential equations and sum infinite series.

REFERENCE:

1. WOLFE, J. M. Reducing truncation errors by programming. *Comm. ACM 7* (June 1964), 355–356.

W. KAHAN
University of Toronto
Toronto, Ontario, Canada

RECEIVED JULY, 1964

(Pracniques are continued on page 48)

the domain last entered is known and it is this domain in which the terminal point lies.

VII. Comments on Implementation of Algorithm

It is to be noted that actual use of this algorithm will frequently involve considerations of accuracy and significance. These must be viewed from the viewpoint of numerical analysis in terms of permissible errors as required by the nature of the given problem. Consequently, associated with each test, such as those for inclusion within or on a rectangle, coincident points, etc., there will normally be some small quantity δ (perhaps unique to the given test) which results in the test being more "conservative," i.e. retaining certain borderline cases for consideration under subsequent tests in the procedure.

A second point is that although the description of the algorithm is in terms of a static set of partitioning boundaries, it is quite adaptable to a situation in which the boundaries of the region are dynamically changing during the course of the problem solution. The structure of the list of significant points is such that insertions and deletions may be easily made to reflect changing boundaries. The nature of the algorithm will be unchanged. Of course, as already noted, processing time is conditioned by the number of boundaries involved, their orientation, etc.

An example of an applications area where a procedure of the type here described is extensively employed is that of computer-generated paths over configurations of surfaces for purposes of numerically controlled machining.[1] Tool paths are constructed by a series of connected straight-line segments. The generation of these tool paths is dependent on knowledge of the surface (domain) on which the current and previous tool points lie. Thus the intersection curves (boundaries) between contiguous constituent surfaces in the complex must be observed. In addition, the tool path currently being generated must conform to certain restrictions imposed by the previously generated tool paths, which thus constitute a dynamic set of boundaries. Since this algorithm is developed in terms of partitioning boundaries in a bounded, *plane region*, the given configuration of surfaces in such a machine tooling problem must be such that the projections into some fixed plane of the boundaries corresponding to their delimiting curves of intersection as well as the boundaries corresponding to the tool paths generated on the surfaces must satisfy the criteria for valid boundaries established in Section II of this paper.

Acknowledgments. The conception of the essence of the procedure described in this paper was primarily achieved by Mr. Arnold Siegel, then with IBM, now with Computronics, Inc., and Mr. Samuel M. Matsa, IBM Mathematics and Applications Department.

RECEIVED JUNE, 1964; REVISED SEPTEMBER, 1964

REFERENCE

1. AUTOPROMT, A System for Numerical Programming. M & A-12, IBM Math. & Appl., New York, May 1961.

PRACNIQUES—Cont. from page 40

LONGER STRINGS FROM SORTING*

Presented here is a description of a SORT/MERGE technique which can be incorporated into existing sorting methods to substantially increase the lengths of the sort output strings. This is accomplished by relaxing the sort string order criteria to the extent that string order is recoverable by the first merge pass. The standard sorting programs generate sequenced file records in tape record blocks. The records within a tape block are sequenced, and the last record of one block precedes the first record of the next block according to sequence. This new technique still requires that the tape blocks contain sequenced file records, but relaxes the second criteria. This revised requirement is that for a set of tape blocks with last record sort keys, the next tape block must contain records with sort keys at least as large as the lowest of the tabled last record sort keys. In addition, the last record sort key of this new block will replace the sort key which was previously lowest. The number of entries in this set of last sort keys is the number of tape records which must be merged from this tape on the first merge pass with the multiple records from the other sort pass output tapes. The number of buffers required for reading this tape exceeds the size of the set by one buffer. If the set consists of one entry, then the sort pass will generate the standard type output.

To use the above technique, it is necessary to have special sort pass routines. Separate techniques are proposed below for fixed and variable-length file records. They appear to have the most promise. The fixed-length method will be superior to the variable-length method when most records approach the longest record in length.

For fixed-length records, two read buffers and a scatter write procedure should be used. A binary selection technique (see tournament or replacement selection of the literature) should be used to select file records for output. Output is considered a paired procedure. As one tape record is being written, the file records from the other tape record are being replaced in a burst from the input buffers. After the replacement the next tape record is selected. This will cause the binary selection tree to pulsate. The file records on each tape block will be in sequence. During the replacement each new record is checked to determine if it is in the current string. When core is filled with the next string, that string is started.

The above method is believed to be superior to all others since it uses the fastest selection technique, requires only one data move, and uses a minimum of buffers. Conservative estimates are that overlapping tape blocks will be equivalent to having one extra magnetic tape drive on the first merge pass.

For variable-length records, it is proposed that a read to full core, and then write all of core, be used for great length variations. For this procedure, the sets of tape block last record keys will be maintained for all output tapes. After the first few core loads, output will be to all output tapes on each core load. Selection of output tape is according to the lowest sort key permitted for a tape. The procedure begins with the tape with the highest tabled sort key. When there are records remaining after all tapes are used for output, a new string is begun on the tape with the largest tabled sort keys.

Excluding the first and last records on each tape the strings will be approximately 2.5 core lengths long if the strings are evenly distributed on two or more tapes. Additional gains will be made for files where the record length variances cannot be efficiently handled by other techniques.

R. J. DINSMORE
The Bunker-Ramo Corp.
Eastern Technical Center
Silver Spring, Maryland

RECEIVED SEPTEMBER, 1964

* These techniques were developed by the author during his employment at Control Data Corporation.