elongated because the computer printer prints out characters leaving more space between rows than between columns.

RECEIVED FEBRUARY, 1969; REVISED JUNE, 1969

REFERENCES

1. TIECHROEW, D., AND LUBIN, J. F. Computer simulation—discussion of the technique and comparison of language. *Comm. ACM 9*, 10 (Oct. 1966), 723–741.
2. YOUNG, D. M., AND JUNCOSA, M. J. SPADE, a set of subroutines for solving elliptic and parabolic partial differential equations. Rep. P-1709, Rand Corp., Santa Monica, Calif., May 21, 1959.
3. ENGELI, M. Design and implementation of an algebraic processor. Ph.D. Diss., Institut für Angewandte Mathematik der ETH, Zurich, Apr. 1966.
4. EDGAR, D. S. Matrix inversion and iterative refinement. ILLIAC IV Doc. No. 194, Dep. Computer Sciences, U. of Illinois, Urbana, Ill., June 27, 1968.
5. MCINTYRE, D. E. Iterative problems on meshes—the equation of state of multimaterial Eulerian hydrodynamics. ILLIAC IV Doc. No. 184. Dep. of Computer Sciences, U. of Illinois, Urbana, Ill., Apr. 1968.
6. Quarterly Progress Report, ILLIAC IV Doc. No. 203, Dep. of Computer Sciences, U. of Illinois, Urbana, Ill., Apr.–June 1968.
7. CEA, J., NIVELET, B., SCHMIDT, L., AND TERRINE, G. *Techniques Numeriques de L'Approximation Variationnelle Des Problems Elliptiques*, Tome 1. Institut Blaise Pascal, Paris, Apr. 1966.
8. CEA, J., NIVELET, B., SCHMIDT, L., AND TERRINE, G. *Techniques Numeriques de L'Approximation Variationnelle Des Problems Elliptiques*, Tome 3,. Institut Blaise Pascal, Paris, Mar. 1967.
9. MORRIS, S. M. AND SCHIESSER, W. E. Salem—a programming system for the simulation of systems described by partial differential equations. Proc. AFIPS 1968 Fall Joint Comput. Conf. Vol. 33, Pt. 1, Thompson Book Co., Washington, D.C., pp. 353–357.
10. MORRIS, S. M. SALEM—A programming system for the simulation of systems described by partial differential equations. Ph.D. Diss., Lehigh U., Bethlehem, Pa., 1967.
11. CARDENAS, A. F. A problem oriented language and a translator for partial differential equations. Ph.D. Diss., Dep. of Engineering, U. of California, Los Angeles, Calif., Dec., 1968.
12. IBM System/360 operating system; PL/1 language specifications, Form C28-8201. IBM Systems Reference Library. 1968.
13. TAM, W. C. Non-linear partial differential equations and equations with derivative boundary conditions: improvements to PDEL. M.S. Th. Dep. of Engineering, U. of California, Los Angeles, Mar. 1969.
14. WIRTH, N., AND WEBER, H. EULER: A generalization of ALGOL, and its formal definition: Pt. I. *Comm. ACM*, *9*, 1 (Jan. 1966), 11–23; Pt. II *Comm. ACM 9*, 2 (Feb. 1966), 89–99.
15. BUSSELL, B. Properties of a variable structure computer. Ph.D. Diss., Dep. of Engineering, U. of California, Los Angeles, Calif., 1962.
16. IBM System/360 operating system, concepts and facilities, Form C28-6535. IBM Systems Reference Library.
17. ROSEN, S. (Ed.) *Programming Systems and Languages*. McGraw-Hill, New York, 1967.
18. SAMMET, J. *Programming Languages: History and Fundamentals*. Prentice-Hall, Englewood Cliffs, N.J., 1969.

ALGORITHM 377
SYMBOLIC EXPANSION OF ALGEBRAIC EXPRESSIONS [R2]

MICHAEL J. LEVINE*
Department of Physics, Carnegie-Mellon University, Pittsburgh, PA 15213
AND STANLEY M. SWANSON† (Recd. 27 Jan. 1969)
89 Mid Oaks Lane, St. Paul, MN 55113

* This work was done in part at the Division of Theory, CERN, Geneva, Switzerland.
† This work was done in part at the Institute of Theoretical Physics, Stanford University, Stanford, California.

KEY WORDS AND PHRASES: algebra, symbolic algebra, symbolic multiplication, algebraic distribution, algebraic multiplication, distribution algorithm, multiplication algorithm, product algorithm, polynomial distribution, polynomial expansion
CR CATEGORIES: 3.10, 3.17, 3.20, 4.13, 4.90

```
procedure EXPAND(M); integer M;
comment This algorithm algebraically expands arbitrarily
  parenthesized expressions into monomials. Distribution is direct,
  without intermediate expansion of lower level expressions. The
  algorithm has been used as a part of algebra programs in the-
  oretical physics [2, 3]. It was devised by H. J. Kaiser [1] and re-
  constructed by M. J. Levine. Expansion proceeds in two steps:
  First, parsing an input expression into a sequence of variable-
  operator pairs with associated parenthesis-level information,
  and then picking out the variables which belong together as
  factors of monomial terms. EXPAND accepts an abbreviated
  ALGOL-like syntax:
          ⟨variable⟩ ::= A | B | C | D | E | F | G
           ⟨primary⟩ ::= ⟨variable⟩ | (⟨expression⟩)
              ⟨term⟩ ::= ⟨primary⟩ | ⟨term⟩ × ⟨primary⟩
        ⟨expression⟩ ::= ⟨term⟩ | ⟨expression⟩ + ⟨term⟩
  REFERENCES:
  1. KAISER, H. J. Trace calculation on electronic computer. Nu-
     clear Physics 43 (1963), 620.
  2. LEVINE, M. J. Dirac matrix and tensor algebras on a compu-
     ter. J. Computat. Phys. 1 (1967), 454.
  3. SWANSON, S. M. Computer algorithms for Dirac algebra.
     J. Computat. Phys. 4, 1 (1969), 171;
begin
  integer LVL, N, T, U; Boolean array MULT[0:M];
  integer array V, VL, OPL, INDEX[0:M];
  integer procedure CHAR;
  begin
    integer C;
A: insymbol (2, '×') + (ABCDEFG␣;', C); if C = 12 then go
  to A;
    CHAR := C
  end CHAR;
  procedure DISTRIBUTE(N); integer N;
  comment There are two problems in distribution: first, to se-
    lect the variables in an expression which belong together as
    factors of the current monomial, and then to alter the reference
    marks in USED to indicate the next monomial. A Boolean
    value in USED is associated with each variable-operator pair.
    The expression is scanned from the left to select the first un-
    used variable, and then any variables in an additive relation
    to the selected variable are skipped before continuing the
    scanning for other factors. For the next monomial, the first
    selected variable followed by a "+" is marked used, and the
    marks on all the variables to the left are altered, depending on
```

```
    their operator type and level relation to the "+". Distribu-
    tion is from left to right (initial factors change most often);
  begin
    integer I, J, K, L, LEVEL;
    Boolean ALTER, PRODUCT, TERM;
      Boolean array USED[0:N];
    for K := 0 step 1 until N do USED[K] := false;
NEXT:  ALTER := true;  J := I := -1;
FACTOR: I := I + 1;  if USED[I] then go to FACTOR;
    J := J + 1;  INDEX[J] := I;
SKIP:  if MULT[I] then go to FACTOR;  LEVEL := OPL[I];
    if LEVEL > 0 then
    begin
      if ALTER then
      begin
        L := LEVEL;  LEVEL := VL[I] + 1;
        USED[I] := PRODUCT := TERM := true;
        ALTER := false;
        for K := I - 1 step -1 until 0 do
        begin
          if OPL[K] < LEVEL then
          begin
            LEVEL := OPL[K];  PRODUCT := MULT[K];
            if PRODUCT then LEVEL := LEVEL + 1;
            if LEVEL ≦ L then TERM := false
          end;
          if PRODUCT then USED[K] := TERM
        end
      end
      else
      begin
R:  I := I + 1;  if LEVEL ≦ OPL[I] then go to R
      end;
      go to SKIP
    end;
    PROCESS(J);  if ¬ ALTER then go to NEXT;
  end DISTRIBUTE;
  procedure PROCESS(J);  integer J;
  comment  A skeletal output routine (normally, monomials are
    further manipulated, sorted, and accumulated);
  begin
    integer I;  outstring (1, '+');
    for I := 0 step 1 until J do
    begin
      outsymbol (1, "×) + (ABCDEFG", V[INDEX[I]]);
      if I ≠ J then outstring (1, "×")
    end
  end PROCESS;
  comment  The following statements parse the input. A full-
    fledged input routine would extend ⟨primary⟩ to include num-
    bers and would class both "-" and "+" together as ⟨adding
    operators⟩. DISTRIBUTE still works with only "+" and "×"
    since a "-" is either absorbed into a following unsigned num-
    ber or replaced by the string "-1×". Only a single subexpres-
    sion, followed by an unparenthesized "+", is expanded at a
    time. M limits the size of this subexpression. A syntax error or
    a semicolon terminates the processing of input;
  LVL := N := 0;  U := CHAR;  if U < 4 then go to ERR;
A:  T := U;  if U = 13 then T := 3 else U := CHAR;
  if U ≧ 4 then
  begin
    if T = 1 then
    begin
      MULT[N] := true;  OPL[N] := LVL;  N := N + 1
    end
    else if T = 3 then
    begin
      MULT[N] := false;  OPL[N] := LVL;
      if LVL = 0 then begin DISTRIBUTE(N);  N := 0 end
        else N := N + 1
      end
      else if T = 4 then LVL := LVL + 1
      else go to ERR
    end
    else
    begin
      if T = 2 ∧ LVL > 0 then LVL := LVL - 1 else
      if T ≧ 5 then begin V[N] := T;  VL[N] := LVL end
      else go to ERR;
    end;
    if U ≠ 13 then go to A else if LVL = 0 then go to B;
ERR: outstring (1, 'syntax error');
B:  end EXPAND
```

CERTIFICATION OF ALGORITHM 310 [A1] PRIME NUMBER GENERATOR 1 [B. A. Chartres, *Comm. ACM 9* (Sept. 1967), 569]

DONALD G. RAPP AND LARRY D. SCOTT (Recd. 21 Apr. 1969 and 13 Aug. 1969)
Computer Science Group, Texas A&M University, College Station, TX 77843

KEY WORDS AND PHRASES: prime numbers, generator
CR CATEGORIES: 5.0

Algorithm 310 was coded in ALGOL 60 reference language and run on an IBM 360/65. The algorithm was tested for a large range of values including $m = 5$, 10, 501, and 2000. Reference [1] was utilized to verify the theory involved in the algorithm before actual machine testing.

The intention of Algorithm 310 is to give only the number of primes less than or equal to m. Actual confirmation in the initial phases was accomplished through additional instructions that printed the array of prime numbers, p, and the number of primes, k. Both references listed were useful in substantiation of the prime numbers given. These references were again useful in verifying that all the primes in the array had been discovered and printed.

Each test produced the correct number of primes, k, for the specified range, m. When the primes were listed, the total taken by hand agreed with the number, k, given by the algorithm.

REFERENCES:

1. ESTERMANN, T. *Introduction to Modern Prime Number Theory.* Cambridge U. Press, Cambridge, England, 1952.
2. LEHMER, D. N. Carnegie Institution of Washington, Publication No. 165. Hafner, New York, 1956.

REMARK ON ALGORITHM 336 [H] NETFLOW [T. A. Bray and C. Witzgall, *Comm. ACM 11* (Sept. 1968), 631–632]

T. A. BRAY AND C. WITZGALL (Recd. 20 Oct. 1969)
Boeing Scientific Research Laboratories, Seattle, WA 98124

KEY WORDS AND PHRASES: capacitated network, linear programming, minimum-cost flow, network flow, out-of-kilter
CR CATEGORIES: 5.32, 5.41

The algorithm as published contains an error on the 11th line following the line labled *XD*, which reads:

if *del* = *abs*(*cok*) ∧ ...

This line should read

if *del* ≥ *abs*(*cok*) ∧ ...

Fortunately, this error does not invalidate the algorithm but may in some cases lead to additional operations.