# 2 Chapter 2 Generation and Processing of Boolean Functions

## *2.1 Introduction*

The program block, SYSTEM, contains programs for:

1. Generating the Existence Function of as Boolean function from a set of constraints
2. Generating the truth table of a Boolean function from constraints or from its decimal equivalents
3. Analyzing or processing a Boolean function after it has been entered via either of the first two procedures

This chapter describes and illustrates the use of these programs.

## *2.2 Existence Function Generation*

Figure 2-1 shows the sequence of programs to be executed.

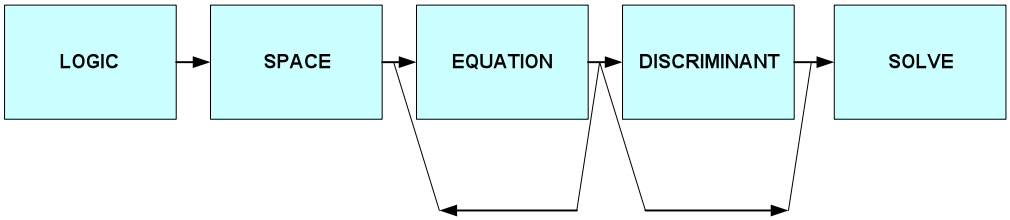


**Figure 2-1 Existence Function Generation**

1. The sequence always starts with LOGIC, which allows the user to type in the number of ***independent*** variables: (X j), j = 1, 2, … , NX
2. Function SPACE is called to establish the number of ***dependent*** variables (Y k),, k = 1, 2, … , NY.
3. Then, Function EQUATION is called as many times as there are constraints postulated for the system.
4. Calling DISCRIMINANT causes the existence function of the system to be printed in the form of a chart, where variables (X j) are represented horizontally and (Y k) are represented vertically. The Existence Function depends on all variables of the system, and is true for each validity configuration.

$$(X_1, X_2, \dots, X_{nx}, Y_1, Y_2, \dots, Y_{ny})$$

Which satisfies all the constraints of the system.

5. The constraints from a system of Boolean equations, which can be solved by calling SOLVE after DISCRIMINANT has been formed. Each of the existing solutions is printed in the form:

$$(Z\ k) = \text{Function of } (X\ j)$$

For k = 1, 2, … ,NZ with NZ = NY. The substitution of a solution in any equation (constraint) reduce that equation to an identity.

## 2.3 Truth Table Generation

Figure 2-2 shows the sequence of programs used to generate a truth table of Boolean functions (Z k), k = 1, 2, 3, … , NZ.

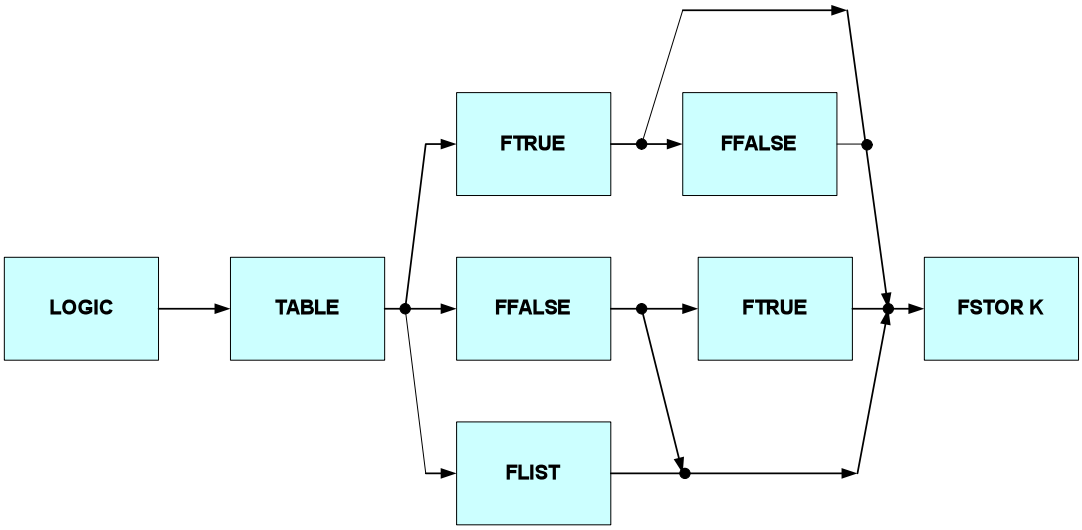


**Figure 2-2 Truth Table Generation**

The sequence starts with LOGIC again (to set the number of independent variables (X j), j = 1, 2, 3, … , NX). Routine TABLE prepares the format of the truth table (by setting the number o f functions (Z k), k = 1, 2, … , NZ to be stored).

The routines FTRUE and FFALSE develop Boolean functions from APL-defined constraints. The routine FLIST develops the Boolean function from its decimal equivalent (the conventional approach).

## 2.4 Processing of Boolean Functions

To process Boolean functions for logic design, a number of routines are available. These are listed in Table 2-1.

**Table 2-1 Routine for Processing Boolean Functions**

| Routine | Description |
|---|---|
| F CHART WIDE | Prints the Marquand chart of the function F whose width is indicated by integer WIDE |
| MINIMA F | Prints one of the N-minimal ∑Π forms of the function F.<br>Symbolism of the printout: A ≡ (X 1), B ≡ (X 2), etc. |
| PRIMEIMPLICANT 0F | Prints the sum of all prime implicants of the function 0F in the algebraic form using the symbolism described for MINIMA |
| DEGENERATION F | Looks for the latest dependencies between variables in a system with Existence Function F |
| K BULDIF F | Develops and prints the Boolean difference of the function F in relation to the variable (X k) |
| DECIMIN F | Produces decimal equivalents defining ONEs of a Boolean function F stored as a string of ZEROs, ONEs, and TWOs (DON'T'CARES) |
| DECIDONT F | Produces decimal equivalents defining the DON'T CARES of a Boolean function F stored as defined for DECIMIN |

## 2.5 Examples

Example 2.5.1. Develop the Existence Function of the full adder (see Figure 1-3). Use its equation in APL given in formula 1.1

Procedure:

1. Load program block SYSTEM
2. Call ∆SYSTEM for explanations
3. Begin by calling:

```
    LOGIC
NUMBER OF X-VARIABLES:
⍞:
    3
NX= 3
SYMBOLS FOR X-VARIABLES: (X J). (X J); J = 1 2 3
CALL: TABLE, SPACE
```

To produce the Existence Function of this system, call:

```
    SPACE
NUMBER OF Y-VARIABLES:
⍞:
    2
SYMBOLS FOR Y-0VARIALES: (Y K), (Y K); K = 1 2
NY= 2
XY-SPACE SYMBOL:  F[Y;X]
CALL: EQUATION

    EQUATION
WRITE THE EQUATION IN THE PRESCRIBED FORM:
F←F∧(((Y 1)+2X(Y 2)=(X 3)+(X 2)+(X 1))
```

**Note**: At this moment the terminal is in the APL immediate-execution mode.[1] The equation can be represented by any number of conditions executed immediately after one another:

```
F ← F ∧ (Relation 1)
F ← F ∧ (Relation 2)
F ← F ∧ (Relation 3)
F ← F ∧ (Relation 41)
          o
          o
          o
F ← F ∧ (Relation n)
```

Of course, each relation must be in the prescribed form. Routine DISCRIMINAANT can be called at any time to check the evolution of the Existence Function.

[1] Reference to the APL computer terminal. An IBM Selectric hooked up to the system and equipped with an APL type-ball.

When all relations (i.e., constraints of the system) have been introduced, the Existence Function is printed by calling DISCRIMINANT.

```
    DISCRIMINANT
HORIZONTAL SCALE: (X J) FOR J = 1 2 3
VERTICAL SCALE: (Y K) FOR K = 1 2
1 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0
0 0 0 1 0 1 1 0
0 0 0 0 0 0 0 1
```

Compare this with 1.2c.

The Existence Function always has the X-scale printed horizontally, and the Y-scale vertically. A with a Marquand chart, the X-coordinate IX = $(x_3\ x_2\ x_1)_2$ = 0, 1, … , 7, and the Y-coordinate (downward) IY = $(y_2\ y_1)_2$ = 0, 1, 2, 3. Each column contains exactly one non-zero. For that reason, we can develop the truth table of the full adder from its Existence Function. At the terminal, we do it by calling:

```
      SOLVE
NUMBER OF SOLUTIONS AFTER CONSTRAINTS: SOL = 1
DESIRED SOLUTION VECTOR:
□:
      1
SOLUTION NUMBER: 1
(Z 1) → 0 1 1 0 1 0 0 1              [1 2 4 7] ∪ ( )
(Z 2) → 0 0 0 1 0 1 1 1              [3 5 6 7] ∪ ( )
```

**Explanation**: The number of solutions is equal to the product of eight integers (in this example), one for each column. Each integer is equal to the count of non-zeros in the column. Here, SOL = 1 (number of solutions); the solution has only one element, element 1 (first solution); and the resulting Boolean functions of the truth table are designated by (Z 1) for the output (Y 1) = 1 and (Z 2) for the output (Y 2) = 1.

In the present case, functions do not contain DON'T CARES (expressed by the integer 2 within the string on the left-hand side of the printout and by decimal equivalents in the parentheses at the right-hand side. To print the truth table we call:

```
      FX
0 1 1 0 1 0 0 1
0 0 0 1 0 1 1 1
```

To understand the reason for using the symbol (Z 1) for the output Boolean function at the terminal (Y 1), let us call:

```
      (Y 1)
 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1
      (Y 2)
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1
```

The N-minimal ∑π form of the functions (Z k) of the truth table can be obtained immediately by the procedure MINIMA.

```
      MINIMA (Z 1)
ABC  + ABC  + ABC  + ABC
CRITICAL SET: 7 4 2 1

      MINIMA (Z 2)
BC + AC + AB
CRITICAL SET: 6 5 3
```

```
      MINIMA (Z 2)
AB + AC + BC
CRITICAL SET: 4 2 1

      MINIMA  (Z 1)
ABC + ABC + ABC + ABC
CRITICAL SET: 6 5 3 0
```

The minimization results suggest immediately:

```
(AB + AC + BC)(A +  B + C) = ABC + ABC + ABC
```

So that:

```
(Z 1)  ≡  (Z 2)(A + B + C) + ABC
```

(The preceding symbolism is Boolean: Multiplication stands for AND, and addition (+) stands for OR.

**Note**: The ∑π forms are printed by the terminal using this same symbolism.)

**Example 2.5.2.** Full adder analysis based on the truth table of threshold functions of its input (Figure 1-3 again). The sequence of procedures is illustrated in Figure 2-3.

```
      LOGIC
NUMBER OF X-VARIABLES:
⎕:
      3
NX = 3
SYMBOLS FOR X-VARIABLES: (X J), (X J);  J = 1 2 3
CALL: TABLE, SPACE
```

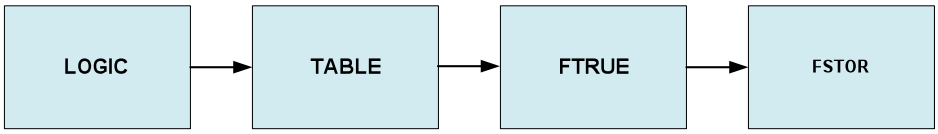


**Figure 2-3 Program Module Sequence for Truth Table of Threshold Functions**

```
      TABLE
NUMBER OF FUNCTIONS:
⎕:
      5
TABLE IS READY FOR FUNCTIONS (Z K) WITH K = 1 2 3
4 5
CALL OFFERINGS: FTRUE, FFALSE, FLIST

      FTRUE
WRITE A SUFFICIENT CONDITION OF  F  IN THE PRESCRIBED
FORM:
      F←(ANY LOGICAL APL-MEANINGFUL RELATION WITH VARIABLES
(X  J), (X  J))
EXECUTE IT AND CALL
EITHER: FSTOR K (WHERE  K  IS THE INDEX OF THE FUNCTION
(Z  K)
      OR: FALSE
      F←(0<(X 1) = (X 2) + (X 3)
```

```
      FSTOR 1
CALL: FTRUE, FFALSE, FLIST TO DEFINE THE NEXT
FUNCTION  (F K)
WITH K = 2
```

Again, a fixed form is prescribed for the condition definition. The variable F represents the Boolean function defined by the APL expression. The procedure FSTOR 1 will store it as the first row of the truth table. As a result of the response to program TABLE, there are five rows in the table ready to be filled with threshold functions.

The next threshold function is entered by:

```
      FTRUE
WRITE A SUFFICIENT CONDITION OF F IN THE PRESCRIBED
FORM:
  F←(ANY LOGICAL APL-MEANINGFUL RELATION WITH VARIABLES
(X J), (⍉X J))
EXECUTE IT AND CALL
EITHER: FSTOR K (WHERE K IS THE INDEX OF THE FUNCTION
(Z K)
      OR:  FFALSE
      F←1<(X 1) + (X 2) + (X 3)

      FSTOR 2
TRUTH TABLE IS READY:
MAY CALL: FTRUE, FFALSE, FLIST FOR K = 3 ≤ 5
MAY EXECUTE FX TO PRINT THE TABLE
```

Two rows of the truth table have now been filled. A third threshold function will be placed in the third row.

```
      FTRUE
WRITE A SUFFICIENT CONDITION OF F IN THE PRESCRIBED
FORM:
  F←(ANY LOGICAL APL-MEANINGFUL RELATION WITH VARIABLES
(X J), (⍉X J))
EXECUTE IT AND CALL
EITHER: FSTOR K (WHERE K IS THE INDEX OF THE FUNCTION
(Z K)
      OR:  FFALSE
      F←2<(X 1) + (X 2) + (X 3)

      FSTOR 3
TRUTH TABLE IS READY:
MAY CALL: FTRUE, FFALSE, FLIST FOR K = 4 ≤ 5
MAY EXECUTE FX TO PRINT THE TABLE
```

To check progress of function generation, let us print the state of the truth table:

```
      FX
0 1 1 1 1 1 1 1
0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

One of the  output functions of the adder (designated in Figure 1-3 by Y1) will be placed in the next (fourth) row of the truth table.

```
      FTRUE
WRITE A SUFFICIENT CONDITION OF F IN THE PRESCRIBED
FORM:
  F←(ANY LOGICAL APL-MEANINGFUL RELATION WITH VARIABLES
(X J), (⍉X J))
EXECUTE IT AND CALL
EITHER: FSTOR K (WHERE K IS THE INDEX OF THE FUNCTION
(Z K)
      OR:  FFALSE
      F←2<(X 1) + (X 2) + (X 3)

      FSTOR 4
TRUTH TABLE IS READY:
MAY CALL: FTRUE, FFALSE, FLIST FOR K = 5 ≤ 5
MAY EXECUTE FX TO PRINT THE TABLE
```

The truth table printout now appears as:

```
      FX
0 1 1 1 1 1 1 1
0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 1
0 1 1 0 1 0 0 1
0 0 0 0 0 0 0 0
```

It is clear that the function of the fourth row can be obtained by Boolean algebraic combination of the functions of rows 1, 2, and 3. To prove this, let us put the function: F ← (Z 3) ∨ ( (⍉Z 2) ∧ (Z 1) ) into row 5 of the table:

```
      FTRUE
WRITE A SUFFICIENT CONDITION OF F IN THE PRESCRIBED
FORM:
  F←(ANY LOGICAL APL-MEANINGFUL RELATION WITH VARIABLES
(X J), (⍉X J))
EXECUTE IT AND CALL
EITHER: FSTOR K (WHERE K IS THE INDEX OF THE FUNCTION
(Z K)
      OR:  FFALSE
      F←(Z 3) ∨  (Z 2) ∧ (Z 1)

      FSTOR 5
TRUTH TABLE IS READY:
MAY CALL: FTRUE, FFALSE, FLIST FOR K = 6 ≤ 5
MAY EXECUTE FX TO PRINT THE TABLE
      FX
0 1 1 1 1 1 1 1
0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 1
0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1
```

**Note**: The second output function of the adder (designated by Y2 in Figure 1-3) happens to be the threshold function of the inputs stored in the second row of the table.

To complete the analysis, the functions are N-minimized:

```
      MINIMA (Z 1)
C + B + A
CRITICALSET: 4 2 1

      MINIMA (Z 3)
ABC
CRITICALSET: 7

      MINIMA (Z̲ 2)
A̲B̲ + A̲C̲ + B̲C̲
CRITICALSET: 4 2 1

      MINIMA (Z 2)
BC + AC + AB
CRITICALSET: 6 5 3
```

By inspection of the truth table and by using the results of the minimizations, we get:

```
(Z 5) = (Z 3) v ((Z 2) ∧ (Z 1))      (2.1)
      =  ABC + (Z̲ 2)(A + B + C)
      = ABC +DA +DB + DC
WHERE
    D = (Z̲ 2)
      = ~(BC + AC + AB).
```

The resulting equations are implemented in the two-bit binary full adders (for instance, SC7482).

**Example 2.5.3.** Develop the Existence Function of a given sequential circuit, the S-R NOR flip-flop (Figure 1-5, Example 1.7). Because of the feedback loop, sequential circuit behavior is expected. The truth table generation procedure is inadequate for expressing this behavior. The proper procedure sequence is shown in Figure 2-4.

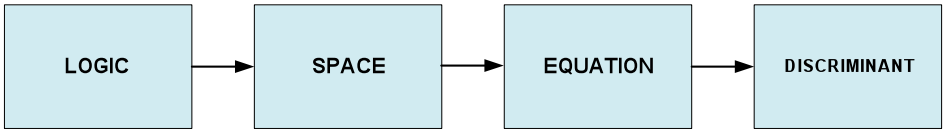


**Figure 2-4 Program Module Sequence of the Existence Function of a Sequential Circuit**

To begin we call:

```
      LOGIC
NUMBER OF X-VARIABLES:
□:
      3
NX = 3
SYMBOLS FOR X-VARIABLES: (X J), (X̲ J); J = 1, 2, 3
CALL: TABLE, SPACE
```

```
      SPACE
NUMBER OF Y-VARIABLES:
□:
      2
SYMBOLS FOR Y-VARIABLES: (Y K), (Y̲ K); K = 1, 2
NY = 2
XY=SPACE SYMBOL:  F[Y; X]
CALL: EQUATION

      EQUATION
WRITE THE EQUATION IN THE PRESCRIBED FORM:
F←F∧(ANY LOGICAL RELATION)
EXECUTE IT AND CALL: EQUATION, DISCRIMINANT, SOLVE
```

The note given in Example 2.5.1 shows that it is unnecessary to condense all constraints into a single APL expression, which for this circuit would be:

```
F ← F ∧ ((Y 2) = (Y̲ 1) ∧ (X̲ 1))
      ∧ ((Y 1) = (Y̲ 2) ∧ (X̲ 2) ∧ (X̲ 3))
```

Clearer expressions are obtained by introducing and executing the simultaneous constraints one after the other, as it is done here:

```
F ← F ∧ (Y 2) = (Y̲ 1) ∧ (X̲ 1)
F ← F ∧ (Y 1) = (Y̲ 2) ∧ (X̲ 2) ∧ (X̲ 3)
```

Again, remember that APL executes these expressions as soon as they are entered. The existence function is now printed by calling:

```
      DISCRIMINANT
HORIZONTAL SCALE: (X J) FOR J = 1 2 3
VERTICAL SCALE: (Y K) FOR K = 1 2
 0 0 0 1 0 1 0 1
 1 1 0 0 0 0 0 0
 1 0 1 0 1 0 1 0
 0 0 0 0 0 0 0 0
```

Each non-zero of the Existence Function represents a steady state of the circuit. The present circuit has nine steady states.

**Note**: This Existence Function agrees with Figure 1-2d (Example 7, Chapter 1).

**Example 2.5.4.** Design a combinational network specified only by its truth table. The table is given by “decimal equivalents”. In that case, the procedure sequence of Figure 2-5 is indicated. The given function has six independent variables.

```
      LOGIC
NUMBER OF VARIABLES:
□:
   6
NX = 6
SYMBOLS FOR X-VARIABLES: (X J), (X̲ J); J = 1 2 3 4 5 6
CALL: TABLE, SPACE
   TABLE
NUMBER OF FUNCTIONS:
□:
   1
TABLE IS READY FOR FUNCTIONS  (Z K) WITH K = 1
CALL OFFERINGS: FTRUE, FFALSE, FLIST
```

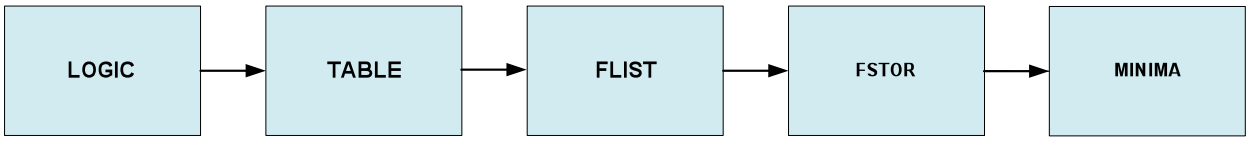


**Figure 2-5 Program Module Sequence for Combinatorial Design via Truth Table**

Here, we will select FLIST:

```
      FLIST
DECIMAL EQUIVALENTS OF ONES (AT LEAST ONE ITEM):
[]:
      0 8 14 17 18 19 20 22 24 27 34 36 38 43 46 52 56
 59 60 61 63
DECIMAL EQUIVALENTS OF DONT CARES:
[]:
      2 3 4 5 6 7 10 11 12 23 25 26 28 33 35 37 41 45 49
53 54 55 57 58 62
CALL: FSTOR  K  (WHERE K IS WELL SPECIFIED)

      FSTOR 1
CALL:  FTRUE, FFALSE, FLIST TO DEFINE THE NEXT
FUNCTION  (F K)
 WITH K = 2
```

The given Boolean function is now stored as (Z 1), and we want to inspect its Marquand chart. To print that chart, we call:

```
      (Z 1) CHART 8
HORIZONTAL SCALE: (X J) FOR J = 1 2 3
  VERTICAL SCALE: (X J) FOR J = 4 5 6
0 1 2 2 2 2 2 2
1 0 2 2 2 0 1 0
0 1 1 1 1 0 1 2
1 2 2 1 2 0 0 0
0 2 1 2 1 2 1 0
0 1 0 1 0 2 1 0
0 2 0 0 1 2 2 2
1 2 2 1 1 1 2 1
```

**Note**: The procedure is called with two arguments:

- on the left side of the procedure name is the symbol of the function to be charted;
- on the right side is the number of columns of the chart, which must be a power of two (for triadic charts, a power of three).

The chart is filled with integers from the set (0, 1, 2) with the following meanings:

- 0 means FALSE
- 1 means TRUE
- And 2 means UNSPECIFIED (DON'T Care)

To design a two-level AND-OR (two-level NAND) combinational network, we first program the N-minimization of (Z 1):

```
      MINIMA (Z 1)
ABCE + BCDE + ACDF + ACD + DEF + BCF + ACDF + ACEF
CRITICAL SET: 46 34 8 36 63 18 17 43
```

That calls for eight gates at the first level and an 8-input fan-in at the second level. The first level gates need 29 inputs. Inversion of B is not necessary. The maximum number of inputs per gate is 4.

To design a two-level OR-AND (two-level NOR) combinational circuit for the given function, we N-minimize the complement of (Z 1) of the function (Z 1) and develop an N-minimal π∑ form by inverting the form obtained for (Z 1):

```
      MINIMA (Z 1)
ABCE + ACE + BDEF + ABCD + CDEF + CEF + ACEF
CRITICAL SET: 16 47 44 42 50 31 9
```

That calls for seven gates at the first level and a 7-input fan-in at the second level. The first level gates need only 26 inputs. It is clear that this solution is simpler. **Note**:

```
(Z 1) = (A+B+C+D)(A+C+E)(B+D+E+F)
        (A+B+C+D)(C+D+E+F)(C+E+F)(A+D+E+F)
```

The low value of the execution time connected with the MINIMA procedure is due to the fact that the ∑π extension algorithm avoids the listing of all prime implicants of the given function, which is time-consuming. To illustrate this point, the sum of all prime implicants is printed out by calling:

```
      PRIMIMPLICANT (Z 1)
ALGEBRAIC FORM OF THE COMPLEMENTARY FUNCTION:
BCDEF + ABDEF + ACDEF + ABCEF + BCDEF + ABCEF + ABDEF +
ABCD + ACDF
CRITIAL SET: 51 44 42 39 30 21 9 16 15
SUM OF ALL PRIMIMPLICANTS:
ABCD + ACD + ABCD + ABCE + ABCE + ABDE + CDE + BCDE +
ABCE + BCDF + CEF + DEF + ACEF + ABCF + BCF + ABDF +
ACDF + ABCE + ABDE + ABDE + BCDE + ACDE + BCDE + ABF +
ABCF + ACDF + BDF + ACDF + ACEF + ABEF + ACEF + ADEF +
ADEF + CDEF
```

The function (Z 1) has 34 prime implicants. Only eight of them were used to form an N-minimal ∑π form. This is wasteful. The procedure MINIMA developed prime implicants covering points belonging to a ***critical set***:

```
{46, 34, 8, 36, 63, 18, 17, 43}
```

whose elements are ***mutually term exclusive*** (that is, no term implicant of the given function exists that covers any pair of elements of that set) and which has the property to include the maximum number of mutually term exclusive elements.

**Note**: The algebraic form of the complementary function covers all zeros of the given function without being N-minimal. The N-minimal ∑π form of (Z 1) is much simpler due to DON'T CARES, which are not considered for the algebraic form of the printout.

**Example 2.5.5**. When the function is very simple and when all we want to do is minimize its algebraic form, is is possible to develop its APL representation without beginning with LOGIC. For instance, a Boolean function FX is specified by decimal equivalents {3, 5, 6}, and FX is unspecified for {1, 2, 4, 7}. The APL vector will be a string of eight integers (it is obvious that the function has three independent variables) taken from the set {0, 1, 2}:

- 0 means FALSE
- 1 means TRUE
- and 2 means UNSPECIFIED (DON'T CARE).

At the APL terminal, the function can be developed by:

```
           FX ← 0 2 2 1 2 1 1 2
        FALSE → 0
         TRUE →       3   5 6
UNSPECIFIED →  1 2    4     7
```

The function can thus be minimized immediately as follows:

```
FX ←0 2 2 1 2 1 1 2

      MINIMA FX
CYCLE DISSOLOTION
C + B
CRITICAL SET: 6 3
```

The given function FX was “cyclic”. In that case, the procedure MINIMA dissolves the cycle by relaxing the condition leading to the minimal count of literals in the resulting ∑π form. The number of terms of that form remains minimal. However, the critical set printout can be distorted.

For instance, in the present case, it can be shown that no critical set exists that has two elements. The set {6, 3} is not “critical” because the term B covers both elements 6 and 3.

On the other hand, all N-minimal ∑π forms of FX: C + B, B + A, and A + B, have two terms more than the number of elements in critical sets {6}, {3}, and {5}, which possess one element only. The function FX is labeled “abnormal” because the N-minimal ∑π form always has more terms then the count of elements in its critical set.

To meet an abnormal Boolean function in actual computer application is highly improbable. Cyclic functions usually happen to be normal. For instance, the function treated in the following is cyclic, but normal:

```
      LOGIC
NUMBER OF VARIABLES:
⎕:
   4
NX = 4
SYMBOLS FOR X-VARIABLES: (X J), (X̲ J); J = 1 2 3 4
CALL: TABLE, SPACE

   TABLE
NUMBER OF FUNCTIONS:
⎕:
   1
TABLE IS READY FOR FUNCTIONS  (Z K) WITH K = 1
CALL OFFERINGS: FTRUE, FFALSE, FLIST

      FLIST
DECIMAL EQUIVALENTS OF ONES (AT LEAST ONE ITEM):
⎕:
      0 1 3 4 6 7 9 10 11 12 13 14
DECIMAL EQUIVALENTS OF DONT CARES:
⎕:
      10
CALL: FSTOR  K  (WHERE K IS WELL SPECIFIED)

      FSTOR 1
CALL:  FTRUE, FFALSE, FLIST TO DEFINE THE NEXT
FUNCTION  (F K)
 WITH K = 2
```

```
      (Z 1) CHART 4
HORIZONTAL SCALE: (X J) FOR J = 1 2
  VERTICAL SCALE: REMAINING VARIABLES
  1 1 0 1
  1 0 1 1
  0 1 1 1
  1 1 1 0

      MINIMA (Z 1)
CYCLE DISSOLUTION
BCD  +  AC  + ABD  +  BCD  +  ABD
CRITICAL SET: 13 9 10 7 0
```

**Note**: "Cycle dissolution" means that the critical set may be faulty. Here, minterms 13 and 9 are mutually term ***inclusive*** (both being covered by A$\underline{B}$D), which is against the ruling that each pair of elements in the critical set must be mutually term ***exclusive***. The resulting form is always correctly N-minimal. The minimal count of literals is not warranted. To get the best N-minimal ∑π forms of cyclic functions, use the program block called OPTIMA.

**Example 2.5.6**. The procedure X NEWORDER F is used to interchange the indexing of variables (i.e., to change the charting pattern of F). This procedure is used as a subroutine in BUIDIF, generating the Boolean difference of F. The dummy variable X is used to express the reordering law; it has the form of a vector composed from the elements of the set {1, 2, 3, …} containing one element for each variable (X j), j = 1, 2, 3, … of the function F.

**Note**: These variables are represented in the MINIMA printout as A, B, C, …. For instance:

```
2 3 4 1  NEWORDER  F
```

Is well-formed when F is a function of four variables (X j), j = 1, 2, 3, 4 (represented by A, B, C, and D in the MINIMA printouts).

To describe the reordering law, the following sketch is useful:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Original order* | *1* | *2* | *3* | *4 = J old* | *A* | *B* | *C* | *D* |
| *X-symbol* | *2* | *3* | *4* | *1* | *2* | *3* | *4* | *1* |
| *New order* | *4* | *1* | *2* | *3 = J new* | *D* | *A* | *B* | *C* |

The variable (j 1) of the new order represents the variable (j 4) of the old. IN the printout, printing of A will be replaced by the printing of D (note that new D stands below the old A in the preceding sketch.

To illustrate, start with:

```
F1←1 0 1 0 1 0 1 0 0 1 0 1 1 1 0 1
```

and do

```
      F1 CHART 4
HORIZONTAL SCALE: (X J) FOR J = 1 2
  VERTICAL SCALE: (X J) FOR J = 3 4
  1 0 1 0
  1 0 1 0
  0 1 0 1
  1 1 0 1
```

To print the Marquand chart of F1. In this chart, the horizontal variables are A and B; the vertical variables are C and D.

The MINIMA printout gives:

```
      MINIMA F1
AD  +  ABC  + AD
CRITICAL SET: 15  12  6
```

The reordering procedure, called by:

```
F4←2 3 4 1 NEWORDER F1
```

Produces the function F4, whose variables are reordered.

To show the result of the reordering operation, call:

```
      MINIMA F4
CD  +  ABD  + CD
CRITICAL SET: 15  6  3
```

**Note**: The printout of MINIMA F1 was transformed into that of MINIMA F4 according to the sketch discussed previously:

AD transforms into DC, ABC into DAB, and AD into DC.

Second illustration:

```
F4←2 1 3 4 NEWORDER F1
```

This time, the variables A and B were mutually interchanged, while C and D were left unchanged. The Marquand chart after this reordering is produced by:

```
      F4 CHART 4
HORIZONTAL SCALE: (X J) FOR J = 1 2
  VERTICAL SCALE: (X J) FOR J = 3 4
  1 1 0 0
  1 1 0 0
  0 0 1 1
  1 0 1 1
```

Comparison of this chart with the chart of the original function F1 shown previously shows that the column corresponding to BA was interchanged with the column belonging to BA.

Finally, note the result of:

```
      MINIMA F4
BD  +  ABC  + BD
CRITICAL SET: 15  12  5
```

Example 2.5.7. Find the Boolean difference of a given function F1 (from example 2.5.6) in relation to each of its variables (X k), k = 1, 2, 3, 4. A procedure is available to solve this problem. By calling:

```
R ← K BUILDIF F
```

a function of all variables is formed. This function is true for a given configuration of all variable validities if and only if the value of the function F varies with the variation of the variable (X k).

To illustrate, let us call:

```
F4 ←1 BULDIF F1
```

And express F4 (the difference for (X 1) ) algebraically by calling:

```
      MINIMA F4
B  +  C  +  D
CRITICAL SET: 15 9 5
```

In the sense of the definition, the function F1 must change its value if and only if:

```
F4  = 1  = B + C + D
```

To check it, we recall that:

```
F1 = AD + ABC  + AD
```

```
Test 1: B = 1, C, D = (any) => F4 = 1.
        Then F1 = AD + AD ≡ (A = D).
        Conclusion: F1 must change with
        A for any D.

Test 2: D = 0, B, C = (any) => F4 = 1.
        Then F1 = ABC + A ≡ A, which
        Changes with A.

Test 3: BCD = 1 => F4 = 0.
        Then B = C = D = 0,
        and F1 = A + A = 1
        So that F1 does not change with A.
```

Another illustration:

```
 F4←4 BUILDIF F

      MINIMA F4
B  +  A  +  C
CRITICAL SET: 14 13 8

      F4 CHART 4
HORIZONTAL SCALE: (X J) FOR J = 1 2
  VERTICAL SCALE: (X J) FOR J = 3 4
  1 1 1 1
  0 1 1 1
  1 1 1 1
  0 1 1 1
```

The chart shows that the value of F1 changes with (X 4) (or D) in every window of the chart except two (those filled with zeros).