THEORY OF CUBICAL COMPLEXES WITH APPLICATIONS TO DIAGNOSIS AND ALGORITHMIC DESCRIPTION

by

J. Paul Roth
IBM T. J. Watson Research Center
Yorktown Heights, New York

SECOND QUARTERLY REPORT
Covering the Period May 26, 1970 - August 10, 1970

This work was performed for the Jet Propulsion Laboratory, California Institute of Technology, sponsored by the National Aeronautics and Space Administration under Contract NAS7-100.

Subcontract No. 952709
IBM/California Institute of Technology, JPL
(Subcontract Under NASA Contract No. NAS7-100)

1 of 10

1. 2-Level Boolean Minimization PL/1 Program

Planning and programming MOM, the multiple-output 2-level Boolean minimization algorithm[*], has continued. Subroutines for subsuming and for inputing and output lists (covers) of singular cubes has been completed. Planning is being made for the #-procedure (for differencing cubical covers). The PL/1 programming is being done by Leroy Junker of the Programming Systems Department. This work is of interest to CD architecture and SDD EIS.

2. Module Diagnosis

For computer circuits design with medium or large-scale integrated circuits, it is necessary to diagnose failures down to the module; it is not necessary or desirable to diagnose down to the logic circuit element, for the (macroscopic) module is a replaceable unit. For this purpose, an algorithm has been developed which computes a test, if one exists, to distinguish between module failures. In general, however, it takes more than one test to make such a distinction.

This is a generalization of "An Algorithm to Compute a Test to Distinguish Between Failures in a Logic Circuit", which appeared in the Proceedings of the 1970 International Computer Group Conference, June 1970, Washington, D. C. (IEEE Inc 70C 23-C), pp. 247-249.

The mathematical formulation of the problem goes as follows. Given failures $F_1, \ldots, F_r$ associated with module f and failures $G_1, \ldots, G_s$ associated with module g ; given that exactly one failure of the type F_i or G_j has occurred, to ascertain whether this failure is one of the F_i's or one of the G_j's .

The technique is to start a set of D-chains $TC_1, \ldots, TC_r$ from failures $F_1, \ldots, F_r$ and a set of "E-chains" $TE_1, \ldots, TE_s$ from failures $G_1, \ldots, G_s$ as in the above reference (a copy is attached as an appendix). Then only the TE_j chain can interfere with the TC_i chains and vice versa.

[*] J. P. Roth and E. G. Wagner, "A Calculus and an Algorithm for a Logic Minimization Problem Together with an Algorithmic Notation", IBM Research Report, RC-2280, November 1968.

Now it is desirable that <u>each</u> TC_i chain (or TE_j chain) be driven to a primary output, and then in the consistency operation justified in terms of primary inputs without any interference from the TE_j chains or (TC_i chains). In general, however, it is not possible to do this. Thus, what is done is that a TC_1 for failure F_1 is generated which distinguishes it from each of the failures $G_1,\ldots,G_s$; if this is not possible, then two or more such test cubes defining tests, say S_1, S_2 , are obtained distinguishing F_1 from the G_j . This is done by the process of backing up at decision points when the test cube fails. Then S_1 and S_2 are simulated for the remaining failures $F_2,\ldots,F_r$ to ascertain those which distinguish themselves from $G_1,\ldots,G_s$. From among these which are not distinguished, one is selected and a test (one or more) developed which distinguishes it from the G_j etc., until a complete set of tests distinguishing the F_i from the G_j is obtained. Note that, in general, the tests will be distinguished on more than one primary output. Likewise, the roles of the F_i and G_j may have to be reversed in the event that no tests of the type S_1, S_2 exist.

3. <u>Diagnosis of Circuit Failures Wherein the Circuits are Described Using Higher-Level Blocks</u>

Diagnosis of the failures of complex sequential logic circuits by means of an algorithm from the detailed logic diagrams themselves is today considered a practical impossibility. The reason is that these complicated circuits may require very long sequences for the explication of their behavior. If, however, a higher-level description of the design were available from the system master tape and the primitive D-cubes and primitive D-cubes of failure* were available, the computation of tests would be considerably simplified. In particular, we are thinking of including all the higher-level blocks that are used in the field

*Roth, Bouricius, Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits", IEEE TEC, October 1967, V. EC-16, pp. 567-579.

engineering automated logic diagram FEALD; there already exists a master tape or means of generating a master tape for these.

4. Reading ALD Charts by the Field Engineer

It has been discovered that the field engineers are taught to trace signal patterns forward from primary inputs through the circuits to primary outputs in the attempt to locate failures. We have shown, however, that it is a considerably more efficient, and hence faster, procedure to trace from the primary output lines, where errors have been indicated, backward to the site of failures. This result will be communicated to Advanced Maintenance Development AMD of the FE Division.

5. Parallel Execution of Algorithms

The F-notation (cf. Roth-Wagner above and appendix) is a functional means of description of algorithms. It turns out that the "deep" parallelism inherent in an algorithm is rather naturally evident by examination of the "structural graph" of an F-algorithm, an algorithm written in the F-notation. With Jerry Kurtzberg of the Computer Science Department, we have been looking at the hardware implications of this; i.e., how to design a computer system for efficiently executing algorithms that are highly parallelizable.

6. Comparison of Various Algorithms for Computing Diagnostic Tests

With Carter et al, we have completed a paper entitled, "A Comparison of Various Methods to Compute Tests for Failures in Logic Circuits".

APPENDIX

A NOTATION FOR THE DESCRIPTION OF ALGORITHMS

by

J. Paul Roth

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT: This paper describes a notation for the description of algorithms; written in this form, the algorithms are termed F-algorithms. Indeed, an F-algorithm may be viewed as a precise means of defining an algorithm. It is proven that every functionally-expressed algorithm can be written as an F-algorithm. One of the features of this "F-notation" is that a fixed set of primitives is not used.

A method is given for transforming an F-algorithm into an APL program provided a translation is given of the primitives of the F-algorithm into appropriate APL programs.

The advantage of the F-notation is the great comparative ease of specifying an algorithm in this notation because of the fact that the primitives are arbitrary, and that in a certain precise sense, it is a minimal means for describing algorithms.

INTRODUCTION

The theory of algorithms, as practiced e.g. by Markov, is a rather esoteric affair and difficult to relate to algorithms of practical interest. On the other hand, when algorithms are described in various programming languages, they may be rather far removed from their mathematical formulations. The F-notation is a functional and rather natural means for expression of algorithms, and will be described in this paper. It will be shown that any algorithm functionally expressed may be expressed as an F-algorithm. F-algorithms will be defined, and an example of an F-algorithm will be given.

1. DEFINITION OF AN F-ALGORITHM

The first thing that we must define is the F-statement. These are of two varieties -- the executional statement and the conditional statement. The <u>executional statement</u> has the form

$$(C = \underline{F}(A_1,\ldots,A_n))$$

where C is the <u>range-</u> (codomain), and the A_i are the domain-symbols, and $\underline{F}$ the <u>function-symbol</u>, for either a primitive function or a previously defined function; the interpretation is that $\underline{F}$ is a function to be executed, the A_i its arguments, and C its value.

The conditional statement has the form

$$(F(A) = B|D \equiv F(E_1,\ldots,E_r)) .$$

This has the interpretation that if $f(A) = B$, f being another well-defined function, then the statement function $D = \underline{F}(E_1,\dots,E_r)$ is to be executed.

An <u>F-formula</u> is composed of a string of F-statements in the following format: a left square bracket followed by the functional name

$$[\, Z \equiv \Gamma(R_1,\dots,R_S)\ \dots]$$

of the F-formula followed by the equivalence sign $\equiv$, followed by the string of F-statements, each enclosed by round brackets, concluded with]. The arguments $R_1,\dots,R_S$ of the F-formula are independent arguments of its F-statements.

An <u>F-algorithm</u> is a string of F-formulas, in which the name of the F-algorithm $\underline{E}$ is written before the F-string in the form $E = \underline{E}(G_1,\dots,G_n)$ where the G_i are the primitive arguments of the F-statements and F-formulas, i.e., arguments which are not themselves the values of F-formulas or F-statements.

The only requirement concerning the order of F-formulas and F-statements of an F-algorithm is a natural one: if F-formula A is used as a statement in F-formula B, then this version of A must be completed before B is completed. Similarly, for F-statements in an F-formula: within an F-formula, statement S must be completed before statement T, if statement T has as an argument a value computed by statement S. The computer, on which the F-algorithm is being executed, should decide on the order of execution and the amount of parallelism that is possible in execution.

We now wish to show that an algorithm expressed in a general kind of functional notation can always be written as an F-algorithm. The algorithm A is described by a sequence of statements of the form

$$B_i = F_i(A_1,\dots,A_{n_i})$$

consisting of both the conditional and executional variety. Suppose then that A consists of exactly one statement $B = F(A_1,\ldots,A_n)$; the F-formula

$$[B = \Gamma(A_1,\ldots,A_n) \equiv (B = F(A_1,\ldots,A_n))]$$

faithfully describes the algorithm.

Suppose then it has been established for all algorithms of n formulas or less that an appropriately defined F-algorithm faithfully describes the algorithm. Consider then an algorithm consisting of $n+1$ formulas $\phi_1,\ldots,\phi_{n+1}$. But $\phi_1,\ldots,\phi_n$ is an algorithm of n formulas, and therefore equivalent to an F-algorithm ψ_n. But then $\psi_n(\phi_{n+1})$ is an F-algorithm which clearly is a faithful representation of the algorithm $\phi_1,\ldots,\phi_{n+1}$.

QED

We have thus proved

Theorem. Any algorithm written in a functional notation can be faithfully represented by an F-algorithm.

To illustrate the method of describing algorithms, we shall describe the Newton-Raphson algorithm for finding the square root of a given number R.

$$X' = \underline{M}(R)$$

(1) $[X' = \underline{N}(R,X) \equiv (X' = 1/2\ (X + R \div X))$

$(X' - X < \varepsilon \mid X')$

$(\underline{N}(R,X'))]$

(2) $[X' = \underline{M}(R) \equiv (X = 1/2\ (1 + R))$

$(X' = \underline{N}(R, X))]$

The expression $X' = \underline{M}(R)$ specifies that the value of the algorithm is X', given by the formula $\underline{M}(R)$. The first F-formula (1) $X = N(R,X)$ specifies a function of two variables R and X. The first F-formula given a first approximation for the square root $X' = 1/2\ (X + R \div X)$. The second gives a terminating condition: if $X' - X$ is less than ε, then X', i.e., the answer is X'; otherwise $\underline{N}$ is performed again in the third F-formula $N(R,X')$ for a different approximation X', and this is repeated until the termination condition is satisfied.

Finally, the second F-formula $X' = M(R)$ gives the zero th approximation $X = 1/2\ (1 + R)$ for the square root. Thence, the value of the algorithm is $X' = \underline{N}(R,X)$.

2. TRANSLATION OF AN F-ALGORITHM INTO APL

In general, an F-algorithm does not have a fixed set of primitives, although for this construction we will assume such and assume further a translation into primitives or defined APL functions. Furthermore, since an F-algorithm is itself a string of F-formulas, it will be sufficient to describe the translation of a single F-formula into an APL function. Let thus

$$[y = N(x) \equiv (F_1)(F_2)\ \dots\ (F_k)]$$

be a single F-formula. This is translatable into APL in the form

```
∇y  ←  Nx
[1]     F_j
[2]     F_2
  ⋮
[k]     F_k
[k+1]   ∇
```

Note that, in general, APL functions can have only one or two arguments, but since these may be vectors, there is no loss in generality in assuming single (vector) arguments. The F_i are assumed to be either primitives or defined functions, and by inductive hypothesis, expressible in APL.

QED