ISSN 0146–411

[redacted]

AUTOMATIC CONTROL AND COMPUTER SCIENCES

(Avtomatika i Vychislitel'naya Tekhnika)

1985

Vol. 22, No. 1
26 5

ALLERTON PRESS INC.

(6)

STRUCTURAL APPROACH TO SOFTWARE IMPLEMENTATION OF BOOLEAN FUNCTIONS

B. P. Kuznetsov and A. A. Shalyto

Avtomatika i Vychislitel'naya Tekhnika,
Vol. 19, No. 5, pp. 84-88, 1985

UDC 62-507

The authors examine structured binary programs that implement Boolean formulas. Estimates are obtained for the complexity of such programs, and an analytic method of constructing them is proposed.

To ensure program quality (e.g., tractability, modifiability), extensive use of currently made of the structured approach to computer programming [1, 2]; this approach was considered in part in [3] in relation to programmable logic devices (PLD).

In this paper, with allowance for the specific features of binary programs (BP) that implement Boolean formulas, we will consider a structural approach to programming that is oriented toward obtaining estimates for the complexity of BP and of the corresponding flow charts.

When formula constructs (e.g., the FODUS language [4]) are employed, translation is performed in sequential and independent fashion. Therefore the complexity of a translated program is equal to the combined complexity of the programs translated for each formula separately [5]. In this connection, we will consider the problem of constructing a structured program for one formula.

To set up a flowchart for a BP that implements a Boolean formula specified in AND-OR-NOT logic, we employ the formula method [6]. The resultant flowchart corresponds to a linear binary graph (LBG) [7]; however, this graph is not structured.

In order to structure an LBG, we construct an execution tree for it [1], in which the paths represent all possible implementational sequences of the LBG. By combining the corresponding statement vertices, we obtain a structured binary graph (SBG).

The execution tree is constructed with allowance for duplication of LBG vertices upon traversing all paths in it. The fewer paths the LBG contains, therefore, the smaller the number of vertices that must be duplicated, and hence the fewer the number of conditional and statement vertices that the SBG will contain.

Paper [7] proposed a method of calculating the number of paths in a LBG through conversion of the Boolean formula (or its inversion) realized by this graph into an orthogonal disjunctive normal form (ODNF). In addition, this paper offered a technique for choosing the order of writing a specified formula, so as to minimize the number of paths in the LBG. For threshold formulas, this technique yields the minimum number of paths; for nonthreshold formulas it yields a fairly simple method of calculating their number.

The number of paths S_r in an LBG containing h conditional vertices satisfies the inequality [8]

$$h+1 \leqslant S_r \leqslant F_{h+2}, \qquad (1)$$

where F_{h+2} is the (h + 2)-th Fibonacci number. Here the maximum number of "one" paths S_r^1 is equal to F_{h+1}, while the maximum number of "zero" paths S_r^0 is F_h, where F_{h+1} and F_h are the (h + 1)-th and h-th Fibonacci numbers, respectively.

The number of paths S_f in an LBG corresponding to a Boolean formula in AND-OR-NOT log-

85