INTRODUCTION

The abstract synthesis of a control automaton the operating conditions of which are given in the form of a microprogram is that stage of the synthesis which sets up a correspondence between the microprogram and the transition and output tables describing the Mealy automaton which realizes this microprogram (in the sense used in [1]). It follows that tables can be used for the output language of the abstract synthesis, though these are somewhat unwieldy compared with an analytic representation of the operating conditions. On the other hand, the existing methods for the minimization and structural synthesis of automata have in the main been developed for automata defined by tables, so that we shall use them too.

Naturally we make the requirement that the tables obtained at this stage are written economically, i.e., are not more unwieldy than is necessary; that the additional minimization possibilities determined by the nature of the microprogram and the specific features of the microinstructions shall be completely taken into account; and that the algorithm for transition from the microprogram to the tables shall be simply and conveniently realized by a digital computer.

In this paper we solve the problem of obtaining an algorithm which satisfies these requirements and constructs transition and output tables of an automaton from the representation of the microprogram taking into account various properties of the microoperations. Since the problem is solved on an abstract level we consider only the properties of microoperations able to change logical conditions in the microprogram.

1. DESCRIPTION OF LANGUAGE DEFINING MICROPROGRAMS

A microprogram written in some language on the one hand defines the conditions of operation of the control automaton and, on the other, defines its structure.

The language in which microprograms are defined must therefore satisfy the following basic requirements.

1. It must be convenient for the human user, i.e., the form in which microprograms are written must be sufficiently readable.

2. Transition from the microprogram to the tables describing the automaton which realizes it must be simple.

Before proceeding to a description of the language, let us suppose that all the various microoperations which might occur in a microprogram have been numbered so that we can denote each microoperation by the number corresponding to it.

A set of microoperations executed simultaneously is a microinstruction and is written in the form of a sequence of microoperation numbers separated from one another by commas.

Let us now proceed at once to a description of the language. A microprogram is written in the form of a sequence of operators separated from one another by a semicolon.

There are four types of operators in all:
1) microinstruction operator,
2) conditional jump operator,
3) unconditional jump operator,
4) end of microprogram operator.

All the microinstruction operators and the end of the microprogram operator are numbered in arbitrary order by the natural numbers starting from one, and these numbers are their labels. Each microinstruction is written out together with its label and separated from it by a colon.

Let us give a formal description of each operator.

1. (microinstruction operator): = (number of microoperation 1 (microinstruction operator), (number of microoperation).


Thus a conditional jump operator is a sequence of branches divided by the symbol V, and enclosed in round brackets, each branch, in its turn, possibly being a conditional jump operator or simply a label.

To each symbol V in a conditional jump operator there corresponds a logical condition which is a Boolean function of certain logical variables, which we shall call elementary conditions. This logical condition is written under the corresponding symbol.

The jump is made to the branch which is immediately to the left of the symbol V, to which the truth condition corresponds. (The truth of a condition is denoted by one.) If all the conditions are false, the jump is made to the extreme right-hand branch.

The simplest conditional jump operator can be written: (P1 V P2) where P1 and P2 are elementary conditions. For example, (15 V 10).
Here if \( p_2 \) is true, a jump is made to the microinstruction with label 15, and if \( p_2 \) is false, then to the microinstruction with label 10.

A more complicated conditional jump operator is

\[
(2 \lor 3) \lor 6.
\]

Here if \( p_2 \lor p_3 = 1 \), then a jump is made to \((2 \lor 3)\), and if \( p_2 \lor p_3 = 0 \), then to the microinstruction labeled 6.

Then if \( (p_2 \lor p_3) p_1 = 1 \), a jump is made to the microinstruction labeled 2 and if \( (p_2 \lor p_3) p_1 = 0 \), to the microinstruction labeled 3.

3. Unconditional jump operator \( : = \) ((label)), i.e., the unconditional jump operator is the label of the microinstruction to which the jump is to be made, enclosed in round brackets.

Unconditional and conditional jump operators destroy the natural order in which operators are carried out, i.e., the order followed when each successive operator in the microprogram representation is carried out after the preceding one.

4. (End of microprogram operator) \( : = \# \).

This operator is written after all the operators which can be executed last in the microprogram and symbolizes all the operations connected with the conclusion of the microprogram.

Let us give an example of a simple microprogram, agreeing to write \( i \) and \( \bar{i} \) instead of the elementary condition symbols \( p_i \) and \( \bar{p}_i \) for the sake of simplicity.

\[
\begin{align*}
1: & 1,5; 2: 4; (3 \lor 7) \lor 7; 3: 5,3; (4 \lor 5); 5: 7; \\
6: & 2,5; 7: 3; (1); 4: 2,6; 8: 3; 9: 8,1; (10 \lor 12); \\
10: & 5,3; 11: 6; (8); 12: \#.
\end{align*}
\]

After the first microinstruction, consisting of microoperations 1 and 5, the second microinstruction, consisting of the single microoperation 4 is carried out, and after it there is a conditional jump, and so on.

We examine the transformation of a conditional jump operator to normal form in the following example:

\[
(((2 \lor 3) \lor (4 \lor 5)) \lor 1) = (2 \lor 3 \lor 4 \lor 5 \lor 1) = 1.
\]

Although the conditional jump operator represented in the form considered above is clear, it is not convenient when we go from a microprogram to tables, and so it is transformed to normal form, defined as follows:

\[
\begin{array}{c|cccccccccccc}
\text{Table 2} \\
\hline
& 0 & 1 & 2 & 3 & 5 & 6 & 7 & 4 & 8 & 9 & 10 & 11 & 12 \\
\hline
1 \lor 1 & 1 & 4 & \times & \times & 2,5 & 3 & 1,5 & 3 & 8,1 & \times & 6 & 3 & \times \\
32 \lor 1 & (1,5)(4) & 5,3 & \times & (2,5)(3) & (1,5)(3)(8,1) & \times & (6)(3) & \times \\
32 \lor 2 & (1,5)(4) & 3 & -- & (3) & (1,5) & -- & (8,1) & \times & (6)(3) & \times \\
5 \lor 1 & (1,5)(4) & 3 & -- & (3) & (1,5) & -- & -- & -- & -- & -- & -- & -- \\
1 \lor 1 & (1,5) & 4 & \times & (2,5)(3) & (1,5) & -- & -- & -- & -- & -- & -- & -- \\
1 \lor 1 & (1,5) & 4 & \times & (2,5)(3) & (1,5) & -- & -- & -- & -- & -- & -- & -- \\
4 \lor 1 & (1,5) & 4 & \times & (2,5)(3) & (1,5) & -- & -- & -- & -- & -- & -- & -- \\
3 \lor 1 & (1,5) & 4 & \times & (2,5)(3) & (1,5) & -- & -- & -- & -- & -- & -- & -- \\
\hline
\end{array}
\]

We examine the transformation of a conditional jump operator to normal form in the following example:

\[
(((2 \lor 3) \lor (4 \lor 5)) \lor 1) = (2 \lor 3 \lor 4 \lor 5 \lor 1) = 1.
\]

We transform the conditional jump operator to normal form by the following equation:

\[
((2 \lor 3) \lor (4 \lor 5)) \lor 1 = ((2 \lor 3) \lor (4 \lor 5) \lor 1) = 1.
\]