called categories. While the patterns in one of these categories may represent all of the possible classes, the criteria for categorizing require mutual "similarity" among members of a category. The number of categories which will be formed can be controlled by altering a parameter in the precise definition of "similar."

Property Finding (Part II): This part of the design procedure considers each category (formed in Part I) separately and treats each in the same manner. The sample patterns of a given category are examined in a formal way to determine and, if necessary, to construct a set of features (properties), which will be useful in classifying the members of this category. A feature is considered useful if it has a high probability of being associated with only one pattern class or if it helps distinguish between two different pattern classes.

Recognition Criteria (Part III): Each category is considered separately, and sample patterns in a given category are classified on the basis of the features obtained in Part II for use with this category. The classification rule assigns a pattern to the class for which its features have the highest percentage match. That is, each pattern class and each pattern has a number of associated features. The normalized number of features a pattern has in common with each pattern class is determined, and the pattern is identified as belonging to that class producing the highest score. (If some of the other scores are uncomfortably close to the highest, additional features are considered to help resolve the difficulty.)

If errors or rejects occur in classifying patterns in the sample set, an iterative procedure attempts to rectify this by changing some of the features and/or assigning a relative weighting to the features when determining the percentage match.

Patterns *not* in the sample set are identified in the same way as patterns belonging to the sample set. That is, the pattern is first placed in the appropriate category, its associated features are determined, and it is identified as belonging to that class for which its features have the highest percentage match.

In my opinion, the weakest part of the design procedure is Part I. The object of this part of the design is to divide the original recognition problem into a number of simpler recognition problems which can be solved one at a time, independently of each other, and whose solutions taken together equal the solution to the original problem.

Aside from the disadvantages of the given categorizing procedure (some of which are pointed out by the author himself), it is questionable whether an initial partitioning step of the *type* suggested is of any value. The justification offered by the author is that control over the number of categories gives the designer a means of trading equipment for performance: both performance and equipment cost are expected to increase as the number of categories increases. However, as indicated in the experimental section of the paper, there is no such simple relationship between equipment cost and number of categories. Even more important, there is no reason to believe that recognition rate, *for patterns not in the original sample set*, will increase with an increase in the number of categories. For example, if the partitioning into categories is based on identity (partitioning based on identity instead of similarity is a situation permitted by the author), the maximum number of categories will result. However, any pattern not identical to one of the sample patterns will be rejected by the system.

The most interesting part of the paper to this reviewer was Part II of the design procedure. By iteratively combining existing properties with simple logical connectives, an attempt is made to improve the usefulness of these properties for representing the various pattern classes. Since the capabilities of most pattern recognition schemes are directly tied to the usefulness of the properties selected by the designer, it has long been recognized that more effort must be expended in finding formal ways to obtain an efficacious set of properties for any given set of patterns. Part II of the design procedure presented in this paper, while certainly not the final answer to this problem, is a step in the right direction.

The final portion of the paper presents experimental results obtained through the use of recognition systems designed by the methods given above. Unfortunately, since the author chose to use the same set of sample patterns for both the design and testing, it is difficult to draw any conclusions about the recognition capabilities of these systems.

In spite of some of the above criticism, it should be stated that this is a clearly written and provocative paper, well worth reading.

MARTIN A. FISCHLER
Lockheed Missiles and Space Co.
Palo Alto, Calif.

F. PROGRAMMING AND PROGRAMMING TECHNIQUES

R63-32 Computational Chains and the Simplification of Computer Programs—Thomas Marill. (IRE TRANS. ON ELECTRONIC COMPUTERS, vol. EC-11, pp. 173–180; April, 1962.)

This paper is a step toward the development of techniques for the formal description and systematic simplification of digital computer programs. The author restricts consideration to *computational chains*, which are intended to represent "straight-line" programs or subprograms having no conditional transfers of control. Computational chains are sequences of statements of three types:

1) *Input statements* of the form $w \rightarrow L_i$; here the symbol w is called an input variable and L_i is called a *storage label*. Such a statement may be interpreted as the command, "Insert the object called w into memory cell L_i."

2) *Output statements* of the form $L_j \rightarrow z$; z is called an output variable. The interpretation is "Output a copy of the object in cell L_j and call this output z."

3) *Computational statements* of the form $f(L_{i_1}, L_{i_2}, \cdots, L_{i_n}) \rightarrow L_k$, with the interpretation, "Obtain from memory copies of the objects in cells $L_{i_1}, L_{i_2}, \cdots, L_{i_n}$; apply the n-ary operator f (whatever it may be) to this ordered n tuple, and store the result in cell L_k."

The paper is concerned with properties of computational chains which may be asserted to hold regardless of the nature of the operators occurring in computational statements; it is assumed, however, that distinct operators have distinct names. A computational chain is said to be *well formed* if every storage label occurring on the left-hand side of a statement occurs on the right-hand side of an earlier statement. A well-formed computational chain specifies each of its output variables as a composite function of its input variables. Two well-formed computational chains which have the same output variables specified as the same functions of input variables (regardless of the interpretation of the operators in computational statements) are considered equivalent. A statement is called *vacuous* if its removal from a computational chain yields an equivalent chain; a *redundancy* is said to exist if two computational statements yield the same functions of the same input variables. Techniques are given for identifying and removing vacuous statements and redundancies.

The author considers the problem of finding, in an equivalence class of computational chains, a chain having a minimum number of distinct storage labels. He shows how to obtain a minimal labelling given the order of occurrence of the statements, but does not solve the complete problem, which involves consideration of the permissible permutations of the statements. It appears to the reviewer that this problem can be solved using the techniques of dynamic programming.

The final problem considered is that of constructing a composite computational chain C_3 which represents a computation equivalent to the successive application of given computational chains C_1 and C_2.

The author does not give a reference to an interesting paper[1] which develops quite elegantly the syntax of parenthesis-free expressions which are in correspondence with a subclass of the class of computational chains; for this subclass, the storage label minimization problem has a simple solution based on the concept of tail weight.[1]

From the viewpoint of mathematical logic, the logistic system which the author presents is too primitive to yield deep results. But the problems of temporary storage minimization which can be treated within such a framework are surely of interest to programmers concerned with writing efficient compilers.

RICHARD M. KARP
IBM Research Center
Yorktown Heights, N. Y.

[1] A. W. Burks, D. W. Warren and J. B. Wright, "An analysis of a logical machine using parenthesis-free notation," *Mathematical Tables and Aids to Computation*, vol. VIII, pp. 53–57; April, 1954.

G. SEQUENTIAL SWITCHING CIRCUIT THEORY

R63-33 Introduction to the Theory of Finite State Machines—Arthur Gill. (McGraw-Hill Book Co., New York, N. Y. xi pp., 198 pp., plus 4 bibliography pp., plus 5 index pp., $9.95.)