Canonical Structure in Attribute Based File Organization

Eugene Wong and T.C. Chiang
University of California,* Berkeley

A new file structure for attribute based retrieval is proposed in this paper. It allows queries involving arbitrary Boolean functions of the attribute-value pairs to be processed without taking intersections of lists. The structure is highly dependent on the way in which the file is to be used and is uniquely determined by the specification of the allowed queries. Thus, for example, the structure for retrieval on the basis of ranges of values of a given attribute would be very different from one where only retrieval on the basis of a single value is permitted.

The file organization being proposed is based on the atoms of a Boolean algebra generated by the queries. The desirable properties claimed for this structure are proved, and file maintenance questions are discussed.

Key Words and Phrases: address calculation, atoms of Boolean algebra, attributes, Boolean functions, Boolean queries, file organization, information retrieval, inverted file, key words, multilist, queries, searches

CR Categories: 3.70, 3.73, 3.74

1. Introduction

A basic goal of file organization is to enable a large body of information to be accessible via its content. This is almost always done by creating a secondary body of information which in some sense reveals the content of the file. For a large class of file organization techniques this secondary body of information consists of lists of accession numbers or addresses of records. Each list, in turn, consists of the addresses of all the records in the file having some specified common property. For example, for a file of personnel records the collection of all employees with a Ph.D. in electrical engineering may be such a list.

The lists of accession numbers or addresses which are generated can be stored separately from the main file in a directory such as in inverted file structures, or threaded through the main file by means of pointers as in multilist type of information storage. Hsiao and Harary [1] have elucidated the whole spectrum of alternatives which are possible by proposing the "generalized file structure" which include multilist and inverted-file as extreme cases. In this paper we shall address outselves to the complementary problem of deciding what lists to generate in the first place. Of course, the secondary information that needs to be generated must depend on the nature of the main file on the one hand and on the addressability that we demand of it on the other. The aim of this paper is to make clear this dependence and to arrive at an optimal structure.

In a file structure where some lists of addresses are available, retrieval of all records belonging to a single list is immediate, but retrieval of records corresponding to a Boolean function of the lists is in general time-consuming and difficult. We have found a file structure which has some highly desirable properties. The lists of addresses to be stored in this structure correspond to atoms of the collection of all retrievable sets. As such,

* Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory. This research was sponsored by the Joint Services Electronics Program, Grant AFOSR-68-1488, and the U.S. Army Research Office, Durham Contract DAHCO4-67-C-0046.

they are pairwise disjoint so that no intersection ever needs to be performed. They are complete in the sense that every retrievable set corresponds to a union of these lists. Because they are generated by the specified retrieval requirement, they are also efficient. In particular, retrieval involving ranges of values, which is time-consuming in many file structures, is handled in a natural way in this structure.

2. File Structure

We shall adopt, with some simplification, the model of Hsiao and Harary for an unformatted file. A *record* is a finite collection of attribute-value pairs. We permit the broadest interpretation of the terms "attribute" and "value." For example, for the record of a published paper, typical attributes include *author*, *title*, *journal of publication*, etc., but *content* can also be an attribute, the value of which is then the entire text of the paper. Each record is assigned a unique positive integer called its *address*. A *file* is a finite collection of distinct records. We shall only consider the organization problem involving a single file.

Access to the records is via some of the attribute-value pairs. Let $\mathcal{S}$ be a fixed collection of some of the attribute-value pairs appearing in the file. An attribute-value pair appearing in $\mathcal{S}$ will be called a *keyword*. We assume that the file can be interrogated by keywords only and every query is a Boolean function of keywords.

For example, let K_1 equal Author-Hemingway and K_2 equal classification-fiction. Then $K_1 \wedge \bar{K}_2$ represents a request for all the nonfiction works by Hemingway in the file. We assume that every record has at least one keyword. This is reasonable, since otherwise this record can never be retrieved. However, we do not assume that each record is uniquely identified by its keywords. Two records having identical keywords will always be retrieved together.

For each keyword K_i we denote by $R(K_i)$ the set of all records containing K_i. We denote the list of addresses of the records in $R(K_i)$ by $A(K_i)$.[1] In an inverted file structure the lists $A(K_i)$ are stored separately from the main file in a directory. In a multilist structure only the first address of each list $A(K_i)$ is stored in the directory, and the remainder of the list is threaded through the main file via pointers. For example, let a_{ij} denote the jth address in $A(K_i)$. The directory contains a_{i1}, the record at a_{i1} will contain the keyword K_i together with the next address a_{i2} on the list $A(K_i)$. The record at a_{ij} contains the pair (K_i, a_{ij+1}), and at the last address on $A(K_i)$ the record contains $(K_i, 0)$ to denote "end of list." Thus, in a multilist structure, the keywords in each record are augmented by pointers pointing to the next record on the list. Hsiao and Harary have proposed a generalized structure which amounts to breaking up each $A(K_i)$ into sublists, keeping the first address in each sublist in the directory, and threading the remainder of each list in the main file as in the multilist structure.

File structures which involve storing the lists $A(K_i)$ in some form are best suited for retrieval involving a single keyword. Multiple-keyword retrieval involves taking intersections, unions and complementations of the lists $A(K_i)$. In special applications where certain conjunctions of keywords occur frequently in queries, it has been proposed to store the corresponding lists obtained by taking intersections, in addition to the $A(K_i)$. This ad hoc procedure has obvious limitations. In general, one cannot store a list for every query that is anticipated. A different approach involving finite geometry has also been proposed for multiple-keyword retrieval [2, 3]. However, the full advantage of such an approach is realized only under rather special circumstances.

3. Transformation of Keywords

Let Ω be a finite set. Let $\mathcal{C}$ be a class of subsets of Ω. A *Boolean set operation* is any finite sequence of intersections ($\cap$), unions ($\cup$) and complementation ($^-$). If $\mathcal{C}$ is closed under complementation and pairwise union, i.e. $A \in \mathcal{C}$, $B \in \mathcal{C} \Rightarrow A \cup B \in \mathcal{C}$, $\bar{A} \in \mathcal{C}$, and $\bar{B} \in \mathcal{C}$, then $\mathcal{C}$ is called a *Boolean algebra*. A Boolean algebra is closed not only under complementation and union but also under all Boolean set operations, since $A \cap B$ can be reexpressed as $\overline{\bar{A} \cup \bar{B}}$. If $\mathcal{C}$ is an arbitrary class of subsets of Ω then there is a *smallest* Boolean algebra containing $\mathcal{C}$. We call this minimal algebra the Boolean algebra generated by C and denote it by $\mathfrak{B}(\mathcal{C})$ [4].

Let $\mathcal{F}$ be the file, i.e. the collection of all records under consideration. Let $\mathcal{S} = \{K_i, i = 1, \ldots, n\}$ be the set of all keywords. Under the assumption that every record contains at least one keyword, we have

$$\mathcal{F} = \bigcup_{i=1}^{n} R(K_i).$$

A keyword K_i is said to be *true* for a record r_j if r_j contains K_i. It follows from the rules of propositional calculus that each Boolean function $f(K_1, \ldots, K_n)$ is either true or not true for each record. Since we have defined $R(K_i)$ as the set of all records for which K_i is true, $R(K_i) \cap R(K_j)$ is the set of records for which $K_i \wedge K_j$ is true; $R(K_i) \cup R(K_j)$ is the set for which $K_i \vee K_j$ is true, and so forth. It will be assumed that a query is a Boolean function of keywords; hence, the set of records that are to be retrieved for a query is always a Boolean set operation on $R(K_i)$, $i = 1$, $i = 1, \ldots, n$. Therefore, if we denote $\mathcal{R} = \{R(K_i), i = 1, \ldots, n\}$ then $\mathfrak{B}(\mathcal{R})$ is

[1] If the records are short, a possible alternative is to store $R(K_i)$ for each K_i. In general, this clearly involves redundant storage of some of the records.

just the collection of all possible sets of records that can be retrieved. Each set in $\mathfrak{B}(\mathfrak{R})$ corresponds to one and only one. Boolean function of keywords, which by definition is a query.

Since $\mathfrak{R}$ generates $\mathfrak{B}(\mathfrak{R})$, every set in $\mathfrak{B}(\mathfrak{R})$ can be obtained by Boolean set operations on sets in $\mathfrak{R}$. Therefore, if we can retrieve every set in $\mathfrak{R}$, i.e. every $R(K_i)$, then we can retrieve every set in $\mathfrak{B}(\mathfrak{R})$. This is precisely why by storing the lists of addresses corresponding to the sets in $\mathfrak{R}$ we can retrieve every set in $\mathfrak{B}(\mathfrak{R})$. Of course, instead of $\mathfrak{R}$, any collection $\mathfrak{C}$ of subsets which generates $\mathfrak{B}(\mathfrak{R})$ can serve the same function provided that the union of the sets in $\mathfrak{C}$ is the entire file $\mathfrak{F}$. In other words the lists of addresses that we store need not correspond to $R(K_i)$, $i = 1, \ldots, n$. Alternatively, we can store lists of addresses, each list of addresses corresponding to a set in a collection $\mathfrak{C}$ which generates $\mathfrak{B}(\mathfrak{R})$. Naturally, the flexibility thus provided should be taken advantage of in the organization of the file. The basic question is: "Is there an optimal $\mathfrak{C}$?" While optimality depends on the criterion one chooses, we hope to show that the collection of *atoms* of $\mathfrak{B}(\mathfrak{R})$ has strong claims in that regard.

If $\mathfrak{B}$ is a Boolean algebra of subsets of $\mathfrak{F}$, a set $B \in \mathfrak{B}$ is said to be an *atom* of $\mathfrak{B}$ if it is nonempty, and no nonempty proper subset of B is in $\mathfrak{B}$. Thus, atoms are irreducible units of a Boolean algebra. For a simple example, suppose that $\mathfrak{F} = \{1, 2, 3, 4\}$ and there are two keywords K_1 and K_2 with $R(K_1) = \{1, 2, 3\}$ and $R(K_2) = \{3, 4\}$. Then, $\mathfrak{B}$ is the collection of following sets: ϕ (the empty set), $\{1, 2\}$, $\{3\}$, $\{4\}$, $\{1, 2, 3\}$, $\{3, 4\}$, $\{1, 2, 3, 4\}$. The atoms are: $\{1, 2\}$, $\{3\}$ and $\{4\}$. The atoms of the Boolean algebra $\mathfrak{B}(\mathfrak{R})$ can be generated from $R(K_i)$, $i = 1, \ldots, n$, in a systematic way as shown in the following theorem.

THEOREM 1. *Let $R(K_i)$, $i = 1, \ldots, n$, be subsets of $\mathfrak{F}$ such that $\bigcup_{i=1}^{n} R(K_i) = \mathfrak{F}$ and let $\mathfrak{B}(\mathfrak{R})$ denote the Boolean algebra generated by $\mathfrak{R} = \{R(K_i), i = 1, \ldots, n\}$. Let $C_1, C_2, \ldots, C_{2^n}$ be the 2^n intersections of the form $\bigcap_{i=1}^{n} \tilde{R}(K_i)$, where $\tilde{R} = R$ or $\bar{R}$. Let C_i be so numbered that $C_1, C_2, \ldots, C_m$ are nonempty while $C_{m+1}, C_{m+2}, \ldots, C_{2^n}$ are empty. Then,*

(a) C_j and C_k are disjoint whenever $j \neq k$.
(b) $B \in \mathfrak{B}(\mathfrak{R})$ implies $B \cap C_j = \phi$ or C_j for every j.
(c) Every $B \in \mathfrak{B}(\mathfrak{R})$ is a union of some of the C_i's
(d) $\{C_1, C_2, \ldots, C_m\}$ are the atoms of $\mathfrak{B}(\mathfrak{R})$.

Remark. Although the assertions of the theorem are standard results, we shall reproduce the proof here for completeness.

PROOF.

(a) We write C_j in the form

$$C_j = \bigcap_{i=1}^{n} \tilde{R}_j(K_i), \tilde{R}_j = R \text{ or } \bar{R}.$$

If $j \neq k$ then there is at least one i for which $\tilde{R}_j(K_i)$ is the complement of $\tilde{R}_k(K_i)$ so that $C_j \cap C_k = \phi$.

(b) For each i and j

$$R(K_i) \cap \tilde{R}_j(K_i) = \phi \text{ if } \tilde{R}_j = \bar{R}$$
$$= \tilde{R}_j(K_i) \text{ if } \tilde{R}_j = R.$$

Therefore, for each i and j, $R(K_i) \cap C_j = C_j$ or ϕ. The same is true for $\bar{R}(K_i) \cap C_j$. For $R(K_i) \cap R(K_k)$ we have

$$R(K_i) \cap R(K_k) \cap C_j = R(K_i) \cap (C_j \text{ or } \phi)$$
$$= C_j \text{ or } \phi.$$

It follows that for every $B \in \mathfrak{B}(\mathfrak{R})$, $B \cap C_j = C_j$ or ϕ.

(c) We note that

$$\mathfrak{F} = \bigcup_{i=1}^{n} R(K_i) = \bigcup_{i=1}^{m} C_i.$$

Hence, we can write for $B \in \mathfrak{B}(\mathfrak{R})$

$$B = B \cap \mathfrak{F} = \bigcup_{i=1}^{m} B \cap C_i.$$

Since each $B \cap C_i$ is either C_i or ϕ, every B in $\mathfrak{B}(\mathfrak{R})$ is a union of the C_i's for which $B \cap C_i = C_i$.

(d) Each C_i, being a Boolean combination of $R(K_i)$, is in $\mathfrak{B}(\mathfrak{R})$. Since each set in $\mathfrak{B}(\mathfrak{R})$ is a union of the C's, no proper subset of any C_i can be in $\mathfrak{B}(\mathfrak{R})$. Thus, every C_i is an atom of $\mathfrak{B}(\mathfrak{R})$. On the other hand, any atom of $\mathfrak{B}(\mathfrak{R})$ is nonempty so that it is a union of one or more C_i, $i = 1, \ldots, m$. It cannot be a union of more than one C_i because then it would contain nonempty proper subsets which are in $\mathfrak{B}(\mathfrak{R})$. Thus, every atom of $\mathfrak{B}(\mathfrak{R})$ is one of the C_i, $i = 1, \ldots, m$. ∎

We now propose a file structure in which we store the lists of addresses corresponding to the atoms C_i, $i = 1, 2, \ldots, m$, instead of storing the keyword lists $A(K_i)$, $i = 1, \ldots, n$. Since each C_j corresponds to a Boolean function of the keywords, we can regard this process as one of transforming the keywords. The advantages of this structure include the following:

(a) Each address appears on one and only one list. Hence, the number of addresses to be stored is always less than the total number of addresses in $\{A(K_i), i = 1, \ldots, n\}$.

(b) Every set to be retrieved is a union of disjoint atoms. We never need to take intersection, and we never need to eliminate duplications in taking union.

(c) The computation procedure in translating an arbitrary Boolean function of keywords into a union of atoms is exceedingly simple

Assertions (a) and (b) are obvious consequences of Theorem 1. We now justify assertion (c). A Boolean function $f(K_1, K_2, \ldots, K_n)$ can always be expressed in a *disjunctive normal form* as the disjunction of clauses each *clause* being the conjunction of some of the K_i and $\overline{K_i}$. For example,

$$f(K_1, K_2, K_3, K_4) = (K_1 \wedge K_2 \wedge \overline{K_4}) \vee (K_2 \wedge \overline{K_3} \wedge K_4)$$

is a Boolean function in disjunctive normal form. A disjunctive normal form is said to be *developed*, if every variable appears once and only once in every clause either unnegated or negated (never both). If a variable

K_i does not appear in a clause ϕ then by replacing ϕ by $(\phi \wedge K_i) \vee (\phi \wedge \overline{K_i})$, we have obtained clauses containing the variable K_i. Therefore, a disjunctive normal form can always be developed by successive applications of this procedure [5]. A Boolean function $f(K_i, \ldots, K_n)$ expressed in a developed disjunctive normal form is of the form

$$f(K_1, \ldots, K_n) = \bigvee_{j \leq n} \left(\bigwedge_{i \leq n} \tilde{K}_{ji} \right)$$

where each $\tilde{K}_{ji}$ is either K_i or $\overline{K_i}$. The set of all records for which f is true is precisely

$$R(f) = \bigcup_{j \leq n} \bigcap_{i \leq n} R_j(\tilde{K}_{ji}) = \bigcup_{j \leq n} \bigcap_{i \leq n} \tilde{R}_j(K_i).$$

We recognize immediately that for each j $\bigcap_{i \leq n} \tilde{R}_j(K_i)$ is either an atom or it is empty. Hence, each nonvoid clause in a developed disjunctive normal form corresponds to an atom, and once a Boolean function is expressed in a developed disjunctive normal form, the corresponding set of records is automatically in the form of a union of atoms.

For an example, consider a file with 10 records and 4 keywords. For the purpose of this example, we do not distinguish between a record and its address. Let the keywords be denoted by K_1, K_2, K_3, K_4 and let the records be denoted by 1, 2, 3, ..., 10. Suppose that $R(K_i)$, $i = 1, \ldots, 4$, are given as follows:

$R(K_1) = 1, 2, 4, 5, 7, 8, 10$
$R(K_2) = 2, 7, 10$
$R(K_3) = 1, 4, 5, 8$
$R(K_4) = 3, 5, 6, 8, 9$

The atoms become obvious if we reexpress the sets $R(K_i)$ in a tabular form (see Table I) when an entry "1" means the record belongs to the set $R(K_i)$, and "0" means it does not. Reading the rows of the table, we can immediately write down the atoms as follows:

$C_1 = \bar{R}(K_1) \cap \bar{R}(K_2) \cap \bar{R}(K_3) \cap R(K_4) = (3, 6, 9)$
$C_2 = R(K_1) \cap \bar{R}(K_2) \cap R(K_3) \cap \bar{R}(K_4) = (1, 4)$
$C_3 = R(K_1) \cap \bar{R}(K_2) \cap R(K_3) \cap R(K_4) = (5, 8)$
$C_4 = R(K_1) \cap R(K_2) \cap \bar{R}(K_3) \cap \bar{R}(K_4) = (2, 7, 10)$

We have numbered the C_j's in ascending order of the binary expansion represented by the rows in Table I, but this is entirely arbitrary.

Now, consider a Boolean function.

$$f(K_1, K_2, K_3, K_4) = (K_1 \wedge K_2 \wedge \overline{K_4}) \vee (K_2 \wedge \overline{K_3} \wedge K_4)$$

We can develop the formula by rewriting it as

$$\begin{aligned} f(K_1, K_2, K_3, K_4) = {} & (K_1 \wedge K_2 \wedge K_3 \wedge \overline{K_4}) \\ & \vee (K_1 \wedge K_2 \wedge \overline{K_3} \wedge \overline{K_4}) \\ & \vee (K_1 \wedge K_2 \wedge \overline{K_3} \wedge K_4) \\ & \vee (\overline{K_1} \wedge K_2 \wedge \overline{K_3} \wedge K_4). \end{aligned}$$

Of the four clauses, only $(K_1 \cap K_2 \cap \overline{K_3} \cap \overline{K_4})$ corresponds to an atom, viz., $C_4 = (2, 7, 10)$. Hence, the set to be retrieved for $f(K_1, K_2, K_3, K_4)$ is just $(2, 7, 10)$.

Table I.

	$R(K_1)$	$R(K_2)$	$R(K_3)$	$R(K_4)$
1	1	0	1	0
2	1	1	0	0
3	0	0	0	1
4	1	0	1	0
5	1	0	1	1
6	0	0	0	1
7	1	1	0	0
8	1	0	1	1
9	0	0	0	1
10	1	1	0	0

4. Extensions

There are many instances where certain keywords are never used standing alone. For example, an attribute-value pair like classification-fiction, if used in a query by itself, might well bring forth half of the file. For an example of another type, consider an attribute-value pair like (salary-$14608.25 per year) in a personnel file. It is unlikely that there will be a query asking for all employees earning exactly $14,608.25 per year. A query on salary will more likely be in the form of "find all employees with salary between $14,000 and $15,000 per year." While such a query is expressible as a union of some 10^5 keywords, each of form "salary-$14,xxx.xx," this is hardly a reasonable solution to the problem.

The file structure proposed in the last section is flexible enough to accommodate such situations. Suppose that instead of permitting any Boolean function of the keywords to be a query, only certain Boolean functions are allowed. We note that a single keyword K_i may or may not be a permissible query. For example, we may only allow keywords involving salary to appear in unions spanning $1000 intervals. Each allowable Boolean function corresponds to a set of records in the file. Let $\mathcal{C}$ denote the collection of all sets of records corresponding to allowable Boolean functions. Let $\mathcal{B}(\mathcal{C})$ denote the Boolean algebra generated by $\mathcal{C}$. In general $\mathcal{B}(\mathcal{C})$ is smaller than the algebra $\mathcal{B}(\mathcal{C})$ generated by the individual keywords. The atoms of $\mathcal{B}(\mathcal{C})$ give a coarser partition of the file than the atoms of $\mathcal{B}(\mathcal{R})$. This is precisely what is wanted. Lists of addresses corresponding to the atoms of $\mathcal{B}(\mathcal{C})$ will now be stored and made use of in retrieval. We note that in this way we can answer queries involving ranges of attribute values without taking the union of a large number of sets, provided the increments of the range can be specified a priori.

Since only sets in $\mathcal{C}$, rather than $\mathcal{B}(\mathcal{C})$, are ever re-

trieved, one might ask whether even a better scheme exists. The answer is "no" in the following sense: One can show that the atoms of $\mathfrak{B}(\mathfrak{C})$ give the coarsest partition of the file such that every set in $\mathfrak{C}$ is a union of the subsets of the partition. Furthermore, if the union of the sets in $\mathfrak{C}$ is the entire file $\mathfrak{F}$, then every atom of $\mathfrak{B}(\mathfrak{C})$ appears in at least one set of $\mathfrak{C}$. If the union of sets in $\mathfrak{C}$ is not $\mathfrak{F}$, then there is one atom (viz., $\mathfrak{F}$ minus union of $\mathfrak{C}$) which will not be used and it can be deleted from the lists to be stored.

5. Address Calculation

Thus far, we have emphasized the use of atoms in constructing a directory of address lists. Actually, the main advantage of a file structure based on atoms may well be that a directory is no longer necessary. One of the reasons for maintaining a directory of address lists in an inverted file structure is to permit set manipulations to be done with ease. Since atoms are disjoint, set manipulations are no longer necessary. If the file is so organized that records belonging to a single atom are stored together, then the procedure for representing a Boolean function in a developed normal form yields a means for direct address calculation from the queries.

The number of atoms can be much larger than the number of generating sets. In the extreme case, the number of atoms is equal to the number of records, each atom being a single record in that event. If the number of atoms is large, searching a directory of addresses may be time-consuming, although it is no more so than searching an address table generated by any *identifying* attribute (i.e. an attribute which has a unique record corresponding to each of its values). For such situations the ability to bypass the directory, afforded by organizing the main file according to atoms, is especially valuable. We note that the query-directed address calculation scheme, suggested by the file structure based on atoms, works well even when every atom consists of just a single record. In that case the correspondence, mapping records to atoms, represents an identifying attribute, and the structure that we have proposed can be viewed as reducing the general Boolean retrieval problem to one involving a single identifying attribute.

In practice, it may well be useful to consider a two-tier system in which the main file is organized to correspond to the atoms of the Boolean algebra generated by all queries, but the directory is organized to correspond to the atoms of the Boolean algebra generated by a small subset of the queries. This would allow some frequently occurring queries to receive special treatment.

6. File Maintenance

In general, the structure proposed here is not more difficult to update than the usual structure. It will often be easier. For example, each record belongs to one and only one list so that the addition or deletion of a record requires only knowing which atom it belongs to. If the atoms are generated by keywords then the atom to which a record belongs is simply that corresponding to the conjunction of all keywords in the record and the negation of all keywords not in the record. For example, if K_1, K_2, K_3, K_4 are the keywords and if r_1 has K_1 and K_4 but not the others, then r_1 must be in the atom $R(K_1) \cap R(K_2) \cap R(K_3) \cap R(K_4)$. If the atoms are generated by a general set of Boolean functions of keywords, as in the case discussed in Section 4, the situation is more complicated. To discover which atom a given record is in we have to determine which of the Boolean functions in the generating set are true and which are false for this record. The desired atom is then found by taking the conjunction of all the generating Boolean functions which are true together with the complement of all the generating Boolean functions which are false.

Updating which involves changes in keywords (more generally, changes in the generating set of Boolean functions of keywords) is more difficult. Additions involve breaking up of some of the atoms, and deletions involve coagulation of some of the atoms. Procedure for doing so is routine but may be time-consuming.

Received November 1970; revised February 1971

References

1. Hsiao, D., and Harary, F. A formal system for information retrieval from files. *Comm. ACM 13*, 2 (Feb. 1970), 67–73.
2. Abraham, C.T., Ghosh, S. P., and Ray-Chaudhuri, D. K. File organization schemes based on finite geometrics, *Inform. Contr. 12* (1968), 143–163.
3. Chow, D.K., New balanced-file organization schemes. *Inform. Contr. 15* (1969), 377–396.
4. Halmos, P.R. *Measure Theory*. Van Nostrand, Princeton, N.J., 1950.
5. Quine, W.V., The problem of simplifying truth functions. *Amer. Math. Monthly 59* (1952), 521–531.