Technical Report CS 75012R

TOWARD A FUNCTIONAL REPRESENTATION OF A GENERALIZED DOCUMENT INFORMATION STORAGE AND RETRIEVAL SYSTEM

Richard E. Nance†
Carolyn J. Crouch

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

Revised
June 1974

*A substantial part of this work was done at Southern Methodist University and is described in an earlier technical report.

†The authors express their gratitude to Robert R. Korfhage for his many suggestions during the course of this research.

ABSTRACT

A model of a generalized document storage and retrieval system is proposed. The model consists of six subsystems (or blocks): *logical processor*, *selector*, *descriptor file*, *locator*, *document file* and *analysis* block. These subsystems function in a partial environment defined by the *user* and *data* blocks. Proceeding from a verbal description, a functional representation of each subsystem is developed. The functional representation describes not only <u>what</u> is done but also, to some degree, <u>how</u> tasks are accomplished within each subsystem. An immediate result of the functional representation is the definition of a metalanguage for identifying some necessary characteristics of higher level languages used in the implementation of information storage and retrieval systems. To illustrate the usefulness of the metalanguage, a comparison is made of three languages - FORTRAN, PL/1, and SNOBOL - for implementing document storage and retrieval systems. The functional differences among the blocks of the system are apparent, and the implementation of efficient systems appears to require a multi-language approach.

INTRODUCTION

Lack of a recognized, well accepted theory of information retrieval has provided a constant disturbance to some workers in this field. In the Fall, 1966 issue of the Forum (the newsletter of the Special Interest Group on Information Retrieval), Lauren Doyle [5] refers to the "social turmoil" created by use of the term "information retrieval". This upheaval stems, as he notes, from the inability of people to accept a common definition of the term. With such disagreement on the definition of "information retrieval", a more disparate perception of what is encompassed by the field is a natural consequence. Recognition and acceptance of a theory is believed by some [3] to offer some hope for reducing this diversity of views. We admit our membership in this optimistic group, and our purpose is to attempt a small step in the path toward establishment of some fundamental principles.

The fundamental description offered in this paper is not proposed as a theory; rather, we seek to identify an approach by which a theory could evolve. Characteristic of this approach are the dual objectives: (1) descriptiveness and (2) generality. Descriptiveness is necessary if we are to evolve an accepted theory, i.e. one contributing to "theory users" [3]. Generality or the integration of several seemingly distinct entities, characteristics, and/or methods into a single conceptual unit, is a requirement of any theory, but we are determined to avoid the usual corequisite - abstraction. Abstraction may prove necessary in subsequent stages of development, but our present work is based on the practical objective of describing the functions performed within an information retrieval system.

RELATED WORK

Several authors have proposed theories of information retrieval or documentation, and we survey only the recent attempts that include the perspective of automatic information retrieval systems. A more comprehensive treatment, exploring various subdisciplines and techniques of mathematics applied in the modeling of information retrieval systems and subsystems, is given by Hayes [7]. His purpose is to identify the role and contribution of mathematical models rather than to develop a theory of information retrieval.

Most theories of information retrieval (IR) begin with a specific aspect of the total problem. Jonker [9] offers a theory that deals primarily with the classification or indexing aspect. His idea of the descriptive continuum is that the existing indexing systems form a continuum based on the average length of index terms. This continuum would have at one extreme the indexing systems using single word terms; at the other extreme are the hierarchical classification schemes in which the longest possible terms are used. Since the cost of an IR system is largely dependent on the indexing task, Jonker [9, p. 1311-1312] argues that total system costs are reflected in the position of an indexing system on the continuum. More recently, Soergel [12] proposes a formal system representation of documentation systems in terms of the classification and query search functions. Using primarily a set-theoretic approach, Soergel is able to construct a classification of IR systems based on the relationships among descriptor and query components, i.e. indexing terms in the former case and query terms in the latter. Turski [15] proposes a model of an IR system focused on a formal development of the thesaurus concept.

In his recent text Salton [11] summarizes three approaches to modeling IR systems. From these models certain theoretical relationships can be derived. One approach is based on the search function, i.e. the relationship between the specified set of query terms and the resulting document set retrieved. An IR system (I) is defined by the triple

$$I = (D, R, T)$$

where

D is the finite set of documents,
R is the request language (finite set of request terms),
T is a function mapping R into all possible subsets of D.

Given the requests r and s from a partially ordered set R, and if the ordering ($\leq$) is such that $s \leq r$, then the retrieval function T: $R \rightarrow 2^D$ defines an inclusive retrieval system if

$$s \leq r \rightarrow T(r) \subseteq T(s).$$

A second approach is to model the IR system with respect to the classification function. This approach, stemming from the earlier work of Mooers [10], defines an IR system as

$$\mathcal{I} = (D,R,C,X,F)$$

where in addition to the document set (D) and request language (R)

C is the classification language,
X is the classification function, i.e. X: $D \rightarrow C$, and
F is a function mapping the request language (set) into all possible subsets of the classification language, i.e. F: $R \rightarrow 2^C$.

The retrieval function T: $R \rightarrow 2^D$ is then defined in terms of the functions X and F, i.e. given the request r the set of documents d returned is

$$T(r) = \{d \mid X(d) \in F(r)\}.$$

Mooers uses the basic concepts above to classify IR systems: (1) descriptors (association of a set of terms with each document), (2) characters with hierarchy (an hierarchical classification scheme), and (3) characters with logic (characters combined by logical operations). Mooers [10, p. 1332] defines a character as a verbal symbol which (a) can be independently manipulated, (b) is primitive (non-decomposable), (c) has definite meaning, and (d) is from a finite repertory.

A third approach discussed by Salton uses graph theory as the modeling technique.

Other authors have chosen to avoid mathematical developments of an IR theory and preferred to concentrate on formulating the fundamental problems. For example, Swanson [13] gives a thought-provoking discussion of the several subproblems - indexing, file organization, and performance requirements - comprising the general IR problem.

THE FUNCTIONAL LANGUAGE APPROACH

Concern with the languages of information retrieval has been demonstrated by at least three authors. Dolby [4] reviews the population of programming languages and discusses their relative capability for IR applications. He concentrates on assembly languages, COBOL, PL/I, and several special purpose, primarily string and list processing, languages. Vickery [18] relates the function of an IR language to the indexing and search tasks, providing a description of functions that may be performed in some particular systems. Fairthorne [6] proposes an algebraic representation of IR languages that seeks to describe relationships among terms in the system vocabulary.

Our approach is to propose a model of a generalized IR system.[1] The model is comprised of six subsystems with distinct functions. We use well defined mathematical operations to represent these subsystem functions. One result of this functional representation is to define a <u>metalanguage</u> describing not only <u>what</u> happens within the IR system but, to some degree, <u>how</u> it happens. In this respect we differ from previous approaches but at the potential expense of sacrificing generality. In this effort we have emphasized descriptiveness.

[1] We have shown the model to represent adequately four IR systems described in the literature, QUERY, GIPSY, BIRS and SMART [2].

FUNCTIONAL REPRESENTATION

One difficulty in developing a theory of information retrieval is the lack of a well defined, completely comprehensive, existing system. In contrast with the physical sciences, no entity is available for our examination. Consequently, the comparison of the theory with the physical process, i.e. retrieving information, is impossible. Thus, as Soergel [12, p. 170] notes, we must begin with a preconceived model around which the theoretical framework can be structured. We propose a generalized model of an IR system, identifying the six subsystems and the environment in which the total system functions, i.e. the user and data populations. Each subsystem is called a "block" (or module), and the user and data populations also constitute "blocks". The blocks are examined independently, and each subsystem is represented in terms of the language requirements for implementing that block.

The Generalized Model

Figure 1 shows the generalized model of an information storage and retrieval system (IR system) proposed in the earlier work by Crouch [2]. The structural similarity to models proposed by other authors, notably Vickery [16], is acknowledged. In developing the representation of the IR system, we concentrate on the functions executed by or within each subsystem (the rectangular blocks). Together the user and data blocks serve to define the partial environment in which the system operates. The total environment would include the funders or operators of the system with considerations of policy and economics of operation. A brief description of the relationships among the blocks follows.

The *user* (generally assumed to be unfamiliar with mechanized ISR systems or digital computers) inputs its query to the system. The query is taken by the *logical processor* which operates on the query and outputs to the *selector* the query in terms of descriptors or index terms. The *selector* uses the descriptors

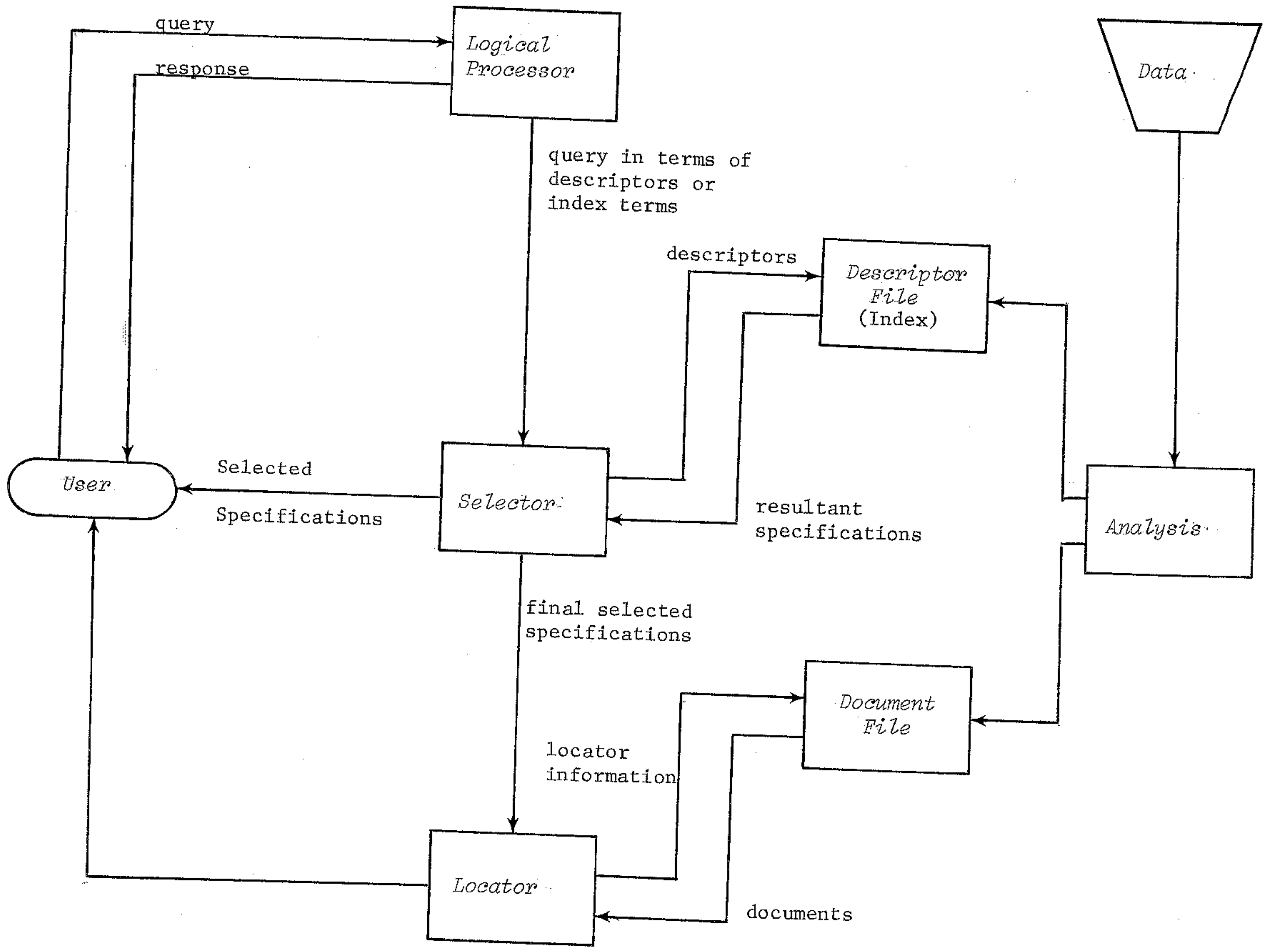


Figure 1. Model of a Generalized Information Storage and Retrieval System

to search the *descriptor file* (or index). The resultant specifications, i.e., pointers to those documents which have successfully satisfied the search according to some pre-established criteria, are returned to the *selector*. The *selector*, which may or may not operate on these specifications, sends the final selected specifications to the *locator*, which uses this information to search the *document file*. The documents themselves are returned first to the *locator* and from there to the *user*.

The second part of the environment definition is the *data*. *Data* enters the system at the *analysis* block. The *analysis* block operates on the input to produce two outputs--a representation of the document in terms of descriptors, to be stored in the *descriptor file* along with a pointer to the document in the *document file*, and a reference to the document itself (i.e., an identifier) to be stored in the *document file*.

Note the three feedback loops involving the *user*:

(1) from the *user* to the *logical processor* and back to the *user*,

(2) from the *user* to the *logical processor* and *selector* then back to the *user*, and

(3) from the *user* to the *logical processor*, *selector*, and *locator*. then back to the *user*.

In the first case, the *logical processor* is asking the *user* to re-formulate, clarify, or augment his query. In the second case, the *selector* is requesting *user* approval of the selected specifications, i.e. for the *user* to designate from amongst the set those that most accurately describe his needs.

We assume that all information stored within the system enters through the *analysis* block; thus any information concerning the *user*, his use of the system, or resulting from this use must be viewed as imput to the *analysis* block.

Description of the Environment

The environment is described by the *user* and *data* blocks. Economic aspects of system operation are ignored; so that we actually describe a partial environment. Our assumptions about this environment are limited. We consider that the *user* is motivated by a need for information and interacts with the IR system in his attempt to satisfy this need. Perhaps being quite unknowledgeable of the system structure and/or capability; nevertheless, he is able to supply the initial character string in interaction with the system. We designate the input query Y to be the set of all strings initially used to describe the *user* need for information.[2]

$$Y::=\{y\}$$

The second part of the environment, the *data* block, comprises the raw material input to the IR system. We assume this input to be unprocessed textual material in the recorded form convenient to the system. Although certain conventions may be followed in compiling this material for input, no manipulation by trained personnel prior to entry is assumed. No doubt the form of this raw material can influence the system's processing effectiveness (reducing the requirements for automatic content analysis [15, 7] for example), but for our purposes this material is considered as a set of recorded symbols recognizable by the *analysis* block. This set of recognizable recorded symbols is called a document (D), i.e.

$$D::=\{[\alpha_i] \mid [\alpha_i] \varepsilon A\}$$

where each document is composed of a finite number of symbols (characters) $[\alpha_i]$, i.e. single character strings which are members of the finite symbol set A.

We impose few requirements on the *user* and *data* blocks, consequently forcing

[2] All symbols and notation used, except the operators in Table 1, are defined in the Appendix in addition to their definition in the body of the paper.

the IR system to accept an increased responsibility at two points--the *logical processor* and *analysis* blocks. In fact, we see nothing at this time to prevent the IR system's serving a more general purpose; however, the representation of the blocks corresponding to the six IR subsystems (the *logical processor, selector, descriptor file, locator, document file,* and *analysis* block) is oriented toward document retrieval.

<u>A Language Approach to Functional Representation</u>

The symbols used in specifying the functional representation are defined in the Appendix. Wherever possible we have attempted to follow "conventions" employed in programming language definition or the "usual" mathematical notation. Unfortunately, no single set of symbols and no standard terminology are universally accepted; hence, we apologize <u>a priori</u> to the reader for our failure to adhere to his individual preference.

The operators used in the functional representation are defined in Table 1. Basic definitions used in the development of the representation are given in Table 2. Necessary additional notation is introduced within the development of each block.

The *Logical Processor*

We assume that the query is expressed in a restricted natural language; if desirable the degree of restriction could be minor. The primary task of the *logical processor* is to accept the query as input and to produce a reduced query, <u>i.e.</u> the query expressed in the system's vocabulary, as output to the *selector*. Production of the reduced query can be subdivided into the following tasks:

Operator	Description/Definition	Use
$\uparrow$	comparison	compares element on left side of operator to every element of the set on the right side of the operator
()	parentheses	alters usual left-to-right execution of Boolean expression by giving higher priority to operations to be performed within innermost nested parentheses
$\circledcirc$	relational $(=,=,>,\geq,<,\leq)$	$\circledcirc$ denotes any member of the set of relational operators
o	logical $(\wedge,\vee)$	o denotes any member of the set of logical operators. Both <u>and</u> $(\wedge)$ and <u>or</u> $(\vee)$ have the same priority, modified only by the presence of parentheses
oX	$oX::[x_1 o x_2 o \ldots o x_{\nu(X)}]$	the application of the operator o to the set X to form a string (where square brackets denote that the contents of the brackets is considered a string, and $\nu(X)$ denotes the number of elements in the set X)
;	$[\alpha_i];[\alpha_{i+1}]::=[\alpha_i \alpha_{i+1}]$	string concatenation

Table 1. Operators Used in the Functional Representation

Notation	Description
D	a document
d	a descriptor
$\bar{D}::=\{D\}$	set of all documents
$\bar{\Delta}::=\{d\}$	set of all descriptors
R(x)	a. contents of record R corresponding to record identifier x, or b. a set of items associated with identifier x, or c. a mapping which associates with an identifier x a set of items R(x)
Q(D)	set of descriptors associated with (describing) document D
T(d)	set of documents (document identifiers) associated with (described by) descriptor d
v(x)	
$G::=\{g\}$	set of all grammatical constructions (punctuation symbols, non-meaningful strings)
$U::=\{u\}$	set of all index terms
$A::=\{[\alpha_i],i=1,2,\ldots,\nu(A)\}$	the symbol set recognizable by the system

Table 2. Basic Definitions in the Functional Representation

(1) query formation – transformation of single character strings into multiple character strings to form a set Y;

(2) query recognition – identifying the input set Y as a legitimate query and possibly performing a syntactic analysis of the input either independently or in dialogue with the *user* to enable modification according to system requirements;

(3) query reduction – removing all grammatical constructions and nonsubstantive words unrelated to the supposed "information content" of the query;

(4) normalization – expanding the query by dictionary reference in the process of translating the input strings into terms consistent with the system vocabulary; and

(5) pre-search activities – using the formulation of the query resulting from the three previous tasks, to allow *user* feedback in further query modification.

We can represent the function of the *logical processor* by beginning with the query input Y, which initially is viewed as a set of single character strings $[\alpha_i]$ comprised of members of the IR system alphabet, i.e.

$$Y ::= \{[\alpha_i]\} \mid [\alpha_i] \in A\}.$$

The alphabet A is the finite symbol set recognizable by the system $A::=\{[\alpha_i], i=1,2,\ldots, \nu(A)\}$. This set can be partitioned into two subsets

$$C^T \subseteq A \quad \text{and} \quad C^R \subseteq A$$

where C^T is the subset of terminator symbols and C^R the subset of non-terminators. Obviously $C^T \cap C^R = \phi$.

Query Formation

The first function of the *logical processor* is to form multiple character strings, i.e. to convert Y into a set of multiple character strings that are candidates for query terms. This is accomplished by successively concatenating non-terminators until a terminator is encountered.[3] The concatenated multiple character string becomes an element $[\alpha]$ of the transformed query set Y, and the terminators are deleted.

$$L_F: \quad (;[\alpha_i] \; [\alpha_i]\notin C^T \rightarrow [\alpha]_j, [\alpha_i] \; [\alpha_i]\varepsilon C^T \rightarrow \Lambda, \quad \forall [\alpha_i]\varepsilon Y) = Y$$

Thus the query Y is a set whose elements are multiple character strings $[\alpha]_j$. For simplicity we denote this as

$$Y = \{y\}:: = \{[\alpha]_j\}$$

An essential requirement is the left-to-right ordered scan of all character strings, those produced as well as those supplied. Thus all strings are examined in a left-to-right order unless precedence operators, e.g. the parentheses characters, are present to alter this order. We assume the permitted operator set to be composed of the precedence operators, (,) and the logical operators $\vee$ and $\wedge$ (with the negation operator omitted for the sake of simplicity.)

Query Recognition

During the query recognition phase, denoted by the subscript, R,[4] the *logical processor (L)* acts either to reject the query (if it is not syntactically recognizable) or to augment it (in the case of incomplete syntax)

$$L_R: ((Y\cup\{y\}) \mid \phi) \rightarrow Y,$$

where the augmentation {y} may be null and the rejection, indicated by ϕ, obviously prompts some later error message.

Query Reduction

Let

G::={g}, the set of all grammatical constructions (punctuation) and non-substantive terms, each of which is a string, and

U::={u}, the set of all index terms, each of which is a string.

[4]The use of a subscript on a function symbol, e.g. L_R, serves only to identify a particular task of a more comprehensive function in this case the *logical processor*. No relationship is intended between the function and the entities to which it is applied.

A (Boolean) query b is defined recursively, using BNF notation as a string[5]

$$b::=u \mid (b) \mid bob$$

and the set of all possible (Boolean) queries is

$$B::=\{b\}.$$

The *logical processor* begins the query reduction task with a set Y (possibly different from that submitted to the query recognition phase) where

$$Y = \{U' \cup G'\} \text{ and } U' \subseteq U,\ G' \subseteq G.$$

In the query reduction phase the *logical processor* is applied to construct Y′ (the reduced query) a member of the Boolean query set implicit Y, i.e.

$$L_P\colon\ (y_{t_i}, M(E) \uparrow \{U, G, \text{"("}, \text{")"}, \text{"and"}, \text{"or"}\} \rightarrow t_i\ t_i \varepsilon T,\ \forall_y \varepsilon Y) = Y'$$

where T is an ordered set of values returned as a consequence of examining Y. The subscript on t indicates the element correspondence between the compared set and the set of returned values

$$T = \{U, \Lambda, (\ ,\ ), \wedge, \vee\}.$$

The element-set membership function, in general $x_{t,q} \uparrow Z$ is defined in Table A2 of the Appendix. Note that M(E) is an error message to indicate that some element has been unrecognizable in the recognition phase.

Normalization

In the normalization or expansion phase, each term u is used to identify the subset of all descriptors associated with u, i.e. N(u). Associated with each element (e) in the set Y′ (consisting of terms, parentheses and logical operators) is a unique set N(e) defined below.

$$N(e) = \begin{cases} N(u) & \text{if } e = u \\ N(M(E)) & \text{if } e = M(E) \\ e & \text{otherwise} \end{cases}$$

[5] The alternation operator should not be confused with the vertical line enclosed in brackets, i.e. {a|condition} defined in Table A1 in the Appendix.

The values returned for N(e) form a string by the application of the logical disjunction operator to the elements associated with terms u, i.e. to the elements of the set N(u) for all u, (see Table 1 for the definition of oX). These elements are the descriptors, members of the set of all descriptors $\bar{\Delta}$.

$$L_N: \quad (Y'\to\rho["("; \vee N(u); ")"], \forall u\varepsilon Y') = Y''$$

the decomposition function $\rho[oX]$ breaks the string of terms and logical operators into separate elements

$$\rho[oX]::=\rho[x_1 o x_2 o \ldots o x_{\nu(X)}]$$
$$= \{x_1, o, x_2, o, \ldots, o, x_{\nu(X)}\}$$

and contributes to forming the set Y'' by its application.

Members of Y'' are parentheses, the logical operators $(\vee,\wedge)$ and descriptors (d) from the set of all descriptors $(\bar{\Delta}::=\{d\})$. The error message reference is simply a descriptor referencing a single document identifier that is used in the return of selected specifications by the *selector* (its presence obviously indicates an error).

The function of the *logical processor* in presearch activities would involve repetition of these four phases. One can visualize the function of the *logical processor* to be defined in terms of the individual tasks as

$$L::=L_N(L_P(L_R(L_F(\cdot))))$$

The *Selector*

The *selector*, using the processed form of the query (Y'') as input, retrieves from the *descriptor file* the set of all document identifiers associated with each descriptor (dεY''). The indicated logical operations are then performed in the order specified (by the use of parentheses). The result is a set of specifications, i.e. the set of all document identifiers associated with the initial query. In

[3] Recall that the operator preceding a single operand indicates iteration (see Table 1).

Document Selection

In representing the function of the *selector*, we must consider the relationship between this block and the *descriptor file*. We represent the *descriptor file* essentially as a passive block acted upon by the *selector* (and the *analysis* block). Beginning with the first descriptor d, all documents associated with this descriptor (T(d)) are identified and formed into a single element (a string) by application of the logical disjunction operator ($\vee$), i.e. $\vee T(d)$. The string decomposition operator (ρ) breaks the string into elements, where an element is either a document identifier, a parenthesis, or a logical operator. The set formed by $\rho[\vee T(d)]$ replaces d in the set of descriptors associated with the query Y'', and the sequence of operations is repeated beginning with the identification of all document identifiers associated with the next descriptor d

$$(d \rightarrow \rho[\vee T(d)]),\ \forall d \varepsilon Y''$$

Parentheses used to define the precedence of logical operations remain as undisturbed elements of Y''.

Finally the logical operators ($\vee,\wedge$) are replaced by the set union and intersection operators respectively ($\cup,\cap$), and the expression is evaluated to produce the set of document identifiers (loosely, the set of documents) D' associated with (or "relevant to") the original query Y.

Letting Ω be the function which evaluates any valid set expression, the function of the *selector* (Sd) with respect to the *descriptor file* is represented as

$$S_d\colon\ Y'' \rightarrow \Omega(((d \rightarrow \rho[\vee T(d)]),\ \forall d \varepsilon Y''),\ \vee \rightarrow \cup,\ \wedge \rightarrow \cap) = D' \subseteq \bar{D}$$

where $\bar{D}$ is the set of all documents.

While appearing complex, the representation of the *selector* is quite straightforward. Consider the following example

[illegible] $,\wedge,\ d_3,)\}$

with the following references

d_1 refers to D_3, D_5, D_6

d_2 refers to D_4, D_5

d_3 refers to D_4, D_7.

the actions of the *selector* are:

(1) identify the document set T(d) associated with d, e.g. for d_1-$\{D_3, D_5, D_6\}$;

(2) "or" the members of the document set (in a sense this creates a string) and enclose them in parentheses, e.g. for d_1 -- $(D_3 \vee D_5 \vee D_6)$;

(3) apply the decomposition function to T(d) in its string form;

(4) replace d by T(d), e.g. for d_1 -- $(D_3, \vee , D_5, \vee , D_6)$;

(5) this is repeated for all $d \varepsilon Y''$ to give the result, e.g.

$\{(,D_3,\vee, D_5,\vee,D_6,),\vee, (, (,D_4,\vee,D_5,),\wedge, (,D_4,\vee, D_7,),)\}$;

(6) all logical operators are replaced by the union and intersection operators and the result is evaluated by applying

$((D_3 \cup D_5 \cup D_6) \cup ((D_4 \cup D_5) \cap (D_4 \cup D_7)))$

producing the set of documents D', e.g.

$\{D_3, D_4, D_5, D_6\}$.

Post-Search

The second function (sometimes call post-search activity) of the *selector* is to operate on D' in some manner so as to return some aspect of D' to the *user*. We use the notation v(D) to indicate some attribute associated with D is the output.

$$S_u: D' \rightarrow v(D')$$

The nature of $v(D')$ is system dependent since the aspects of D' returned might be quite different. In one case the number of elements in D' might be sufficient while in another the number of documents associated with a particular descriptor might be provided.

Representing both functions of the *selector (S)* requires the applications of the document selection followed by the post-search activity, i.e.

$S::=S_u(S_d(\cdot))$.

The *Descriptor File*

Representation of the *descriptor file* begins with two assumptions:

(1) The hardware capability of the system is similar to that of many existing systems; it includes (besides a large mass memory and multiple tape units) a number of auxiliary storage devices such as disc, drum, and/or data cell.

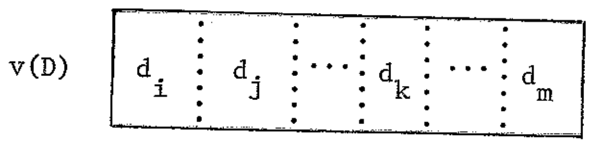
(2) The m n concern in our generalized retrieval system is single query processing (i.e., the query of the individual user), rather than the batch processing of multiple queries.

The *descriptor file* is viewed as passive as we note above. We can characterize it by representing its organization rather than prescribing any active functions performed by it. A similar approach is taken by Hsiao and Harary [8] in representing the search functions (*selector*) as they relate to various file (*descriptor file*) organizations.

We consider the system vocabulary to be changing (probably increasing) and determined by the criteria invoked in the *analysis* block (no static thesaurus is assumed). Firthermore, we assume no weightings are applied to descriptors. The essential task is to represent the process by which the set T(d) is defined for the three principal file organization techniques: serial, inverted, and multi-list.

1. The Serial File

A typical serial file entry is seen as follows:



Associated with each document D is a set R(D) of terms or descriptors, d_i. The serial file may then be characterized by:

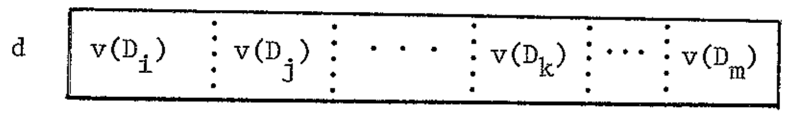
(1) $Q(D)::=R(D)$

(2) $T(d)::=\{d_{v(D)} \uparrow R(D) \mid \forall D\ D'\}$

Thus T(d), the set of all documents associated with descriptor d, is found by the following process. First d is compared to every element of the set R(D). If d is an element of R(D), v(D) (the associated document tag) is returned. The comparison is made for all D contained in the set D′.

2. The Inverted File

A typical inverted file entry is reproduced below.



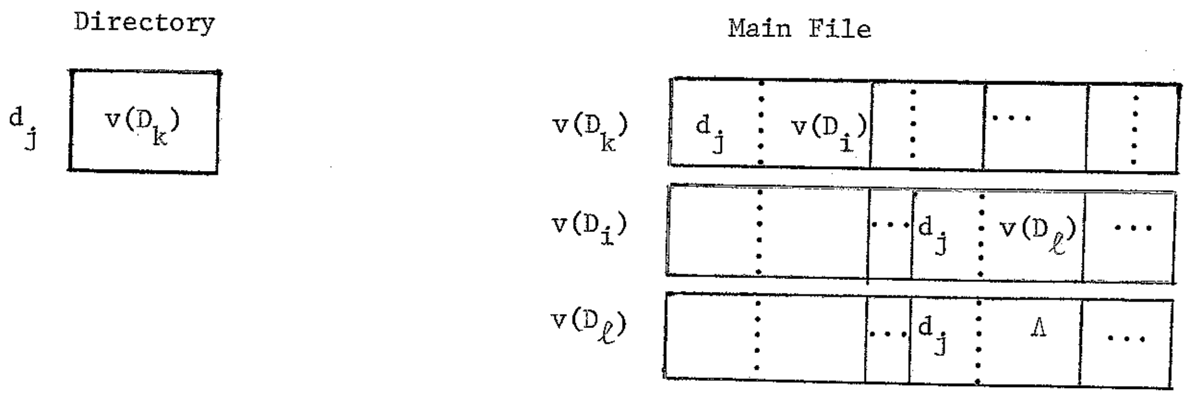
That is, associated with every descriptor d is a set R(d) of documents D_i. Inverted file organization can be represented by:

(1) $T(d)::=R(d)$

(2) $Q(D)::=\{v(D)_d \uparrow R(d) \mid \forall d \varepsilon \bar{\Delta}\}$

3. The Multilist File

Multilist file organization is somewhat more complex than others, since it involves the use of an additional file, frequently called the Directory. Multilist file organization is pictured below.



Thus multilist file organization can be characterized by:

(1) $T(d)::=\{d_{v(d)} \uparrow R(R(d))/v(d)=\Lambda\} \cup R(d)$

(2) $Q(D)::=\{v(D)_d \uparrow T(d) | \forall d \varepsilon \bar{\Lambda}\}$

In each search of the *descriptor file* by the *selector*, the object of the search is the set T(d).

The *Locator*

Just as the *selector* searches the *descriptor file* in order to extract the documents or document tags associated with each descriptor in the normalized query, the *locator (R)* searches the *document file* to extract the record associated with each document in the set (D′) passed to it by the *selector*. This record may consist of the document title, an abstract, or an extract. In any case, the contents of the *document file* entry associated with the document are returned to the *user* under the heading of "located documents." We represent the function of the *locator* as simply:

$$R: \quad \{R(D) | \forall D \varepsilon D'\} \rightarrow \textit{user}$$

where R(D) is the entire data record (entry) associated with document D.

The *Document File*

The *document file* is composed of entries R(D) which are the IR system's representation of the corresponding documents. Formed by the *analysis* block, the system's representation of each document is determined by the criteria applied there. We assume that in every case a unique document identifier v(D) is an entry in the document record R(D).

Vickery [17] states that document representation may be formed in three ways: by simple extraction, be selective extraction, and by the assignment of certain keys (e.g., standard descriptors). The *analysis* block may leave the document (*data*) input virtually intact, operating only to construct the document's representation in the *descriptor file*. Consequently, the entire, unaltered document might serve as its representation in the *document file*.

The *document file*, like the *descriptor file*, is considered a passive block. In this case the *locator* is the active block operating on the *document file*. Similarly, the representation involves defining the file organization which is assumed to be simply by document identifier v(D) or an ordering based perhaps on frequency of use. In either case the *document file* is organized according to some attribute (or combination of attributes) of the record R(D) corresponding to the document D. Thus file organization is represented simply as R(D).

The *Analysis* Block

The *analysis* block constitutes the second entry point for input external to the IR system (the other being the *logical processor*). The function of the *analysis* block is to process the incoming *data* in order to produce two outputs:

(1) some indication of the contents of the incoming document, to be stored in the *descriptor file* along with a pointer to the document in the *document file*, and

(2) a representation of the document itself (i.e., the system's representation of the document), to be stored in the *document file*.

Obtaining the description of the document's contents is commonly called the indexing task.

The importance of the indexing task has been noted by several authors [17, p. 22], [1, p. 317]. Automatic indexing techniques fall into four general categories:

(1) permutation indexing.
(2) citation indexing,
(3) statistical procedures or
(4) syntactic procedures.

While application of the techniques in each category require quite different assumptions and utilize different aspects of the *data*, they all operate on the *data* with the same objective: to construct a set of descriptors that "...somehow indicate (emphasis given originally) the information content of the document..." [1, p. 317]. In executing the indexing function, especially for the third technique, arithmetic and relational operators are required. While we do not use these specifically in the functional representation, one should not overlook the necessity for their presence.

The second major task of the *analysis* block is the construction and storage of a document representation in the *document file*. This representation would include a document identifier, usually all the elements of a bibliographic reference (author, title, publisher, etc.), and might include citations, an abstract, and/or the complete document text.

We should also note the possible use of clustering techniques within the *analysis* block. In information retrieval, the object of clustering algorithms is to generate groups of associated terms (for use in a thesaurus) or to form document clusters facilitating the matching of the analyzed search request with the document identifiers. The intent is to simplify the retrieval process.

We view the task of document representation as requiring some of the functions employed in the indexing task. Usually the indexing task is much more complex while the document representation may be almost perfunctory. Considering the indexing function of the *analysis* block, Vickery [17, pp. 21-22] recognizes three stages in the assignment of document descriptors:

(1) scan of the text to derive those words, phrases, and/or sentences which best represent information content,

(2) a decision as to which of the descriptors are worthy of being recorded in the *descriptor file*, in view of the purpose of the system, and

(3) the transformation of the selected descriptors into a standard "descriptor language", the resulting terms of which serve as the entry or entries in the *descriptor file*.

We describe these three stages by two functions, <u>i.e.</u> the string formation function (A_F) and the descriptor determination function (A_d).

String Formation

Recall that a document D is defined as a set of strings, <u>i.e.</u>

$$D::=\{[\alpha_i] \mid [\alpha_i]\varepsilon A\}$$

In its raw, unprocessed form, the *data* entering the *analysis* block are members of the finite symbol set $A::= [\alpha_i]$, $i=1,2,\ldots,\nu(A)$, which can be considered single character strings. This set can be partitioned into two subsets

$$C^T \subseteq A \text{ and } C^R \subseteq A$$

where C^T is the set of terminator symbols and C^R the ste of non-terminators and

$$C^T \cap C^R = \phi.$$

Thus we represent the *data* (a single unprocessed document) as a set

$$D::=\{[\alpha_i] \mid [\alpha_i]\varepsilon A\}.$$

The scan of D is assumed to be from left to right.

The string formation function A_F is applied to form the set of descriptor candidates Δ. This function (A_F) forms a multiple symbol string $[\alpha]$ by concatenating single symbols until a terminator is encountered. The string produced becomes an element of Δ, and terminators are deleted. This function continues to operate until all symbols $[\alpha_i]\varepsilon D$ are examined.

$$A_F\text{:} \quad (;[\alpha_i] \ni [\alpha_i] \ \varepsilon \ C^T \rightarrow [\alpha]_j, \quad [\alpha_i] \ni [a_i]\varepsilon C^T \rightarrow \Lambda, \quad \forall[\alpha_i]\varepsilon D) = \Delta$$

As a consequence of this operation the set Δ can be defined as a set of strings

$$\Delta::=\{[\alpha]_j \ [\alpha_i] \ \varepsilon \ C^R \ \forall[\alpha_i] \ \varepsilon \ [\alpha]_j\}.$$

Descriptor Determination

The descriptor determination function A_d operates on the set Δ to select those descriptors to be inserted in the *descriptor file.*

$$A_d: \quad ([\alpha]_j \rightarrow d|\Lambda, \forall[\alpha]_j \in \Delta) = Q(D)$$

where Q(D) is the set of all descriptors associated with document F. In this manner no limitations are placed on the size of the descriptor vocabulary $(\bar{\Delta})$ where Δ is the set of all descriptors $\bar{\Delta};;=\{d\}$.

Again, we use an example to illustrate our representation.

Let $C^T = \{\cancel{b},.,,,:,?,!,;,/\}$

$C^R = \{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,Y,Z,),(,'\}$

then $A = C^T \cup C^R$.

A sample from the input string follows:

THIS BOOK DESCRIBES THE USE OF THE DIGITAL COMPUTER IN THE WORLD OF INDUSTRY, COMMERCE, AND BANKING. ...

This the first five symbols are

$[\alpha_1]$ = "T"

$[\alpha_2]$ = "H"

$[\alpha_3]$ = "I"

$[\alpha_4]$ = "S"

$[\alpha_5]$ = "$\cancel{b}$"

The operation of A_F results in the set

$$\Delta = \{[\alpha]_1, [\alpha]_2, \ldots, [\alpha]_{17}\}$$

where $[\alpha]_1$ = "THIS", $[\alpha]_2$ = "BOOK",..., $[\alpha]_{17}$ = "BANKING".

The criteria used in the selection of descriptors (the descriptor determination function A_d) is system dependent since several alternative indexing techniques, as we mention above, could be used. Also, the indexing function could be automated, manual or a combination of the two. Application of A_d to the set Δ is equivalent to applying the function to each member of the set, i.e. for our example

$$A_d\colon\ \Delta \equiv \{A_d\colon\ [\alpha]_1, A_d\colon[\alpha]_2, \ldots, A_d\colon[\alpha]_{17}\}$$

where $A_d\colon\ [\alpha]_j \rightarrow d$ according to the criteria applied, otherwise $A_d\colon\ [\alpha]_j \rightarrow \Lambda$. If a fixed descriptor vocabulary is used, the descriptor qualification $A_d\colon\ [\alpha]_j \twoheadrightarrow d$ is easily determined. In a system where a fixed vocabulary is not used, $\nu([\alpha])_j)$ might be used to determine the result. Thus the function A_d can produce a different set of descriptors Q(D) depending on the criteria invoked. For our example, let us assume that A_d operates on the set of strings Λ associated with document D as follows:

$$A_d\colon\ [\alpha]_1 = A_d\colon\ \text{"THIS"} \rightarrow \Lambda$$
$$A_d\colon\ [\alpha]_1 = A_d\colon\ \text{"BOOK"} \rightarrow \Lambda$$
$$\vdots$$
$$A_d\colon\ [\alpha]_8 = A_d\colon\ \text{"DIGITAL"} \rightarrow d_1$$
$$A_d\colon\ [\alpha]_9 = A_d\colon\ \text{"COMPUTER"} \rightarrow d_2$$
$$A_d\colon\ [\alpha]_{10} = A_d\colon\ \text{"IN"} \rightarrow \Lambda$$

File Maintenance

Two additional functions remain to be accomplished in the *analysis* block, and these relate to the file maintenance requirements. For the *descriptor file* the tasks required differ according to the file organization employed. We denote the maintenance function required for the *descriptor file* by A_M and represent the activities as follows:

1. The Serial File

   $A_M\colon\ \{d \mid d \in Q(D)\} \rightarrow R(D)$

2. The Inverted File

   $A_M\colon\ \{v(D) \cup R(d) \mid \forall\ d \in Q(D)\} \rightarrow R(d)$

3. The Multilist File

$$A_M\text{:}\quad \{d \mid v(d) = \Lambda\ ,\ \forall\ d\ \varepsilon\ Q(D)\} \rightarrow R(D)$$

$$v(D) \rightarrow R(d), \forall\, d \ni d\ \varepsilon\ \{Q(D) \cap \bar{\Lambda}^c\}$$

$$\left.\begin{array}{l} d_{v(d)} \uparrow R(R(d))\ /\ v(d) = \Lambda \\ v(D) \rightarrow v(d) \end{array}\right\} \forall\, d \ni d\ \varepsilon\ \{Q(D) \cap \bar{\Lambda}\}$$

In each case the file maintenance function begins with the set Q(D) produced by A_d. Note that the capability for increasing the descriptor set is shown explicitly for the multilist organization.

The serial and inverted file maintenance functions are simple. In the serial file the descriptor set is assigned to a document record, while the inverted file requires the addition of a document identifier to the set of document identifiers referenced by each descriptor d. For the multilist file, the first operation refers to the formation of a main file record, the second describes the formation of a new directory record, and the third describes the setting of the main file link. In all cases the universal set of descriptors $\bar{\Lambda}$ may be dynamically increasing or remain static.

After determination of the descriptor set Q(D) and its subsequent use in file maintenance functions, the *analysis* block operates on the original text input to construct and/or maintain the *document file*. This function involves only the construction of the document reocrd R(D) and the addition of the document to the set of all documents.

$$A_D\text{:}\quad (D \rightarrow R(D)) \cup \{R(D) \mid D\ \varepsilon\ \bar{D}\} \rightarrow R(D) \mid D\ \varepsilon\ \bar{D}\}$$

In summary, the function of the *analysis* block *(A)* can be represented as

$$A ::= \langle A_M(A_d(A_F(\cdot)))\ ,\ A_D(\cdot)\rangle$$

where the angular brackets enclose an ordered pair.

APPENDIX

Notation	Description
$::=$	definition sign
$=$	equivalence
$\{a \mid \text{condition}\}$	"set of all a, for which condition holds"
$\langle \ \rangle$	an ordered pair
$\{ \ \}$	any set
$[\ ]$	any string
Φ	null set
ε	"is an element of"
$\rightarrow$	replacement (or assignment)
$\circ$	any binary relation $(=,\neq,>,\geq,<,\leq)$
$\odot$	any Boolean relation $(\vee,\wedge)$
Ω	the set evaluation function
$:$	function operator
$\forall D$	$\{D_i,\ i = 1, 2,\ldots,n\}$ where $D_i \varepsilon \bar{D}$
$+,-,*,\div,**$	arithmetic operators (addition, subtraction, multiplication, division and exponentiation respectively)
$\ni$	"such that"
$\wedge,\vee$	conjunction, disjunction
Λ	null field
$\cup$	union of sets
$\vert$	alternative $(A \vert B ::= A \text{ or } B)$
"..."	literal string delimiters
$\cap$	intersection of sets
A^c	the complement of set A $(A^c ::= \{x \mid x \not\in A\})$
$\not b$	blank character

Table A1. Notation Used in the Functional Representation

Function	Notation	Algorithmic Description	Explanation
ement - Set embership (M)	$x_{t,q} \uparrow Z$	M1. [Input] Input values for x, t, q and the set Z.	Compares x with each element of Z until a match is found or the set is exhausted.
		M2. [Compare] $x \uparrow z$ and if the result is <u>true</u> output t, and terminate; otherwise go to M3.	
		M3. [End of set] If the set z is exhausted, terminate with q as output; otherwise, select next member of Z and go to M2.	
ement Selection (E)	$f(x) \ / \ y \odot z$	E1. [Input] Input values for x, y, z and $\odot$.	Provides for selection of a single operator $\odot$. The first element after change of a logical condition is selected.
		E2. [Initial assignment] $r \leftarrow x$.	
		E3. [Apply function f] $r \leftarrow f(r)$	
		E4. [Logical comparison for element selection] If $y \odot z$ is <u>false</u> go to E3; otherwise return r as output and terminate.	
t Selection (S)	$\{f(x) \ / \ y \odot z\}$	S1. [Input] Input values for x, y, z and $\odot$.	Provides for selection of all members of a set until some logical condition is altered.
		S2. [Initial assignment] $r \leftarrow x$.	
		S3. [Output r] $r \leftarrow x$.	
		S4. [Logical comparison for termination] If $y \odot z$ is <u>false</u> then go to S3; otherwise terminate.	

Table A2. Definitions of Functions Used in the Metalanguage Representation