THE CATALOG: A FLEXIBLE DATA STRUCTURE FOR MAGNETIC TAPE

Martin Kay and Theodore Ziehe
The RAND Corporation
Santa Monica, California

The files of data used in linguistic research differ from those found in other research applications in at least three important ways: (1) they are larger, (2) they have more structure, and (3) they have more different kinds of information. These are, of course, all simplifications but not gross ones. It is true that the files that must be maintained by a large insurance company or by the patent office are so large as to pose very special problems, but the uses to which the files are to be put are fairly well understood and their format and organization is not usually subject to drastic and unexpected change. It is also true that the data from a bubble chamber is interesting only if collected in vast quantities, but this is not the only respect in which a bubble chamber is a special kind of tool. A typical linguistic job will bring together a number of files, each very large by the standards of everyday computing: a body of text, a dictionary and a grammar for example. The grammar, if it is anything but a very simple one, will contain a large number of elementary items of information of different kinds, each related to others in a number of different ways. This is what it means to say that the file has a lot of structure. The dictionary may also contain grammatical codes which may consist of characters from one of the languages represented in the dictionary or may be something altogether different. If the dictionary contains alternatives to which probabilities are assigned, then these will presumably be in the form of floating-point numbers. This is what it is like for a file to contain different kinds of information.

The notion of a catalog* was developed principally with the needs of linguistic computing in mind. It is oriented more to the storage of information on a long-term medium, such as magnetic tape, than to its representation in the high-speed store of a computer. The elementary items of information in a catalog are called data. The structure imposed on the data making up a catalog is that of a tree—a hierarchy of sets of information. Let us consider as an example how a bibliography or the acquisitions list of a library—catalogs in the conventional sense—might be organized within this system. Each document or book has an entry in the file containing various items of information about it. One of these can be chosen as the key under which the others are filed. For example, the acquisition number of an item can serve as the key for all information related to the item. Under it there will be sections for author, title, journal if relevant, publisher, and date. In an actual application, there would doubtless be

---

*The catalog system has been developed through the joint efforts of the Centre d'Etudes pour la Traduction Automatique of the Centre National de la Recherche Scientifique at the University of Grenoble, France, and the Linguistics Research Project of The RAND Corporation.
other sections but these will serve for the example. The author section must be capable of handling cases of multiple authorship without confusion and we may assume that there is provision for giving the institution to which each author belongs. The journal section will also have subsections for volume and issue number.

A pair of entries might appear as in the diagram at Fig. 1. The acquisition numbers of these two documents are 1746 and 1747 respectively. The first has one author belonging to a single institution; the second has two, each belonging to a pair of institutions. Furthermore, the two authors of the second paper belong to one institution in common, but since a catalog must, by definition, have the form of a tree and not of an arbitrary graph, the institution must be mentioned separately for each author. The first document is a journal article and there are therefore two nodes below the one representing the name of the journal; the first gives the volume and the second the number. The second entry is for a book and therefore these two nodes do not appear.

In this example, the kind of information at each node can be unambiguously determined from the structure of the tree. The nodes on level 2 all represent acquisition numbers. The last three nodes under a given acquisition number represent title, publication information, and date respectively. Any nodes preceding these represent authors. Any nodes below those for authors are for institutions. A journal article is distinguishable by the volume and number nodes which are absent in the case of a book. However, it is not difficult to imagine cases where rules of this kind would not work. Suppose, for example, that the date of each edition of a book were put in, or that when the date was unknown, we wished simply to omit it. To cope with these situations, it would be necessary either to redesign the catalog structure or to label each datum explicitly to show the kind of information it contains. The structural redesign might be as follows. The names of authors are dropped to level 4 and their institutions to level 5. A new kind of datum is introduced on level 3 dominating the author names. This node has no information of its own; it serves only to show where the authors are to be found. A similar node could be inserted above the date, except that in this case the node would have information of its own to provide for the case where the date is missing.

The catalog system in fact requires that each datum be tagged with the name of the data class of the information it contains. This is useful for other
reasons than the one we have suggested. Each data
class is, for example, associated with a particular
set of encoding conventions. Some contain textual
material, some floating point numbers, some inte-
gers and so on. So, by looking at the class of a da-
tum, we can tell not only what its status is in the
catalog as a whole but also how to decode it.

It is convenient to be able to describe the overall
structure of a catalog, to the computer as well as to
other people, in terms of data classes and the rela-
tions among them. This we do by means of a map,
which, like the catalog itself, is a tree, but in which
the data are replaced by the names of the data
classes. The map of our hypothetical acquisitions
list is shown in Fig. 2. The name of each data class
appears exactly once in the map, a fact which is
bound up with an important restriction on the way
catalog structures may be designed. The members of
a given data class always appear on the same level
of the catalog—the level on which the name of the
class appears in the map—and they always come
immediately below members of the same other class
—the one whose name appears above theirs in the
map. If two classes are shown directly beneath the
same other class in the map, then their members
must appear in that order in the catalog itself.
Thus, in the example, a title may never come to the
left of the corresponding author. Cases can arise
where these restrictions may seem unduly severe,
but, as we shall shortly see, powerful means are
available for dealing with them.

A variant has been proposed for the catalog de-
dsign we are using as an example in which new
nodes would be introduced to represent sets of au-
thors and sets of dates. Since we have data-class
names, we do not need to adopt this variant. How-
ever, classes of this kind, whose members never
contain substantive information, are frequently use-
ful. In other cases, no substantive information is
available for a particular datum, but only for the
nodes it dominates. In these cases, we speak of null
data. The catalog system has been implemented in
such a way that null data occupy no space whatever
on the tape. They can therefore be used freely to
lend perspicuity to the structure of a catalog and
without regard to economy. In order to see how this
is achieved, we must consider the format used for
writing catalogs on tape.

A catalog is reduced to linear order in the most
obvious way. The datum at each node is written on
tape before the data at the nodes beneath, and all
these before nodes to its right. Thus the node at the
root of the tree goes first, followed by those on its
leftmost branch. The first node on a branch is the
first to be written, followed by the nodes on its left-
most branch. When a branch is finished, the one
immediately to the right is taken next. This is the
order arrived at by regarding the nodes as operators
whose arguments are the nodes immediately be-
neath them, and writing the expression out in Pol-


\[
\text{Figure 2. Map of an acquisitions list.}
\]

A set of programs is being written for moving
catalog data between high-speed core and tape. A
blocking scheme is used for the data on tape; the
block size is set by the user. Each datum is treated
as a single logical record whose size is not restricted to that of the physical block. Each logical record is preceded and followed by a link word which gives the lengths of the records on either side of it. The length of the preceding as well as the following record is given in order to make it as easy as possible to backspace. Link words also contain certain other information of importance to the housekeeping of the reading and writing programs, but nothing of the essential structure of the catalog. This being the case, it is possible for a program using the catalog input/output system to make use only of its blocking facilities for certain purposes, calling on the whole system only when required. This plan of building the system as a sequence of layers, each using the facilities provided by the one beneath, has been followed wherever possible in the design. The first two words of each block are an JOBS control word and a FORTRAN control word. These are used nowhere in the current system but are included for reasons of compatibility.

The first word of a logical record which contains a catalog datum is a datum control word. This gives the data class of the datum and its preceding implicit level number (PIL). It is the PIL that enables the system to identify the place of null data which, as we have said, are not explicitly represented by a logical record. The PIL is defined as follows:

1. For all data on level 1, it is 0.
2. The PIL of the first datum dominated by a null datum is the PIL of that dominating null datum.
3. The PIL of every other datum is the level number of the datum that dominates it, i.e. one less than its own level number.

Informally, we can say that the PIL gives the highest point in the tree encountered on the path from the previous non-null datum to the current one. Consider two adjacent data on level $i$ of some catalog and suppose that the PIL of the second is $j$, where $j \leq i = 1, i-j-1$ null data are to be assumed between these two levels $j+1, \ldots, i-1$. Given the class of the current non-null datum, the classes of these null data are uniquely determinable from the map.

It is desirable to be able to write general programs for performing standard operations on catalogs without requiring that the user supply complete information on the structure of the catalog to be treated. For this reason, the map of a catalog is written on tape in the logical record immediately preceding the first datum. The map is represented as a simple list of data-class names paired with level numbers and taken in the order we have described for the catalog itself. Thus, a class whose level is given as $i$ is dominated by the class with level $i-1$ most recently preceding it in the list. With the name of each data class is also given a code showing what encoding conventions are used for the data of that class.

We have noted that the restrictions imposed on the design of catalogs could become onerous in some situations if means were not introduced for overcoming them. We have been considering how a library acquisitions list might be represented as a catalog. But, of all the lists produced by a library, surely this is the least interesting. Suppose instead that we were to undertake to accommodate the subject catalog. Most subject classifications have the structure of a tree to begin with, so that the job should be easy. One possible strategy would be to examine this tree to determine the length of its longest branch, that is, how many categories dominate the most deeply nested one. We may then construct a map with this number of levels plus 5, which is the number used for the acquisitions list. This will make it possible to put a complete entry of the kind considered in the simpler example beneath the node for the most deeply nested category in the classification scheme. In general, the node for a subject heading will have two kinds of nodes directly beneath it, one kind for more particular categories under that heading and one for documents which cover the whole field named by the current heading. The structure already set up for the acquisitions list is repeated once for each document node in the map, but with different data-class names.

This scheme will indeed work, but it is clearly unsatisfactory in a number of ways. For one thing, subject headings will be in different data classes according to their level in the classification as a whole. For another, the map is unduly large and monotonous and liable to change when some minor part of the classification changes. An alternative strategy rests heavily on the claim made for catalogs that any kind of data whatsoever can be accommodated in a datum. If this is so, then an entire catalog can be included as a single datum within another one. From here, it is a short step to the notion of catalogs with recursive structures. Consider the ex-
ample in Fig. 3. The main catalog has two data classes of which the lower one has data that are other catalogs. To emphasize this, we have shown this node with a square rather than a circle. This simple two-class map is written at the beginning of the tape. When a subheading datum is encountered, the first thing it is found to contain is the map of a subsidiary catalog. In order that the data of this catalog should be correctly processed, the tape format must make special provision for them. In fact, subsidiary catalogs are represented not as single logical records, but as sequences of logical records bounded by special markers. However, the user of the system need not concern himself with these details. As far as he is concerned, the included and the including catalogs can be treated in exactly the same manner.

![Diagram of a library subject catalog]

The subsidiary catalog in Fig. 3 is similar to the main catalog for the acquisitions list except for the addition of a single new class to accommodate a further level of subheadings. This again is a class of catalogs and their structure is exactly the same as that of the subsidiary catalog of which they are members. Fig. 4 shows an excerpt from a catalog built on this plan.

Any scheme devised to fill the role for which catalogs were devised must be measured against three main requirements.

1. It must be easy to update.
2. It must provide for retrieval of information in response to a wide variety of requests.
3. It must allow files to be organized on new principles as research proceeds.

Now, the catalog system is not intended as a full-fledged information-retrieval system, but it does contain something of what any such system would have to provide. In particular, it provides powerful and flexible facilities for addressing data and sets of data. Furthermore, this addressing capability is precisely what is required for an updating algorithm where the principal work consists in identifying the items of information to be treated.

There is no obvious limit to the refinements that could be introduced into a catalog addressing scheme, and our ideas on the subject can be guaranteed to far outpace our ability to implement them. Here, we must content ourselves with a survey of some of the simpler notions.

It will be convenient to distinguish between the location of a datum and its address. Each datum in a catalog has a unique location which may be thought of as its serial number on the tape, or as anything else which preserves its uniqueness. But a datum may have an indefinite number of addresses, only some of which refer to it uniquely. The location of a datum will not normally be known to the user of a large catalog, but this is of no consequence provided that there is a clear method of
specifying unique addresses. The notion of location is useful only for understanding how the addressing scheme works.

Some parameters which are useful in identifying a datum are its data class, its class ordinal, its level, its level ordinal and its value. Data class and level have already been explained. The ith datum of a given class dominated by a single datum on the next higher level has class ordinal i. The ith datum of any class dominated by a single datum on the next higher level has level ordinal i. We can now describe the form of an elementary address. This is a triple consisting either of a level, a level ordinal, and a value or a data class, a class ordinal, and a value. In either case, any member of the triple may be left unspecified. The following are some examples:

\[(1, ,a)\] — The data on level 1 with value a.
\[( ,,a)\] — All data with value a.
\[(A)\] — All data of data class A.

Informally, we are using the convention that, if the first member of a triple is an integer, it is a level number; if it contains alphabetic characters, it is a data-class name.

An elementary address, like any other address, can be regarded as a function whose value, if defined, is a location or list of locations. Two other useful functions, descendant and ancestor, have location lists both as arguments and values. These are:

\[\text{Des } [L]\] — All data dominated by the datum or data at L.
\[\text{Anc } [L]\] — All data dominating the datum or data at L.

A concatenation of addresses is itself an address whose value is the intersection of the location lists referred to by each of them. This machinery is already sufficient to call for data in a number of interesting ways. The following examples refer to the

Figure 4. A portion of a library subject catalog.
catalog, part of which is shown in Fig. 1 and whose map is given in Fig. 2.

Des \[(2, 1747)\] (3, 2)

The datum or data on level 3 which are second sons of data on level 2 with value 1747—the value in this case is H. A. Simon.

Des \[(2, 1747)\] (Author)

Here, “1747” is a level ordinal rather than a value, but we may assume that there is just one entry for each acquisition number. This therefore refers to the authors of the 1747th document in the file—E. A. Feigenbaum and H. A. Simon.

Des \[(2, 1747)\] (Author, 2)

This is the same as the previous example except that it selects the second author—H. A. Simon.

Anc \[(lnstitution", R.C.A. Laboratories)\]

(Author.)

All authors from R.C.A. Laboratories. As far as we know from Fig. 1, this means only J. Nievergelt.

Anc \[(Journal", JACM)\] (2)

Everyone who has published in JACM.

Anc \[(Date", 1964)\]

Acquisition numbers of everything published in 1964.

It is clear that other functions could be added to these two without difficulty.

Addressing catalogs, some of whose data contain catalogs poses special problems requiring some new machinery for their solution. At least three new functions, member, recursion, and catalog, are required:

- **Mem**\[L, M\]—where L is an address or location list and M is an address. The value is defined only if some of the data referred to by L contain catalogs. To these internal catalogs, the address M is applied to yield the final value of the function.
- **Rec**\[L, M\]—where L is an address or location list and M is an address. This permits data to be located in general recursive catalogs. Its value is a list of locations arrived at by (1) applying M to the top level catalog and (2) applying the whole function to the catalogs identified by L.
- **Cat**\[L\]—where L is an address. The address L is applied within all catalogs contained in data of the current catalog. The location in the current catalog of any catalog in which a datum is found meeting that address becomes a member of the list which is the value of the function.

It will be easiest to see how these work by reference to a specially constructed example. Fig. 5 shows three maps which are used in the structure of some recursive catalog. Map 1 gives the structure of the main catalog. In this, data of classes B and C contain catalogs with maps 1 and 2 respectively. The catalogs with map 2 have one class containing catalogs, namely Q; class U, of catalogs with map 3, also contains catalogs.

![Figure 5. A recursive catalog structure.](https://www.computerhistory.org)
would, and probably will, fill several more papers.

formation can be broken down into a series of
trees rather than files of independent records, the
algorithms must clearly be unconventional in other
ways as well. However, the differences are less than

The variety of transformations which could be applied to a sufficiently complicated cata-
log is almost endless and the search for algorithms
capable of carrying out any one, provided only that
it is specified in a terse but perspicuous notation,
leads to theoretical and practical problems which
would, and probably will, fill several more papers.

There is good reason to suppose that any trans-
formation can be broken down into a series of
elementary transformations of which there will be
only three or four types. One of these will have the
function of decomposing a catalog into simple cata-
logs. A simple catalog is one which has exactly
one data class on each level. If two simple catalogs
have the same classes on the first $k$ levels, then they
can be merged to form a new catalog with one data
class on each of the first $k$ levels, and two on the
levels below. In the same way, it is clearly possible
to merge a pair of catalogs of whatever structures
provided only that they have a common class on
level one. If they have common classes on levels 2,
3, etc., then the blend will be the more complete.
Of course, before the merge can take place, it may
be necessary to perform a sort. Now, any catalog in
which there are $n$ data classes which have no subor-
dinate classes can be regarded as a compound of $n$
simple catalogs. Many transformations can be ef-
acted by decomposing the given catalog either par-
tially or completely, changing the relative levels of
the data in some of the simple catalogs, and merg-
ing them together again in the same or a different
order.

Sorting and merging are common components of
transformations as well as other catalog procedures
and the overall system must clearly give them an
important place. Since it is part of the essence of
catalogs that they contain a rich variety of data
types encoded according to diverse conventions,
much flexibility is required than most sorting proce-
dures provide. In particular, the encoding of the in-
formation in a given data class will, in general, not
be such that an algebraic sort on the resulting bi-
ary number will give the required results. Further-
more, there may be some classes in any catalog in
which the order of the data cannot be algorithmi-
cally determined; their order may be essentially ar-
binary, each under its own dominating element on
the level above, or the requirement for the sort may
be that the order of the data in the class be pre-
served from the original input. To provide for con-
tingencies of these kinds, the catalog merge and sort
routines allow the user to supply a comparison rou-
tine for each data class in any catalog to be treated.
This routine takes a pair of data of the specified
class and declares which of them should precede the
other in the output. The routine also knows what
the input order was and may use this in arriving at
a result.

Since the objects to be merged and sorted are

\[
\begin{align*}
\text{Mem} \left[(B),(D;x)\right] \\
\text{Data of class D and value x in catalogs in} \\
\text{data of class B in the main catalog.} \\
\text{Rec} \left[(B),(D;x)\right] \\
\text{Data of class D and value x anywhere in the} \\
\text{recursive system.} \\
\text{Cat} \left[(Q,x)\right] \text{Des} \left[(P,y)\right] \\
\text{Data in the main catalog, necessarily of class} \\
\text{C, containing a catalog in which there is at} \\
\text{least one datum of class P with value y domi-
\text{nating at least one datum of class Q with} \\
\text{value x.} \\
\text{Mem} \left[(C),\text{Rec}[\text{Mem} \left[(Q),(U),(R)\right]]\right] \\
\text{Data of class R anywhere in the recursive} \\
\text{system.} \\
\text{Des} \left[\text{Cat} \left[(D,y)\right] \left(E,x\right)\right] \\
\text{Members of class E with value x dominated} \\
\text{by a datum (which, in this case, can only} \\
\text{be of class B) which contains a catalog which} \\
\text{has a datum of class D and value y.} \\
\text{Rec} \left[(B),\text{Des}[\text{Cat} \left[(D,y)\right]](E,y)\right] \\
\text{Same as above, but in the whole recursive} \\
\text{system.} \\
\end{align*}
\]
might at first appear. If each node in a catalog were duplicated once for each lowest-level datum it dominated, the catalog would take on the aspect of a great number of chains with a lowest-level datum at the foot of each. Each of these chains could then be treated as a single record to be sorted in a conventional way. Whilst it is never necessary to actually expand a catalog into this cumbersome form, the computer can arrange to retain a datum in memory until all its decendants are passed so that its instantaneous view of the catalog at any moment is as of a chain.

The catalog system could develop in many different ways and it is our intention that it should. For something so pedestrian as a filing system, it is remarkable how it has captivated the stargazer and the theoretician as well as the bookkeeper and the librarian in everyone who has worked on it. And these have been many. However, it is important for the welfare of computational linguistics that catalog systems or something designed to fill the same need should be made available soon. We have therefore resolved to be done with theorizing, for the present at least. What catalogs need is action.