Institute for
Social Science Research

ISSR
Working Papers
in the
Social Sciences

1987-88
VOLUME 3. NUMBER 14

An Approach to CAT1
Instrument Management Design
Utilizing the Properties
of Logical Structures

by

Matthew Futterman

University of California
Los Angeles

AN APPROACH TO CAT1 INSTRUMENT MANAGEMENT DESIGN UTILIZING THE PROPERTIES OF LOGICAL STRUCTURES

Matthew Futterman

ABSTRACT

This paper recognizes the increasing burden being placed on those who design and program instruments for Computer-Assisted Telephone Interviewing (CATI) systems, and proposes certain enhancements to CAT1 instrument management capabilities that should be useful in easing this burden. The suggested approach for incorporating these features is to use a design based on the properties of the logical structure of instruments, many of which are described. An attempt is made to demonstrate that this approach results in a reliable, extendable and comprehensive CAT1 system design.

Within the short span of ten years or so, the perception of Computer Assisted Telephone Interviewing (CATI) systems has undergone a dramatic change from that of an experimental, high technology innovation to that of an indispensable and powerful tool. This has mirrored the growing popularity and acceptance of computers themselves. The term "personal computer" is a part of our everyday language, as increasingly we see a migration of an ever-widening range of applications from the larger mainframe and minicomputer environments to the smaller realm of the microcomputer. Computer Assisted Personal Interviewing (CAPI) systems are now becoming commonplace.

There are clear implications for CAT1 and CAP1 as they gain greater acceptance:

- As computers become more available and easier to use, the applications that run on them are expected to become increasingly more functional and sophisticated.
- As more people accept the idea of using a computer for a particular application, the more it becomes necessary to make it easy to use.
- Highly functional and sophisticated programs that are easy to use are also difficult to write.

Highly complex questionnaires are considered for implementation on CATI/CAPI systems more readily than was the case when CAT1 was an emerging technology. Techniques that are impossible or difficult to implement using paper-and-pencil methods, such as randomizing the ordering of items, dynamically updating portions of text, randomizing respondent selection, to name a few, are now commonly available on CAT1 systems. New capabilities are continually being devised. The attitude seems to be: if it can be done on the computer, it should be done on the computer; if it cannot be done on the computer, find a way to do it on the computer.

Increasingly, a heavy burden is placed on CAT1 instrument designers, who must contend with the often conflicting objectives of ease of use, greater functionality, and program correctness. Program correctness is a particularly important goal of the instrument designer, since an incorrectly written instrument not only can confuse interviewers and respondents, but can cause loss of data and propagation of errors that are sometimes difficult and costly to correct.

How can the task of the instrument designer be made easier? What facilities can the CAT1 system provide to the designer in order to improve his chances of writing a correct program? What instrument management features can the CAT1 system provide? These are questions that CAT1 system designers need to answer.

The importance of program correctness and the necessity of drawing on sound instrument design principles has been described by Nicholls and House (1987). Two general principles pertaining to the design of Computer Assisted Data Collection (CADAC) instruments were set forth:

- Each item of a CADAC instrument should be protected from every general form of interviewer movement that is possible in the CADAC system including those forms not anticipated in that survey.

- The design of CADAC instruments for consistent correctness should begin with analysis of the structural relationships among its items based on their branching paths and modular design.

In order to help the instrument designer comply with the second principle, CAT1 systems should be designed to provide him with as much information on the structural makeup of an instrument as is possible. This would include diagrams of the instrument's structure, the names of key items, section boundaries, and so forth.

Beyond providing the instrument designer with valuable feedback, however, the CAT1 system itself can benefit by using instrument structural information. For example, the system can decide whether or not to honor an interviewer jump request by evaluating the state of the current instrument path and the logical relationship between the item currently on the screen and the item that is the target of the jump request. The instrument designer need not be concerned about coding explicit instrument instructions for the purpose of protecting against random movement if the CAT1 system provides this protection automatically.

There are other ways a CAT1 system can benefit from having a complete knowledge of an instrument's logical structure. Stored responses can automatically be evaluated for applicability, instrument designers can code conditional instructions without having to make "on-path" determinations, cleaning instructions (excluding those for consistency checking) do not have to be written, and "you-are-here" diagrams can be displayed to the interviewer on request.

There is much to be gained, therefore, by basing a CAT1 system's instrument management capabilities on the structural makeup of instruments. Just as the analysis of an instrument's logical structure should be the first step in

instrument design, so should the utilization of instrument structural information be the first step in instrument management.

Hopefully, this paper will be of interest to the CAT1 system designer who desires an integrated and logical approach to the task of designing a wide range of instrument management capabilities. Emphasized here is a design approach; indeed some of the capabilities described are already implemented on existing CAT1 systems.

The following chapter describes CAT1 instrument logical structures and their properties. The concluding chapter applies these properties to instrument management issues.

STRUCTURAL PROPERTIES OF CATI INSTRUMENTS

In order to develop a strategy for designing a CAT1 system that includes the instrument-management capabilities described in the previous section, it is necessary to take a closer look at instruments themselves. We will begin by describing instruments in terms of logical structures and then take a closer look at logical structures and their properties.

Recognizing CAT1 Instruments as Logical Structures

We will use an example to illustrate an instrument with a simple logical structure. Suppose, for example, that an instrument consists of three items: Ql, Q2 and Q3 as shown in Figure 2a. Ql is executed first, followed by Q2 and terminating with Q3 Note that each item contains an explicit instruction that tells CAT1 which item to execute next.

Now look at Figure 2b. Here, the order of the items as shown on paper is Q2,Ql,Q3, yet note that the branching instructions included in each item are the same as in the previous example. Since CAT1 uses the instructions contained in each of the items to determine the next item, the order that CAT1 will present the items is in each case Ql,Q2,Q3. This ordering, determined by the branching instructions contained in each of the items, is referred to as an instrument's LOGICAL order. The order that the items appear on a piece of paper or on a computer disk is called the PHYSICAL order of instrument items. It is important to remember that CAT1 uses only the logical order to determine the order in which items are presented to the interviewer.

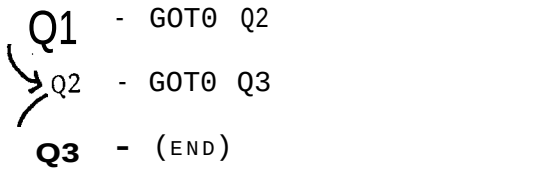


Figure 2a

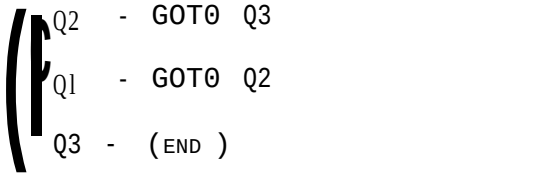


Figure 2b

A LOGICAL STRUCTURE may be defined as the collection of all possible logical orderings within an instrument. In the simple instrument of Figure 2 there is only one logical ordering of the items since each item contains UNCONDITIONAL branching instructions. Few instruments are this simple, however, because most instruments include CONDITIONAL branching instructions as well.

Figure 2c is an example of an instrument with conditional branching. Note that if the answer to Ql is "YES", then Q2 will come next, otherwise Q3 comes next. This instrument therefore has two logical orderings: (1)Ql,Q2 and (2)Ql,Q3. Note also that the logical structure of this instrument is more complex than that of the one in the first example, since it consists of two logical orderings.

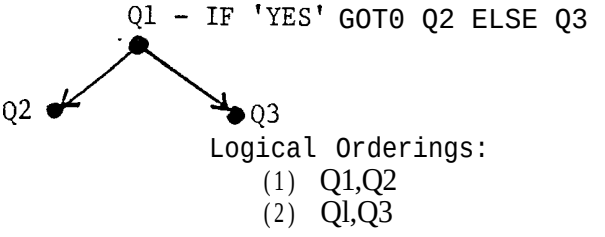


Figure 2c

A further example will illustrate yet another possibility, an instrument that has both conditional and unconditional branching. In this instrument, shown in Figure 2d, if the answer to Ql is "YES", then Q2 comes next, followed by Q3, otherwise Q2 is skipped and only Q3 follows. These two logical orderings, summarized by notation, are: (1)Q1,Q2,Q3 and (2)Q1,Q3

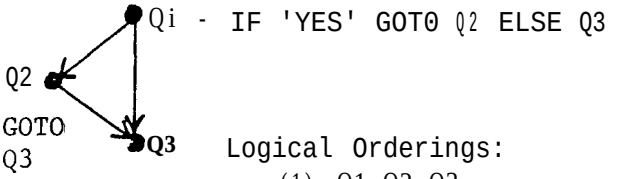


Figure 2d

We have seen that even a very small instrument - one with only three items - has a degree of complexity, given that there is more than one possible logical ordering of the items. Actual instruments have hundreds, even thousands, of items and a similar number of logical orderings. It is easy to see how quickly both man and machine can become "lost" in such complex logical structures unless the nature of the logical structure is well understood. For this reason, the next section takes a closer look at logical structures.

Components of Logical Structures

In the previous section the concept of an instrument as a logical structure was introduced. In this section we will take a closer look at logical structures and identify some of their properties.

Points and Paths

We shall define a logical structure as a collection of all the possible logical orderings of items in an instrument. It is convenient to think of each item as a POINT. We can then represent a logical relationship between two points by connecting them with a line, as shown in the diagrams below. Each connecting line is called a PATH, and represents the possibility of a logical ordering among the points connected by it. Points at the top of a path precede points further down, unless an arrow is drawn to indicate otherwise.

In Figure Z-la, for example, since point A is followed by point B (that is, the logical ordering is A,B) there is a path between points A and B in the direction of A to B (A->B). As another example, Figure 2-lb shows the logical ordering of three points A, B and C as A,B,C. Paths exist between points A and B (A->B), B and C (B->C) and A and C (A-X).

Fi gure 2-1a

Figure 2-lb

Sometimes there is no path between two points. For example, if a logical structure contains two logical orderings, (l)C,D,G and (2)C,E,F,G, then no path exists between points D and E, nor between points D and F, as shown in Figure 2-1c.

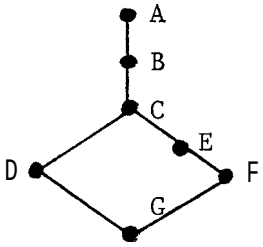


Figure 2-1c

The Main Section, Nodes, and the Active Path

One can learn much about a logical structure by charting all the possible paths it contains. One way to do this is to think of a logical structure as a MAIN SECTION. A main section always has a unique starting point called an INITIAL NODE and a unique ending point called a TERMINAL NODE. All paths in the main section start at the initial node and stop at the terminal node.

Figure 2-2 shows a main section with an initial node labelled BEGIN and a terminal node labelled END. Other points are labelled as well. Note that, though there are many paths, all of them start at BEGIN and terminate at END.

A path represents only the possibility of a logical ordering between two points and not necessarily the existence of such an ordering. In other words, even though a particular path may exist, it may not be the one actually taken. For example, at point C of Figure 2-2, there are two logical orderings, (l)C,D and (2)C,E and, therefore, a choice of two paths to take, either C,D or C,E. The path that is selected is referred to as the ACTIVE PATH. Figure 2-3 shows the same main section as the one of Figure 2-2, but with a bold line indicating a hypothetical active path. In this example, path C,E is part of the active path.

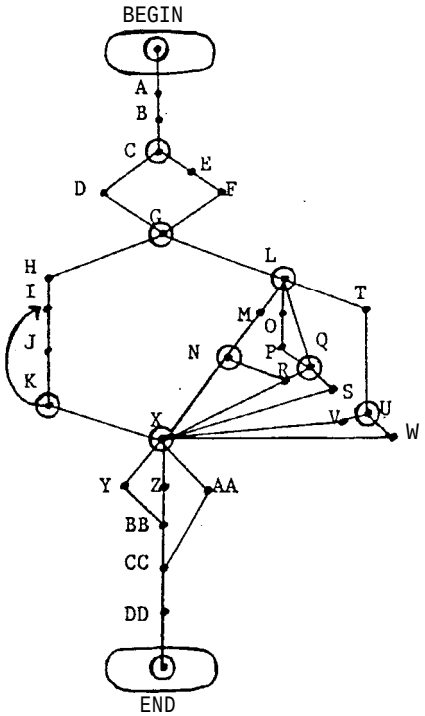


Figure 2-2

There can be only one active path between two points. Referring again to Figure 2-2, we see that there are two paths between points C and G; (1)C,D,G and (2)C,E,F,G. In any instance of traveling from point C to point G, it is not possible to take both paths simultaneously. One and only one path must be selected. It is possible to alter the active path, however. For instance, after having chosen (2)C,E,F,G as the active path, as in Figure 2-3, we can then go back to point C at some later time and choose path (1)C,D,G.

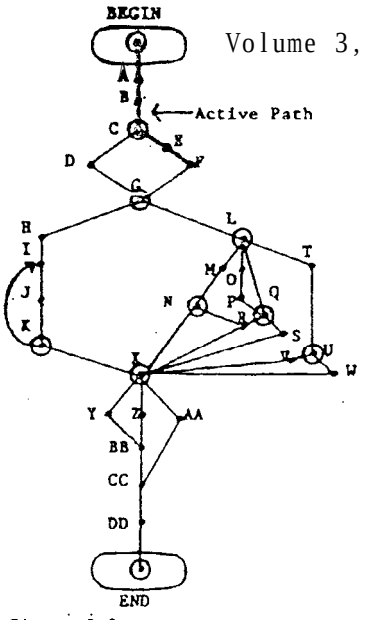


Figure 2-3

Any path between the initial node and the terminal node is called a COMPLETE PATH. A main section always has at least one complete path. Since, by definition, there can only be one active path between two points, it follows that there can only be one ACTIVE COMPLETE PATH between the initial node and the terminal node.

For example, the main section of Figure 2-4 has several complete paths, however only the path BEGIN,A, B,C,E,F,G,L,O,P,Q,R,X,Y,BB,CC,DD,END, is the active complete path.

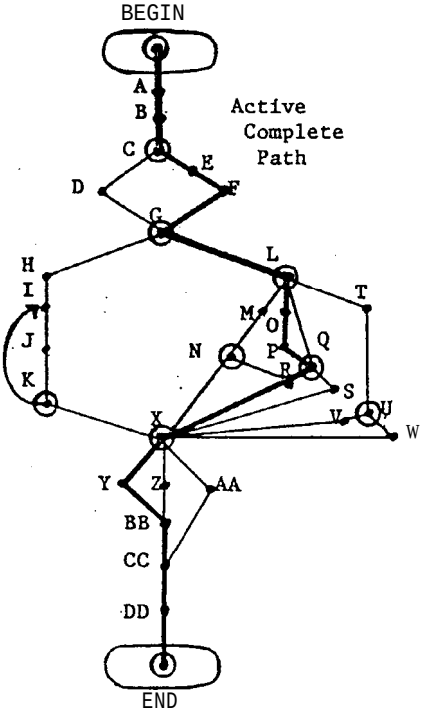


Figure i-4

Notice that the complete paths of the main section shown in Figure 2-4 have several points in common, specifically points BEGIN, A, B, C, G, X, CC, DD and END. This is useful information, since it allows us to identify, in advance, those points that we know must be on each and every complete path. In general, any point that is on all complete paths is called a MANDATORY POINT; any path between two adjacent mandatory points is called a MANDATORY PATH. Thus, in our example, BEGIN,A,B,C is a mandatory path. Path C,E,F,G is not a mandatory path since mandatory points C and G are not adjacent to each other.

Knowing whether or not a point is on the active path is useful in determining the APPLICABILITY of a stored response or value. For example, suppose point E in Figure 2-4 represents an item that stores a response in location @E. If we are certain that there is indeed a path from BEGIN to E (in other words, that E is on the active path), then we can also be certain that the value stored in

@E is valid (APPLICABLE). On the other hand, since point D of Figure 2-4 is not on the active path, it follows that there is no way to get from BEGIN to D. Thus, even if there is a response stored in @D (because, for example, D was previously on the active path), it should be ignored, since it is INAPPLICABLE. We will summarize this property of points on the active path as follows:

1) values stored by points on the active path are applicable

2) values stored by points not on the active path are inapplicable

Certain points in a main section are instrumental in determining the active path. Point C of Figure 2-4, for instance, has a special property since it determines which of two paths is the active path. Similarly, point G selects one of two paths, point X selects one of three paths, and so on. In general, any point that controls which path is to be selected is called a NODE. The circled points of Figures 2-2 through 2-4 are all nodes. (An initial node is a node because it always determines the initial part of the active path; a terminal node is a node because it always determines the path out of the main section.)

A node, unless it is a terminal node or perhaps an initial node, uses conditional branching instructions to select one and only one of two or more paths. Assume that node C, for example, is an instrument item on the active path that contains the following conditional branching instructions:

```
IF THE ANSWER TO C IS "YES"
     THEN GOTO D
     ELSE GOTO E
```

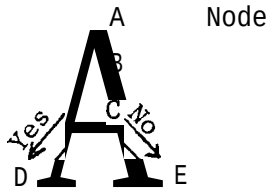


Figure 2-5

If the answer stored at C is "YES", then path C,D is selected, else path C,E is selected. One and only one of the two paths must be chosen as a result of any instance of arriving at node C, as shown in Figure 2-5. This property can be stated more generally as follows:

- If a node is on the active path, that node unambiguously determines the next portion of the active path.

It is important to emphasize that a node, in order to properly determine the next portion of the active path, must itself be on the active path. This is because the answers or other values that are used to evaluate the conditional branching instructions must be applicable.

Determining the active path and knowing when and how the active path has been altered are powerful tools that can be used effectively for instrument management, as we will show later on.

Branches, Logical Units and Junctions

Up to this point we have discussed points, paths and nodes, and how they may be combined to form the active path. Knowing which points are on the active path can, in turn, tell us many important things about a main section (instrument). We will now take a closer look at these basic building blocks and explore how they combine to form branches, logical units and junctions, and how these constructs can be used to tell us still more about how to determine which points are on the active path.

The portion of a path between two adjacent nodes is called a BRANCH. For example, in Figure 2-6 there are two branches between adjacent nodes C and G: C,D,G and C,E,F,G. (Branches C,D,G, and C,E,F,G can also be represented notationally as C(l),D,G and C(2),E,F,G, respectively.) Each point on the branch is called a MEMBER of that branch. Points C, D, and G are, for example, each members of branch C(l),D,G.

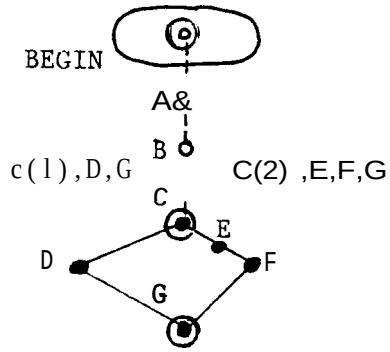


Figure 2-6

Some other examples from Figure 2-2 of branches and their members are G(l),H,I,J,K, L(3),Q, BEGIN,A,B,C, and X(2),Z,BB,CC,DD,END.

A branch that is on the active path is called an ACTIVE BRANCH. A node that is on the active path will always have one and only one active branch. For example, node C of Figure 2-7a is on the active path and its active branch is C(2),E,F,G.

The two nodes that define a branch have special names. The first (top) node is called the CONTROL NODE and the second (bottom) node is called the LINK NODE. Thus, in Figure 2-7a, the branch C(l),D,G is bounded by control node C at the top and link node G on the bottom. Note that node G is also the control node for the branch G(2),L as shown in Figure 2-7b.

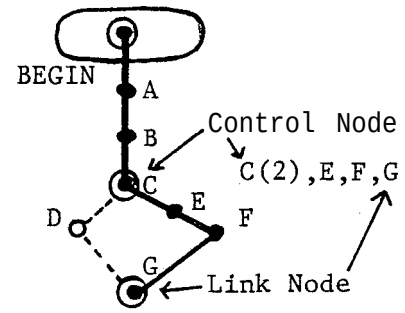


Figure 2-7a

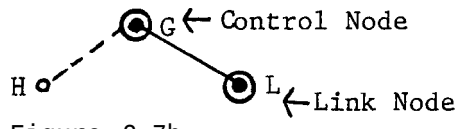


Figure 2-7b

Except for the initial node, which is always a control node, and the terminal node, which is always a link node, all nodes are both control and link nodes. Whether we choose to refer to a particular node as a control node or as a link node depends on how we view that node in relation to other points in the main section. Generally, if we view a node in terms of how we can get to it from other points, we think of it as a link node; if we view a node in terms of how we can proceed from it in order to get to other points, we think of it as a control node.

Note that some branches consist only of a control node and a link node, such as branch G(2),L of Figure 2-7b, while other branches, such as C(2),E,F,G have one or more points between the control and link nodes.

We shall formally define the general structure of a branch as:

c(#),{p1},...pn,)1

where c is a control node
is a control node branch number
p1 through pn are points that may or may not be present
1 is a link node

In general, the set of branches that are bounded by a control node, cn, are collectively referred to as the FAMILY of branches associated with control node cn.

For example, in Figure 2-8, control node L consists of a family of four branches, specifically L(1),M,N; L(2),O,P,Q; L(3),Q; and L(4),T,U. A control node, along with all the points in its family of branches, is called a LOGICAL UNIT. Logical unit L, for example, consists of points, or members L,M,N,O,P,Q,T AND U.

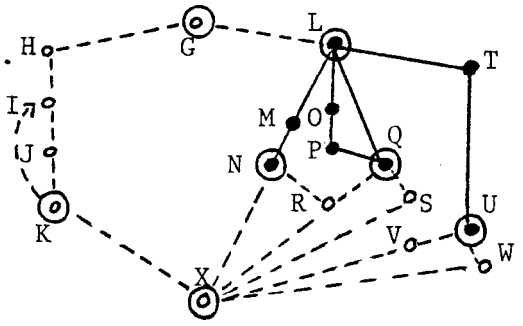


Figure 2-8

It is possible to assess the complexity of a main structure by examining the complexity of its branches and logical units. A SIMPLE BRANCH, for example, is a branch whose members, excluding the control and link nodes, belong exclusively to that branch. For example, in Figure 2-9a, points E and F are members of branch. C(2),E,F,G exclusively, and C(2),E,F,G is therefore a simple branch.

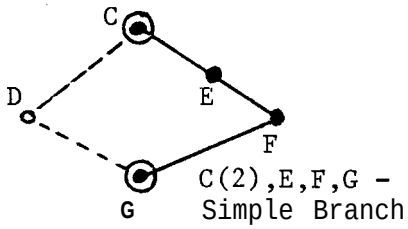


Figure 2-9a

A COMPLEX BRANCH, on the otherhand, is a branch that shares one or more of its non-node members with one or more branches. Branch N(2),R,X of Figure 2-9b, for example, shares non-node member R with branch Q(l),R,X, thus both branches are complex.

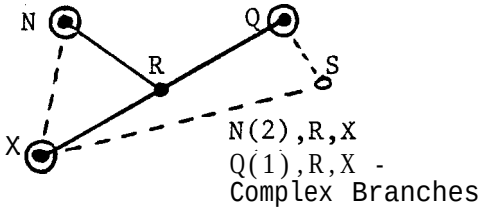


Figure 2-9b

Similarly, logical units can be either simple or complex. Logical unit L of Figure 2-8 is a SIMPLE LOGICAL UNIT because all of its non-node members belong to it exclusively. Logical units N and Q are COMPLEX LOGICAL UNITS because they share non-node member R.

In complex branches and logical units, each set of shared points is called an INTERSECTION and the initial point of each intersection is called a JUNCTION. For example, in Figure 2-9b, the intersection of branches N(2),R,X and Q(l),R,X is R,X and the junction is the first point of intersection, R. As another example, in Figure 2-9c, the intersection of branches X(l),Y,BB,CC,DD,END and X(2),Z,BB,CC,DD,END is BB,CC,DD,END and the junction is BB. Note that all points following the junctions are common to the intersecting branches.

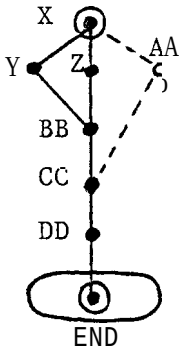


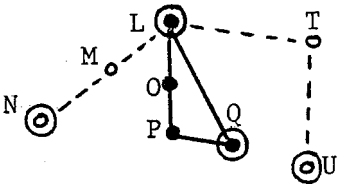
Figure 2-9c

This is an important property of intersecting branches and can be stated more generally as follows:

> when n branches intersect at a junction, j, j, as well as all the points following j on the branch, are common to each of the n branches.

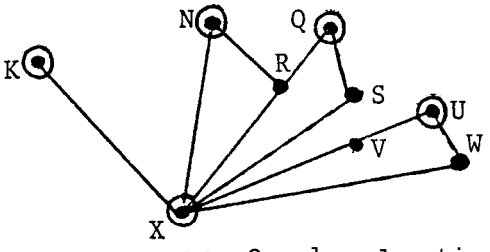
Let us take a closer look at junctions R and BB. Though they are both junctions, BB is a member only of logical unit X, whereas R, as mentioned earlier, is a member of two logical units, N and Q. A junction that is a member of one and only one logical unit, such as BB, is called a SIMPLE JUNCTION; a junction that is a member of more than one logical unit, such as R, is called a COMPLEX JUNCTION.

A point that is a node as well as a junction is called a JUNCTION NODE, and exhibits the properties of both. The main section of Figure 2-2 contains three junction nodes, specifically G,Q and X. Junction nodes are always link nodes for two or more branches. Junction node Q, for example, serves as the link node for branch L(2),O,P,Q, as well as for branch L(3),Q. Another example is junction node X, which serves as a link node for branches K(2),X; N(1),X; N(2),R,X; Q(1),R,X; Q(2),S,X; U(l),V,X and U(2),W,X.



Q- Simple Junction Node

Figure 2-10a



× Complex Junction Node

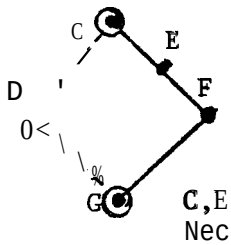
Figure 2-10b

As with junctions, junction nodes can be either simple or complex. Junction node Q, for instance, is a SIMPLE JUNCTION NODE because it is a member of only one logical unit, in this case logical unit L, as in Figure 2-10a. Junction node X, on the other hand, is a COMPLEX JUNCTION NODE since it is a member of four logical units: K,N,Q and U, as shown in Figure 2-10b

A closer look at the properties of nodes, branches, logical units and junctions will shed some light on the varying degrees of complexity that they contribute to the overall structure of a main section.

A simple branch, for example, has the property that any member of the branch must necessarily precede - that is, it must be the NECESSARY PRIOR of - each succeeding non-node member. Consider simple branch C(2),E,F,G of Figure 2-11a, for example. In order to get to point F it is first necessary to pass through points C and E. Thus point F has at least two necessary priors, points E and C. Point E has at least one necessary prior, point C.

In practice, this means that if we know, for example, that C(2),E,F,G is an active branch, as it is in Figure 2-lla, then we know that if we are at point F, we must have passed through points C and E, since C and E are necessary priors of F. It is important to note, however, that none of the non-node members of the branch can be a necessary prior of link node G, since G is a junction and can be preceded by either point D or point F, depending on whether branch C(1) or branch C(2) is the active branch.



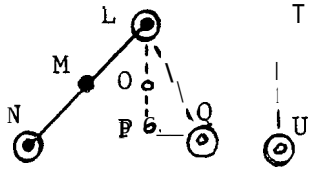
C,E - Necessary Priors of Point F

Figure 2-lla

On the other hand, non-node member M of simple branch L(l),M,N of Figure 2-llb is a necessary prior of link node N because N is not a junction. This leads us to the following two conclusions:

- The members of a simple branch are necessary priors of each of the succeeding non-node members.

- The non-node members of a simple branch are necessary priors of the link node if and only if the link node is not a junction.

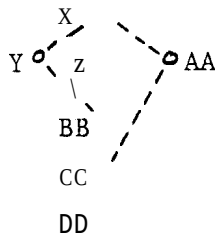


M - Necessary Prior of Node N

Figure 2-11b

It is more difficult to determine which points on a complex branch are necessary priors of other points, since a complex branch always has at least one junction.

Consider, for instance, complex branch X(2),Z,BB, CC,DD,END. Point X is a necessary prior of point Z, but point Z is not a necessary prior of junction BB, since there is an alternate path to BB, specifically Y,BB. Continuing along the path, we next determine that BB is not a necessary prior of junction CC because there is another path to CC, specifically AA,CC. These observations lead us to another conclusion:



- A junction is never immediately preceded by a necessary prior.

: I .

Figure 2-12

For this reason, it is always necessary to go back at least to the nearest node in the direction of the initial node in order to make an unambiguous determination of how we have arrived at a junction.

The properties of a main section allow us to determine all the necessary priors of any point in a main section. In addition to those that we have already identified, here are some general rules that can be applied towards this end:

The initial node is a necessary prior of every other point.

A mandatory point is a necessary prior of every succeeding point.

The control node of a simple logical unit is a necessary prior of every member of its logical unit.

Just as some points must necessarily precede other points in a main section, so must some points necessarily succeed others. For example, any member of a branch, whether simple or complex, must necessarily follow - that is, it must be the NECESSARY SUCCESSOR of - each previous member, excluding the control node.

Consider branch C(2),E,F,G of Figure 2-13, for example. The necessary successors of point E are points F and G; for point F the necessary successor is point G. In practice, this means that if we know, for example, that C(2),E,F,G is an active branch, as it is in Figure Z-13, then we know that if we are at point E, we must pass through points F and G in order to eventually arrive at the terminal node, END, since F and G are necessary successors of E. It is important to note, however, that control node C is not necessarily succeeded by any of the members of the branch. This leads us to the following two conclusions:

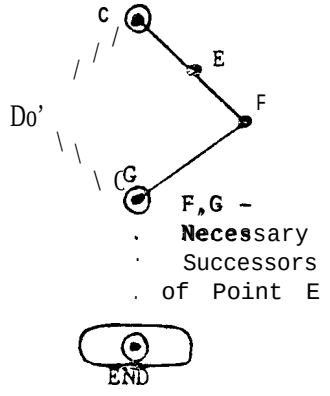


Figure 2-13

- The members of a branch are necessary successors of each of the preceding non-node members.

- A node, unless it is an initial node, is never immediately succeeded by a necessary successor.

Here are some additional rules that can be used to identify the necessary successors of any point in a main section:

- The terminal node is a necessary successor of every other point.

- A mandatory point is a necessary successor of every preceding point.

- A link node is a necessary successor of every member of every branch of which it is a member, with the exception of the control nodes.

Primary Nodes and Sections

Because a typical main section can be quite complex, with a proliferation of branches, logical units and junctions, it can be difficult to comprehend the characteristics of the entire structure. If we could view only a portion of a main section at any one moment, yet still obtain a good sense of where we are in the structure as a whole, we would be much better off. Fortunately, large main sections tend to be organized around smaller structures that exhibit properties very similar to those discussed above. The remaining discussion of logical structure components will focus on identifying these smaller structures and examining their characteristics.

A node that is a mandatory point is called a PRIMARY NODE and has several important properties. For instance, recall that a node that is on the active path always determines the next portion of the active path, and that a mandatory point is always on the active complete path. It follows, therefore, that a primary node always determines a portion of the active complete path. Thus, it is useful to think of each primary node as a major control center that ultimately determines how to get from one part of a main section to another.

A first step in evaluating a main section is to identify all of its primary nodes. The primary nodes of the main section of Figure 2-14, for example, are BEGIN,C,G,X and END. Note that all the paths that lead from one primary node ultimately lead to the next primary node. This is a general property of primary nodes that leads us to the following conclusions:

- A primary node is the necessary prior of each succeeding primary node.
- A primary node is the necessary successor of each preceding primary node.
- A primary node is the necessary prior of each succeeding point down to and including the next succeeding primary node.
- A primary node is the necessary successor of each preceding point up to and including the next previous primary node.

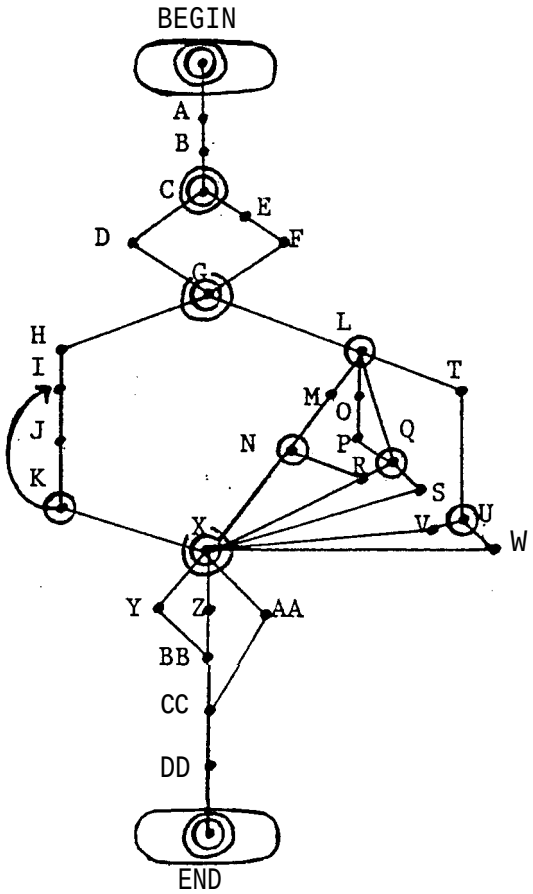
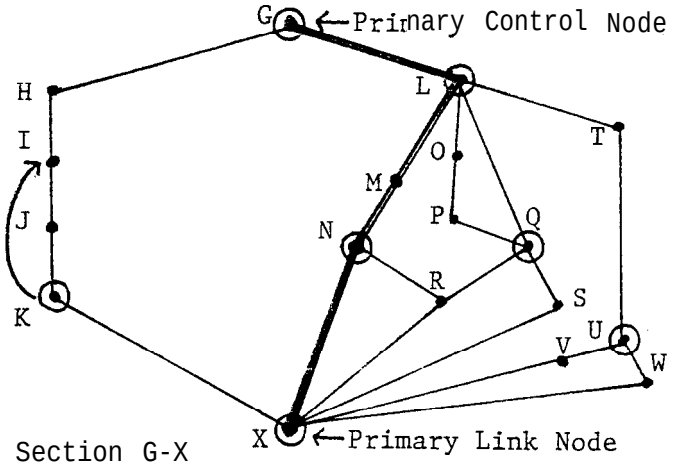


Figure i-14

We shall define all the points between two adjacent primary nodes, including the primary nodes, as a SECTION. For example, primary nodes G and X of Figure 2-15 define section G-X, which consists of the following points (members):

G, H, I, J, K, L, M, N, 0, P, Q, R, S, T, U, V, W and X

A section is very much like a main section, and has "similar properties. For example, a section always has a unique starting point called a PRIMARY CONTROL NODE and a unique ending point called a PRIMARY LINK NODE. All paths in the section emanate from the primary control node and converge at the primary link node. All paths in section G-X, for example, emanate from G and converge at X.



Section G-X

Active Section Path - G,L,M,N,X

Figure 2-15

There can be only one active path between a section's primary control node and its primary link node. This path, if it exists, is called the ACTIVE SECTION PATH, and is similar to the active complete path of the main section. In fact, if each and every section in a main section has an active section path, then we know there is an active complete path between the initial node and the terminal node.

Examining the properties of sections reveals much about which points are part of the mandatory path. We already know, for instance, that all primary nodes are part of the mandatory path. In addition, some sections have a special type of junction, called a PRIMARY JUNCTION, that is always on the mandatory path.

A primary junction is defined as a junction that is the initial intersection of all paths leading from the primary control node. For example, junction CC in section X-END of Figure 2-16 is a primary junction because all paths emanating from primary control node X converge at junction CC. Note also that point DD, which is on the path between primary junction CC and primary link node END, is a mandatory point.

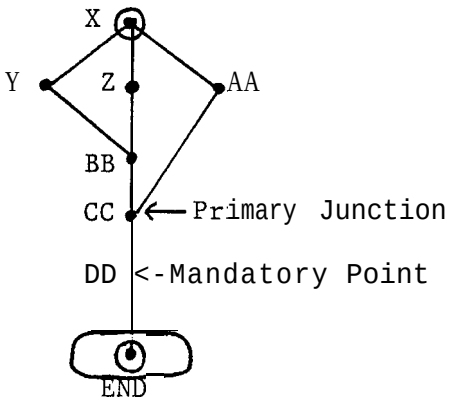


Figure 2-16

The following rules apply to all sections:

- There can never be more than one primary junction in a section.
- The primary junction is always the junction closest to the primary link node.
- Each point on the path between a primary junction and a primary link node is a mandatory point

The first section of a main section is called the INITIAL SECTION. The primary control node of the initial section is always the initial node. An initial section that has an initial node with only one branch has a unique property, specifically:

- Each member of an initial section with only one branch is a mandatory point.

Initial section BEGIN-C of Figure 2-17, for example, has a single path, BEGIN,A,B,C. Each of these points are mandatory and BEGIN,A,B,C is a mandatory path.

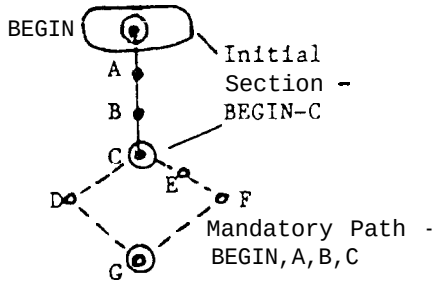


Figure 2-17

There is another rule that applies to any section that is not an initial section with a single path:

> In a section that is not an initial section with a single path the only mandatory points are the primary nodes, and, if present, the primary junction and any points between the primary junction and the primary link node.

A section is a ONE-IN-ONE-OUT structure, meaning that a single point controls the entrance to the structure and a single point controls the exit from the structure. In a section, these points are, respectively, the primary control node and the primary link node. Simple branches and mandatory paths are also one-in-one-out structures, as is another type of structure called a SUB-SECTION. A sub-section is a simple logical unit that has only one link node, for example, logical unit U of Figure 2-14. The table below summarizes the entrances to and exits from the various types of one-in-one-out structures.

STRUCTURE TYPE	ENTRANCE	EXIT
section	primary control node	primary link node
sub-section	control node	link node
simple branch	control node	link node
mandatory path	primary junction	link node

TABLE 2-1 Entrances and Exits to One-In-One-Out Structures

In the next chapter, we will use the properties that we have identified here to lay the groundwork for a CAT1 system design that incorporates many useful instrument management features.

USING PROPERTIES OF LOGICAL STRUCTURES IN INSTRUMENT MANAGEMENT APPLICATIONS

Now that we have identified many of the properties of logical structures, we are ready to use them in some practical instrument management applications. Specifically, we shall develop a model CAT1 system that can be used to determine:

- which instrument items have applicable responses or store applicable values
- how to evaluate logical expressions that have unanswered or inapplicable values
- how any given instrument item is logically related to any other instrument item
- how to allow interviewer jump requests without causing any undesirable side effects

Before we develop our model CAT1 system, let us review some of the problems commonly associated with CAT1 instrument management and design by taking a look at a "basic" CAT1 system.

The Basic CAT1 System and its Limitations

CAT1 systems vary in their design and capabilities, so it is difficult to define a "typical" system. There are elements of design that are common to every CAT1 system that the author is familiar with, however, so we will use these as the basis for defining a "basic" CAT1 system.

A basic CAT1 system is organized around the concept of an "item". A CAT1 item resembles a paper and pencil item in many respects in that it has a unique name, branching instructions and perhaps other attributes, such as text, response storage locations, a list of valid response codes, arithmetic calculations, and so on. However, the scope of each item is limited. An item contains the information that the CAT1 system needs for processing a particular portion of an instrument, but little or no information is provided about the item's relationship to other items. This system works well if an interview proceeds from the first item to each succeeding item based solely on each item's branching and other instructions and without disruption by interviewer commands. Still, precluding the use of interviewer commands is not an acceptable alternative in most circumstances. In order to get around this limitation, instruments written for a basic CAT1 system must incorporate defensive measures as a means of ensuring correctness. This places an added burden on the instrument designer and may lead to other complications. (See Chapter 1 for a discussion of these issues.)

An alternative is to provide the CAT1 instrument designer with a CAT1 system that minimizes the necessity for coding instruments defensively while imposing a minimum amount of restriction on the use of interviewer commands. One way of accomplishing this is to design a CAT1 system that goes beyond the concept of an instrument as simply a collection of items and towards a concept that

recognizes each item as a unique component of an instrument's logical structure. Our model CAT1 system, therefore, shall draw on the properties of logical structures.

A Model CAT1 System

In order to keep our model from getting too complicated, we shall make some assumptions about instrument design. First, we shall assume that all instrument instructions correctly reflect the intentions of the instrument designer and yield correct results if executed in the intended order and without external alteration. Second, we shall assume that all branching instructions are written using top-down branching, with the logical flow always proceeding in the direction of the terminal node. The single exception to this is for the allowance of simple loop-back branching. Finally, we shall assume that, in order for an interview to be considered complete, each section in the instrument must have an active complete section path. In other words, there are no optional sections.

Our model CAT1 system will have, as its primary focus of organization, the concept of a section as defined in the previous chapter. The discussion of the previous chapter revealed several very useful properties of sections, including:

> each item belongs to one and only one section. This makes it easy to determine the general location of any item. Also, a section contains all the information necessary for establishing the logical relationship between any two points in the section. Thus we can readily determine all the structures contained in a section, such as logical units and branches.
>
> the first item of a section unambiguously determines the active path through the section. Thus we can determine which points in the section are or are not on the active path simply by tracing the active path starting with the first item of the section.
>
> the first item of a section is always on the mandatory path. This makes it easy to identify the circumstances under which we can expect to 'arrive" at a section. It is also easy to determine where a section is in relation to any other section. For example, we can determine whether two sections are immediately adjacent, whether one section is closer to the initial node than another section, and so on.
>
> a section is a one-in-one-out structure. We can utilize this property if we choose, making it impossible to enter the section without the "permission" of the first item of the section. Likewise, we can make it impossible to leave the section without the "permission" of the last item of the section. We can also specify the type of movement allowed within the section. In short, because a section is a one-in-one-out structure, we can, if we choose, define a set of section access and control conditions.

In a basic CAT1 system, a CAT1 compiler, also called a translator or instrument builder, scans the set of instructions that are written by the instrument designer in programming language format, collecting the information

it needs to have on hand for each item. Our model CAT1 compiler will be designed to do this also, but in addition will have the task of collecting the structural information it needs to organize an instrument into sections. Here is a list of things our compiler must do in addition to its "basic" tasks:

- make a list of all items that are nodes (those items that have conditional branching instructions).

- starting with the initial node, make an ordered list of all nodes that are primary nodes (those nodes that are also mandatory points). This list, which concisely defines section names and boundaries, is called the Section List.

- create a Section Table for each section. In each Section Table, provide a list of the points (items) that are members of the section.

  Include the following information for each member:

  - type of member (point, node, junction, junction node)
  - type of storage location (unique, shared, none)
  - storage location pointer, or, if the member does not have a unique storage location, an "activation" flag that indicates whether the member has ever been on the active path
  - whether or not the member has a text display

  Include the following information for each node:

  - number of branches
  - branching instructions
  - for each branch, an ordered list of the names of each branch member
  - the names of branch members that are junctions or junction nodes

  Include the following information for each junction and junction node:

  - the number of logical units of which it is a member
  - the name of each logical unit of which it is a member

- create a Shared Storage Location Table. This table contains the following information for each shared storage location:

  - the name of the shared storage location
  - a pointer to the shared storage location
  - the number of items that share the storage location
  - the name of each item that shares the storage location

- collect item information in the usual way. However, to each item's table of information, add the name of the section of which it is a member, and whether or not it shares a storage location with any other items.

Table 3-1 shows how a Section List and a Section Table might look after using our model CAT1 compiler to translate the example instrument of Chapter 2, shown here in Figure 3-1.

```
SECTION TABLE G - X

    Member List Legend

        Position 1:  n = Node                 Position 2:  u = Unique Storage Location
                     j = Junction                          s = Shared Storage Location
                    jn = Junction Node                     n = No Storage Location
                     p = Regular Point

        Position 3:  if Position 2 is u : @loc is Storage Location    Position 4:  d = Display
                     if Position 2 is s or n :  0 = inactivated                    -d = No Display
                                                1 = activated

MEMBER LIST:  G(n,u,@G,d), H(p,s,0,d), I(j,u,@I,d), J(p,u,@J,d), K(n,s,0,-d), L(n,u,@L,d), M(p,u,@M,d), N(n,u,@N,d)
              O(p,u,@O,d), P(p,u,@P,d), T(p,u,@T,d), Q(jn,n,0,d), U(n,n,0,d), R(j,u,@R,d), S(p,u,@S,d), V(p,u,@V,d)
              W(p,u,@W,d), X(jn,n,0,d)

NODE LIST:    G(2)    /if @G EQ 1 goto H else goto L     /G(1),H,J,K;  G(2),L/    /I/
              K(2)    /if @K EQ 1 goto I else goto X     /K(1),I,J,K;  K(2),X/    /I,X/
              L(4)    /if @L EQ 1 goto M else            /L(1),K,M;  L(2),O,P,Q;  L(3),Q;  L(4),T,U/    /Q/
                       if @L EQ 2 goto O else            /                                      /    / /
                       if @L EQ 3 goto Q else goto T     /                                      /    / /
              M(2)    /if @M EQ 1 goto I else goto R     /M(1),I;  M(2),R,X/    /R,X/
              Q(2)    /if @L EQ 1 goto R else goto S     /Q(1),R,X;  Q(2),S,X/    /R,X/
              U(2)    /if @T EQ 1 goto V else goto W     /U(1),V,X;  U(2),W,X/    /X/
              X(n)    ////

JUNCTION LIST:
              I(2),G,K
              O(1),L
              X(4),K,M,O,U
              R(2),M,Q
```

TABLE 3-1 Reflects initial state of Section G-X as shown in Figure 3-1

An item from that instrument, item V, is shown in tabular form in Table 3-3. Note that it contains a reference to Section G-X. The Section List and Section Tables will be described in greater detail in the examples that follow.

ITEM TABLE V

Item Name: V
Text:
Branching: goto X
Valid Responses: 1 - 7
Storage: @V (unique)
Section: G - X

Table 3-3

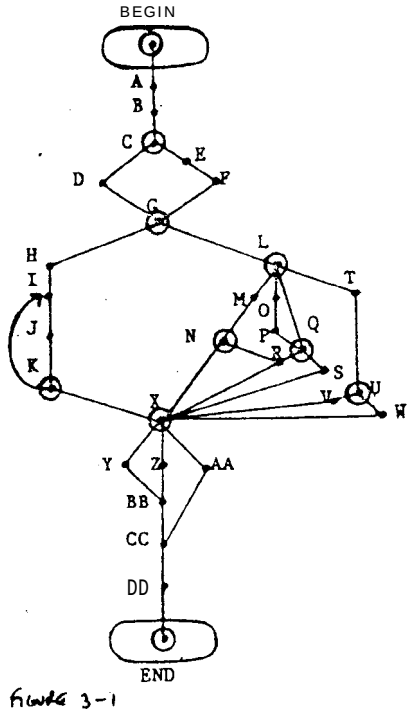


Figure 3-1

The Section List and Section Tables are information superstructures that link subordinate information structures -- items -- together in a way that relates directly to the structure of the instrument. As a result, more information is available to the CAT1 system than before. In the sections that follow, we shall demonstrate some of the ways our model CAT1 system can utilize this information.

The Active Path, Item Status, and Applicability

One of the functions associated with instrument management is to determine if a particular stored value is applicable. In order to do this, many CAT1 systems maintain an item status indicator for each displayable item that stores a response. Typically, item status has three values: unanswered (UN), answered and applicable (AA) and answered but inapplicable (AI). All displayable items that store responses begin as UN. When and if an item is answered, its status changes to AA. If the active path is altered so that the item is no longer on the active path, the item status becomes AI. If at a later time the active path is altered so that the item is once again included on the active path, the status again becomes AA, and so on (Nicholls and House, 1987).

Some items in a typical CAT1 instrument do not store responses, so another way must be found to track them. Rather than use item status flags, we will design our model CAT1 system to keep track of the active path in order to make determinations as to which stored values are applicable, and to maintain active path information about all types of items, including those that do not store responses.

The Section Tables described above are initialized with almost all the information the model CAT1 system needs to have in order to track the active path. To complete the picture, we will instruct the system to record whether or not a particular item has ever been on the active path, unless this information can be determined implicitly.

The procedure for determining if an item has ever been on the active path depends on the characteristics of the item. For example, if Q1 is the only item that stores a response at location @Q1, that is, if @Q1 is a UNIQUE STORAGE LOCATION, we need only examine the value stored at @Q1 to determine if it still has its initial value (referred to here as $NIL) or if it has another value.

Some items do not have unique storage locations but instead share their storage locations with other items, as would be the case, for instance, if items Q2 and Q3 both stored their responses in @Q2. @Q2 in this case is a SHARED STORAGE LOCATION. Other items do not store responses at all, and consequently have no associated storage locations. For items like these that do not have unique storage locations, we need to have a flag that indicates whether or not the item has ever been on the active path. We will instruct our model CAT1 system to set this flag, which we shall refer to as an ACTIVATION FLAG, whenever the item's final instructions are executed. This will be our way of marking the item as having been on the active path at least once. Normally an activation flag is never cleared.

We shall use the term ACTIVATED to indicate that an item has been on the active path at least once, and INACTIVATED to indicate that an item has never been on the active path. We will classify an item as activated if it meets any of the following conditions:

- the item has a unique storage location that contains a value that is not $NIL (its initial value)

- the item has a shared storage location and its associated activation flag is set

- the item does not have a storage location and its associated activation flag is set

If none of the above conditions are met, the item shall be classified as inactivated.

Using these rules, we derive the following procedure for determining if an item is activated:

1. Determine if the item has a storage location.
   If not, go to step 4.
2. Determine if the item has a shared storage location.
   If so, go to step 4.
3. Determine if the stored value is $NIL.
   If so, the item is inactivated.
   Else the item is activated.
4. Examine the item's activation flag.
   If the activation flag is set, the item is activated.
   Else the item is inactivated.

It is important to note that the classification of an item as "activated" does not imply that the value it stores, if any, is applicable. In order to make that determination, we must find out whether the item is on the active path. (Note one exception, however. Since our model assumes that all sections are mandatory, we can conclude that if a primary node is activated, it is on the active path and any value it stores is applicable.)

Using the Section Tables, we can determine if a particular point is on the active path. First we use the procedure described above to determine whether a particular item is activated. If it is, the next step is to find out if the item is currently on the active path. To do that, we shall use the following procedure:

Procedure for determining if an activated item is currently on the active path:

1. Determine which section the item belongs to.
2. Determine if the item is a primary node.
   If so, then the item is on the active path.
3. Determine if the primary control node is activated.
   If not, then the item is not on the active path.
4. Determine the control node's active branch.
5. Determine if the item is a member of the active branch.
   If so, then the item is on the active path.
6. Determine if the active branch link node is activated.
   If not, then the item is not on the active path.
7. Determine if the active branch link node is the primary link node.
   If so, then the item is not on the active path.
8. Consider the active branch link node as the next control node and go to step 4.

Let us work through an example. Assume that Figure 3-2 represents the active section path through section G-X and that its Section Table is the one shown in Table 3-6. Assume also that $NIL represents an initial value of -1 and that these values are stored in the following locations:

@G = 2
@L = 4
@T = 2
@V = 5

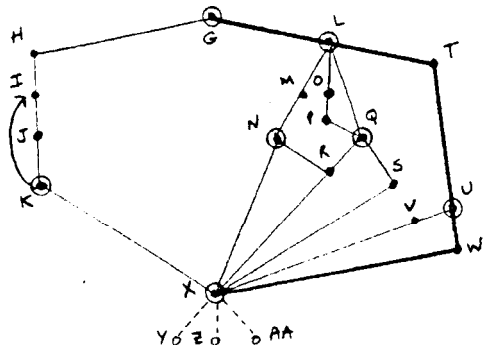


Figure 3-2

ITEM TABLE V,

Item Name: V
Text:
Branching: goto X
Valid Responses: 1 - 7
Storage: @V (unique)
Section: G - X

Table 3-5

SECTION TABLE G - X

Member List Legend

Position 1: n - Node; j - junction; Jn - Junction Node; p - Regular Point

Position 2: u - Unique storage Location; s - Shared Storage Location; n - No Storage Location

Position 3: if Position 2 is U : @loc is storage Location; if Position 2 is s or n : Q - inactivated; 1 - activated

Position 4: d - Display; -d - No Display

MEMBER LIST: G(n,u,@G,d), H(p.s.0.d). I(j,u,@I,d), J(p,u.@J.d). K(n.s.O.-d), L(n.u,@L.d). M(p.u.@M,d). N(n,u.@N.d)
O(p.u.@O.d). P(p.u.@P.d). T(p.u.@T,d). Q(jn.n.0.d). U(n.n.1.d). R(j.u.@R.d). S(p.u,@S.d). V(p.u.@V,d)
W(p,u,@V,d), X(jn.n,l,d)

NODE LIST:	C(2)	/if @G EQ 1 goto H else goto L	/G(l).H.J.K:	G(2).L/	/I/	
	K(2)	/if @H EQ 1 goto I else goto X	/K(l).I,J.K;	K(2).X/	/I,X/	
	L(4)	/if @L EQ 1 goto M • lme if @L EQ 2 goto 0 else if @L EQ 3 goto Q else goto T	L(2).O.P.Q; /L(1),M,N; /	L(3).Q:	L(4).T,U/ / /	/Q/ I I I I
	N(2)	/if @N EQ I goto X else goto R	/N(l).X;	N(2).R,X/	/R.X/	
	Q(2)	/if @L EQ 1 goto R else goto S	/Q(l).R.X;	Q(2).S.X/	/R.X/	
	U(2)	/if @T EQ 1 goto V else goto W	/U(l).V.X;	U(2).U,X/	/X/	
	X(n)	////				

JUNCTION LIST:

I(2) ,G,K
Q(1) .L
X(4) .K,N,Q,U
R(2).N.Q

TABLE 3-6 Reflects state of section as shown in Figure 3-2

Suppose we want to find out if item V is on the active path. We first need to determine if item V is activated. Referring to item V's item table (Table 3-5), we determine that it has a unique storage location, @V. Next we compare the value of @V with $NIL. Since 5 is not equal to -1, we conclude that item V is activated.

Now we can use the procedure for determining if an item is on the active path. We will work through the procedure step by step. (The step numbers in parentheses refer to the corresponding step numbers of the active path procedure described above.)

1. (Step 1) From V's Item Table we determine that V belongs to section G-X.
2. (Step 2) Since V is not G or X, it is not a primary node.
3. (Step 3) Locate G in Section Table Member List. From Member List, we determine that @G is a unique storage location. Since @G = 2, G is activated.
4. (Step 4) Locate G in Node List. Evaluate conditional branching instructions. Since @G=2, the active branch is G(2),L.
5. (Step 5) V is not a member of G(2),L.
6. (Step 6) From Member List, we determine that @L is a unique storage location. Since @L=4, active branch link node L is activated.
7. (Step 7) Since L is not X, L is not primary link node.
8. (Step 8) Consider L as next control node.
9. (Step 4) Locate L in Node List. Evaluate conditional branching instructions. Since @L=4, the active branch is L(4),T,U.
10. (Step 5) V is not a member of L(4),T,U.
11. (Step 6) From Member List, we determine that U has no storage location and that its activation flag is set. Therefore U is activated.
12. (Step 7) Since U is not X, U is not a primary link node.
13. (Step 8) Consider U as next control node.
14. (Step 4) Locate U in Node List. Evaluate conditional branching instructions. Since @T=2, the active branch is U(2),W,X.
15. (Step 5) V is not a member of U(2),W,X.
16. (Step 6) From Member List, we determine that X has no storage location and that its activation flag is set. Therefore X is activated.
17. (Step 7) X is a primary link node. We conclude that V is not on the active path.

Evaluating Conditional Instructions

Assume that item A stores an applicable value in location @A and that it contains the following conditional branching instruction:

IF (A EQ 1) THEN GOTO B ELSE GOTO C

If A represents a value of 1, then the logical expression (A EQ 1) is true and the true condition THEN GOTO B is satisfied. If A represents any other value, then the false condition ELSE GOTO C is satisfied.

The evaluation of a logical expression consisting entirely of applicable values is straightforward. Problems arise, however, if unanswered or inapplicable values are part of a logical expression. If A is unanswered, its initial value can be used in evaluating the logical expression. But what value should be used for A if A is inapplicable? If, for example A stores a value of 1, should the value of A be-evaluated as' l? The answer is clearly no.

A is really on

of 1, should the value of A be evaluated as 1? The answer is clearly no. Unfortunately, however, a basic CAT1 system would indeed evaluate A as 1. The correct approach is to evaluate an inapplicable response as if it were unanswered; that is, to use its initial value. We shall instruct our model CAT1 system, therefore, to obey this rule for evaluating logical expressions:

```
If i is part of a logical expression
      then if @i is applicable
            then i <-- @i
            else i <-- $NIL
```

where:

@i is the contents of the storage location associated with i
$NIL is i's initial value

Simply stated, this means that when evaluating a storage location that is referenced as part of a logical expression, the stored value is used if the value is applicable, and the initial value is used if the value is inapplicable or unanswered.

In order to avoid confusing the initial value with stored responses, we shall have our CAT1 system initialize storage locations with values that never represent responses, for example a -1 for binary data types and " " (space) for character data types.

Enforcing these rules allows the CAT1 instrument designer to avoid testing explicitly to see if an item is on the active path, so long as he is aware of the existence of $NIL, the initial value. For example, instead of coding:

```
IF (A EQ $NIL)
      THEN
            GOTO C                                  ;if A is not applicable
      ELSE
            IF (A EQ 5) GOTO B ELSE GOTO C          ;if A is applicable
```

he can code instead:

```
IF (A EQ 5) GOTO B ELSE GOTO C
```

In this case, (A EQ 5) is true only if @A has a value of 5 and is applicable. Any other applicable value fails the test, as do inapplicable and unanswered values, since they are evaluated as $NIL (-1).

Though this scheme allows the instrument designer to completely avoid testing explicitly for applicability, in some cases instructions may be more readable if' explicit tests for applicability are included. This is accomplished by explicitly testing for the'value $NIL. For example:

```
IF (A EQ $NIL)                          ;if A is unanswered/inapplicable
      THEN
            GOTO D
      ELSE                              ;if A is applicable
            IF (A LT 3) GOTO B ELSE GOTO C
```

The statements below accomplish the same thing without testing for $NIL:

```
IF (A GE 3)
      THEN
            GOT0 C
      ELSE
            IF (A GE 0) GOT0 B ELSE GOT0 D
```

Rules for Allowing Random Movement - Navigating a Logical Structure

We shall use the term BRANCHING INSTRUCTION to refer to the logical flow specifications that are present in the instrument itself, as opposed to the term JUMP REQUEST, which shall refer to an action initiated by a person such as an interviewer for the purpose of manually altering the order of items. Similarly, the term BRANCH refers to a determination of logical flow specified by a branching instruction, whereas the term JUMP refers to a manual change of the order of items initiated by a jump request.

The item that is displayed on an interviewer's screen pending a required action by the interviewer is called the CURRENT ITEM. The item specified by an interviewer as part of a jump request is called the TARGET ITEM. A jump request that is allowed by the CAT1 system is referred to as a GRANTED request, otherwise it is a DENIED request. Upon completion of a granted request, either the target item or another, more suitable item, becomes the new current item, called the GRANTED ITEM. The only item instructions that are executed as a result of a jump request are the granted item's initial instructions and text display. If a request is denied, no instructions are executed and the current item remains the same.

Movement from one part of an instrument to another shall be viewed with respect to the initial and terminal nodes. Thus a jump from the current item to the target item in the direction of the terminal node (away from the initial node) is FORWARD movement; a jump from the current item to the target item in the direction of the initial node (away from the terminal node) is BACKWARD movement.

In order to navigate a logical structure successfully, it is necessary to establish some reasonable rules of the road and then adhere to them. Therefore, our model will conform to the following general rules:

(1) A granted item must be a suitable current item, that is, it must have a text display that requires some action on the part of an interviewer

(2) A granted item must be on the active path

(3) If a jump request for a target item cannot be granted, then a suitable alternate item. if one exists. shall be used instead. This item shall be the closest prior item relative to the target item that meets the requirements of rules (1) and (2).

Rule (1) ensures that the granted item cannot be a hidden item that is not visible to the interviewer, such as an "instruction-only" item.

By adhering to rule (2) we ensure that it will not be possible for an interviewer to jump to an item that is not on the active path. Thus it will always be possible to determine which items are on the active path by following the logic of the instrument's branching instructions.

Rule (3) makes it easier for interviewers to use jump commands successfully without having to be concerned about whether or not their target item is on the active path. By automatically providing the interviewer with a suitable alternative to the tarqet item when necessary, the CAT1 system makes it possible for the interviewer to reach the target item eventually.

For example, suppose an interviewer is faced with the set of circumstances shown in Figure 3-3. Here, the current item is D, and the interviewer wants to jump back to item C. However, when item A was answered, the active path A,B was established, leaving item C, the target item, off the active path. The jump request for item C, therefore, must be denied according to rule (2). At this point, the CAT1 system has two ways in which to respond: the jump request can be denied and the current item remain unchanged, or access can be granted to a suitable alternate item that has a potential path to the target item. Rule (3) requires that our CAT1 system find a suitable alternative if one exists.

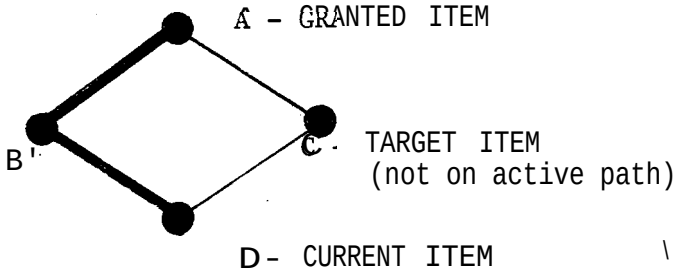


Figure 3-3

In our example of Figure 3-3, therefore, item A is the item granted as a result of the interviewer's jump request. By altering, as appropriate, the response to item A, the interviewer can cause item C to become the current item.

Now that we have defined our CAT1 system's instrument navigation rules, we have the information we need to specify a set of instrument navigation commands. These shall be:

FORWARD - makes the next forward item on the active path the current item. If the current item is the forward-most item on the active path, it has no effect

BACKWARD - makes the next backward item on the active path the current item. If the current item is the backward-most item on the active path, it has no effect

JUMP [trg] - requests a jump to target item trg, where trg is the name of an existing instrument item. The request is evaluated and acted upon according to the navigation rules described above

JUMP $BGN - jumps to the backward-most item on the active path

JUMP $END - jumps to the forward-most item on the active path

Let us see how our model CAT1 system handles the JUMP [trg] command.

Here is a procedure for the JUMP [trg] command:

1. Determine if trg represents the name of a valid instrument item. If not, report the error to the interviewer, else use trg as the target item and continue.
2. If the target item is a non-displayable item, then go to step 5.
3. If the target item is not on the active path, then go to step 5.
4. Make the target item the granted item. Go to step 7.
5. Determine if there is a displayable prior item on the active path with a path to the target item. If not, notify the interviewer that the jump request could not be granted, else go to step 6.
6. Make the alternate item the granted item. Go to step 7.
7. Make the granted item the current item and return control to the interviewer.

Since we have already developed a procedure for determining if an item is on the active path, we can apply it to step 3 of the above procedure. We can also use the Section List and Section Tables to determine which item, if any, satisfies Step 5. For example:

1. If the primary control node of the target item is inactivated, then use the forward-most displayable item on the active path as the alternate item.
2. Else determine the forward-most activated node with a path (not necessarily active) to the target node.
3. If the path from this node is active, then use the forward-most displayable item as the alternate item.
4. Else if the forward-most node is displayable, then use it as the alternate item.
5. Else determine the next-previous activated node with a path to the target item and repeat steps 3,4 and 5 until an alternate item is found.

The Section Tables can be utilized in a similar fashion to implement each of the other instrument navigation commands.

Obtaining Final Data Sets, Context Diagrams and Structural Information

The model CAT1 system described here uses instrument structural logic to determine which items are on the active path. Values, once stored, are never erased or re-initialized (though they may be overwritten by new values). At any given point during an interview, valid data are those values stored by items that are on the active path. Thus, once an interview is complete (all sections have complete active section paths) the final data set is obtained by writing out the contents of all storage locations belonging to items that are on the active complete path.

Context diagrams that show the structure of the instrument in various formats and stages may be obtained by formatting the information available in each of the Section Tables. In some contexts, such as when responding to interviewer help requests, it may be desirable to show only the current section, or to

show only a portion of the active path, or perhaps just a complicated segment such as a complex branch. The instrument designer will want a complete diagram, perhaps for debugging or as an aid in re-designing portions of the instrument. In addition to visual information, the instrument designer can obtain a description of each key item, such as those items that are various types of nodes and/or junctions.

Controlling Access to One-In-One-Out Structures: A Modification to the Model

A section is important not only as a one-in-one-out logical construct but as an application-dependent entity. For example, a lengthy instrument may contain several sections that each have items that normally are asked as part of an interview. It may be desirable under certain circumstances, however, such as when the interviewer must interview an impatient respondent, to consider some of the sections as optional so that other sections are more likely to be included as part of the interview. A section consisting of demographic items, for instance, might be an example of a section that should always be part of an interview, even if it means skipping over certain other sections.

Because sections play an important role in application-dependent contexts, we might want to consider modifying our model CAT1 system to allow sections to be optional. By adding information to the section tables, we can incorporate the specification of section access conditions. For example, a field could be added to the section table specifying that the section is either MANDATORY or OPTIONAL, two mutually-exclusive attributes that can be defined as follows:

MANDATORY - the section must have a complete active section path in order for the main section to have a complete active path

OPTIONAL - the section need not have a complete section path in order for the main section to have a complete active path

Sections often consist of subordinate one-in-one-out structures such as simple branches, sub-sections, and mandatory paths. By setting up access fields in the section table for each of these structures, it is possible to control access to them in much the same way as we would control access to sections. By deliberately creating one-in-one-out structures, instrument designers could use this capability to protect critical portions of an instrument at the CAT1 system level, if, for instance, we define two additional access attributes, LOCKED and UNLOCKED:

LOCKED - the structure cannot be accessed. This is a dynamically managed attribute that can be toggled between LOCKED and UNLOCKED

UNLOCKED - the structure can be accessed. This is a dynamically managed attribute that can be toggled between LOCKED and UNLOCKED

For example, consider a portion of an instrument that consists of several items that perform calculations. The calculations are performed correctly the first time the items are executed, but produce incorrect results if they are executed again. If the instrument designer chooses to organize the items so

is reallocn

that they are part of a one-in-one-out-structure, then access to this structure can be restricted by declaring it LOCKED after exiting it once.

Another switch, RE-INITIALIZE, could be added. If set, RE-INITIALIZE would replace any values associated with items that belong to a specified structure with their initial values. This switch might be useful in order to re-calculate variables reliably without destroying stored responses.

CONCLUSIONS

The instrument designer and the CAT1 system both benefit when instrument management design is based on the properties of logical structures. The benefits are two-fold:

First, CAT1 instrument management capabilities are readily derived from the database created as a result of compiling instrument structural data. These additional capabilities can then be used by instrument designers as tools for programming CAT1 instruments.

Second, the compiled database is useful as a source of information for man as well as for machine. For instance, the instrument designer can use the structural information in the database as an aid for instrument design.

Beyond this, the CAT1 system designer should recognize that there are additional advantages to the design approach described here. Because more information is gathered at compile time, run-time decisions can be based more completely on read-only information and data collected as part of the interviewing process rather than on run-time manipulations of stacks, flags or arrays. For example, the model CAT1 system described above avoids the use of item status flags by drawing instead on information available from the section tables and interview data, resorting to the use of activation flags only where absolutely necessary. Even so, the activation flag for any item need only be updated once. Item status flags, on the other hand, must be maintained and continually updated for all displayable items. Further, because they are merely flags, they convey no real information.

As another example, it was demonstrated that the section tables can be accessed on a read-only basis in order to validate or invalidate interviewer jump requests. The procedure works without requiring the dynamic allocation of memory, and thus there are no limitations on the number of consecutive commands that may be entered (as when continually backing up, for example). Compare this to those systems that require the use of a stack. Once stack space is exhausted, or if stack space is not correctly managed, the system will not accurately reflect the current state of the interview.

Finally, the implementation of a model similar to the one discussed above may be tailored to the requirements and limitations of the hardware involved. For example, where main memory is limited, only one or a few section tables need be present in memory at any one time; on larger systems, all section tables can be kept memory-resident. (The author believes that it is possible to keep the size of section tables very small, on the order of a few hundred bytes.)

REFERENCES

Nicholls, William L., II, and Groves, Robert M. "The status of computer-assisted telephone interviewing: Part I--Introduction and impact on cost and timeliness of survey data." JOURNAL OF OFFICIAL STATISTICS, 2 (No. 2, 1986), 93-115.

Nicholls, William L., II, and House, Carol C. "Designing questionnaires for computer-assisted interviewing: A focus on program correctness." 3rd Annual Research Conference, Department of Commerce, Bureau of Census, Baltimore, Md., 1987, 95-111.

Nicholls, William L., II; Lavender, George A.; and Shanks, J. Merrill. "An overview of Berkeley SRC CATI." SRC Working Paper 31. Berkeley: Survey Research Center, University of California, February 1980.

Palit, Charles, and Sharp, Harry. "Microcomputer-assisted telephone interviewing." SOCIOLOGICAL METHODS & RESEARCH, 12 (November 1983), 169-89.

Shanks, J. Merrill. "The current status of computer-assisted telephone interviewing: Recent progress and future prospects." SOCIOLOGICAL METHODS & RESEARCH, 12 (November 1983), 119-42.