1. DATA STRUCTURE

The AHI data structure has as its basic unit the "statement." The statement is the smallest textual unit defined, and is simply a textual string. The file (i.e., collection of statements) is hierarchically oriented in a tree structure, each statement being a node in the tree. The reasons for this hierarchical structure will be discussed later. The file, however, can be viewed in other ways different from the sequential tree structure. For instance, associational trails can be drawn throughout the file and followed. Thus the AHI file is capable of modeling Bush's[1] network of associational trails as well as a sequential hierarchical text.

1.1 THREE TYPES OF BASIC ENTITIES

A. Statements

B. Vectors

C. Keywords

These three types of entities are stored in statement data blocks (SDB's), vector blocks, and keyword blocks, respectively. In addition, the hierarchical structure of the text is stored in ring blocks. We will only discuss the statement data blocks and ring blocks and their relation to the main file; the access and storage of vectors and keywords is very similar and so do not need to be discussed separately. (In the present version of the system, the only types of vectors that can be stored are straight lines, and no sketching facility exists other than defining the straight line by its endpoints. There is no rubberbanding. A sketching facility is planned for a future version.)

---

1. V. Bush, "As We May Think", Atlantic Monthly, July 45.

1.2 STATUS TABLES

all status tables are in the file header (block 0)

Associated with each set of blocks (there are four sets of blocks: the ring blocks, the statement data blocks, the vector blocks, and the keyword blocks) there is a small status table which has an entry for each block of its kind. Thus there is a ring status table, a statement status table, a vector status table, and a keyword status table. The entry for each block in the status table simply points to a "global" random file status table block, which gives the location of each block, whether in core or on drum. (See Fig. 1)

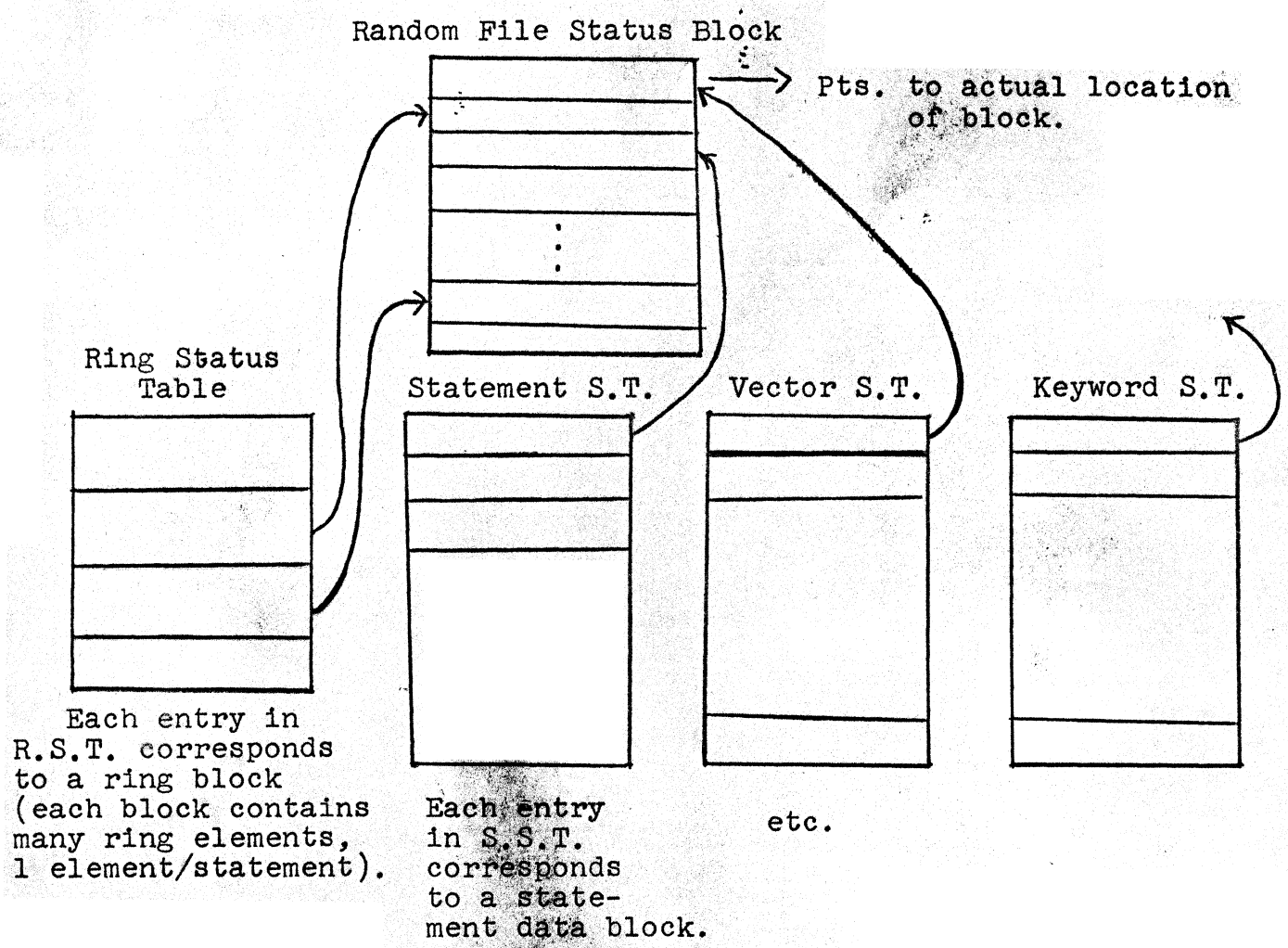

FIG. 1 Status Tables

1.3 RANDOM FILE STATUS TABLE BLOCK

The random file status table block is a block that contains an entry for every block of every type in the system (actually, there is an RFSTB for each (active) file). Each entry tells whether the block is in core or not, or whether it is unallocated (i.e., not being used at the present time and can be allocated when a file is expanded through editing). The entry also gives the information on where the block is located, on the drum or in core. It is through the RFSTB that each desired block is located by the system:

As we saw in 1.2, each ring block is mapped into an entry in the ring status table, each statement data block is mapped into an entry in the statement status table, etc. Then each entry in each status table points to an entry in the bigger RFSTB (at present there are a maximum of 64 blocks in the RFSTB), one containing pointers to the actual location of each different block in the file. This double-table method of location of each block is to facilitate control of the allocated and unallocated area on drum, and for garbage collection; furthermore, this central location mechanism allows blocks to be moved in the system, without internal pointers having to be modified.

The RFSTB contains information other than just a pointer to the block, whether in core or on drum. One area of the RFSTB, if it is less than zero, indicates the block is on the drum. If the number in this area is negative, it is the number of free words in that block. This is to prevent needless retrieval of the block for additions if there is not enough room on it for the desired update. If this area is greater than zero, the block is in core and the number is the core address of the block. There is another area in which an indicator (at present a-2) says free space is too small to consider going there. This is computed from the average lengths of the statements.

The ring status table at present has four entries, one for each of four ring blocks. This number is expandable, and is an assembly time variable. There are probably twice or three times as many SDB's and therefore the statement status table is correspondingly bigger.

1.4 STRUCTURE OF THE RING BLOCK

Each statement is represented in the data structure both by its associated text (see SDB's) and by a ring element, that is,

note: pointers to ring elements are called PSID's (permanent statement ID) + never changes during the existance of a statement.

by an element of a ring that contains the hierarchical tree structure of the file and points to the text associated with each node (statement) in the tree. The ring is broken into ring blocks, each of which is 1024 words long. Each ring block has a header and then is composed of ring elements, each four words long, one ring element per statement (See Fig. 2):

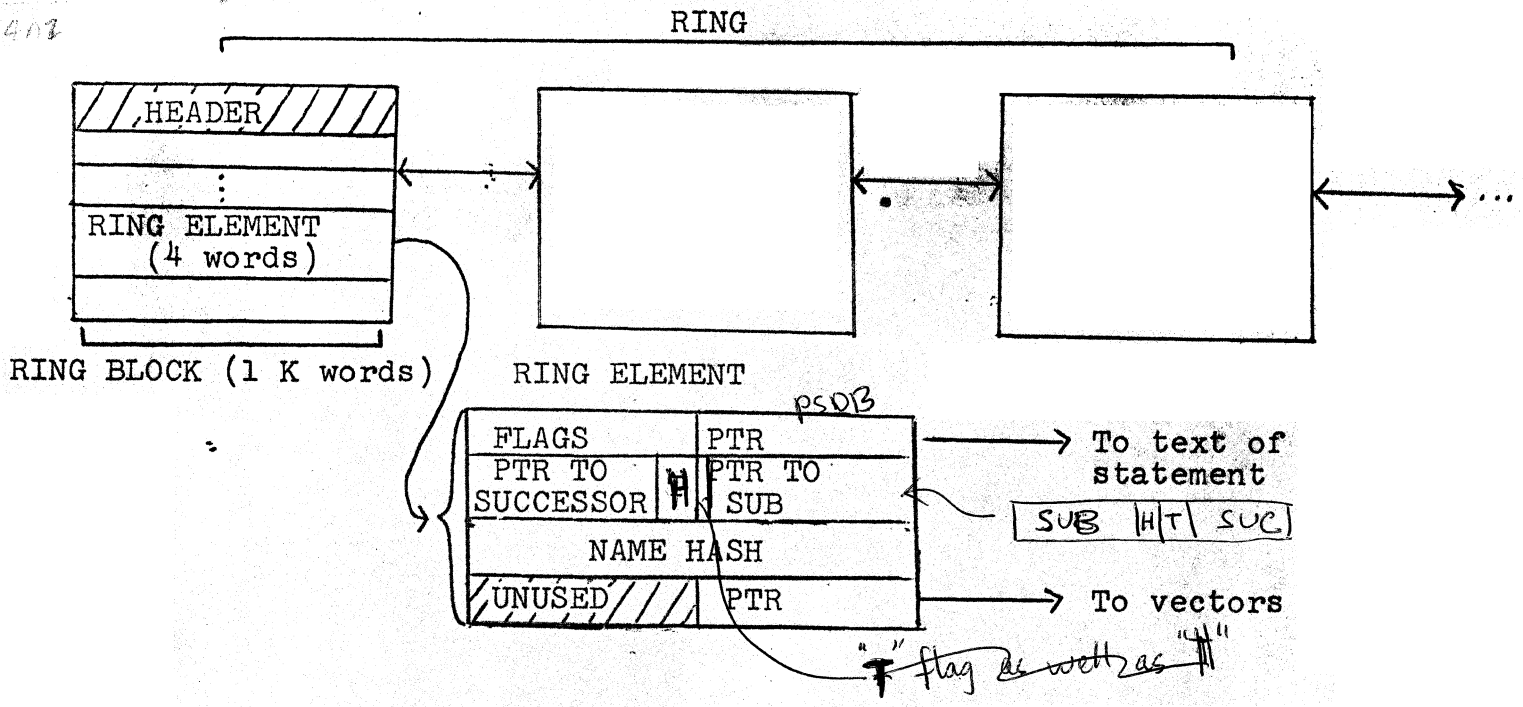

FIG. 2 Ring Block Elements

(PSDB)

Pointer (PTR): the internal pointer to the statement text in a statement data block (SDB). (The structure of an internal pointer (symbolized by 'P') will be discussed in Section 1.6.)

Flags: tell the level (in hierarchy), etc., and thus help with filtering so as to minimize drum I/O, i.e., if we want only to look at file above the third level of hierarchy and this statement is in the fourth level, there is no need to retrieve the text of this statement. (this would require updating the entire ring after every structure change).

Successor: a pointer to the ring element (anywhere in the ring) of the next succeeding statement on the same level.

Sub: a pointer to the ring element of the first statement in the level directly below the current statement.

The sub and successor pointers define the hierarchical tree structure.

SUB points to this ring element if there is no substructure

T: the last successor on each level points to the ~~head~~ source (up 1 level) of that level, thus providing a back pointer. The T bit is set when it is the last successor.

Name hash: this is a 24-bit hash of the (optional) statement name. Thus when jumping by name, we need only scan each ring element sequentialy to get correct statement. This may be done sequentially since each file generally consists of only about 300 statements.

note: unused ring elements ~~are kept~~ in a given ring block are linked together on a free list – pointed to by a pointer in the block header.

1.5 STATEMENT DATA BLOCK (SDB)

The statement data blocks (SDB's) are simply areas in which to store the statement text. No structure or hierarchy is part of the SDB's since that is taken care of by the ring blocks. The system tries to put all sequential statements in the same block to save on drum I/O. The process of initial generation and placement of statements in the SDB's will be discussed in section 2. (See Fig. 3)

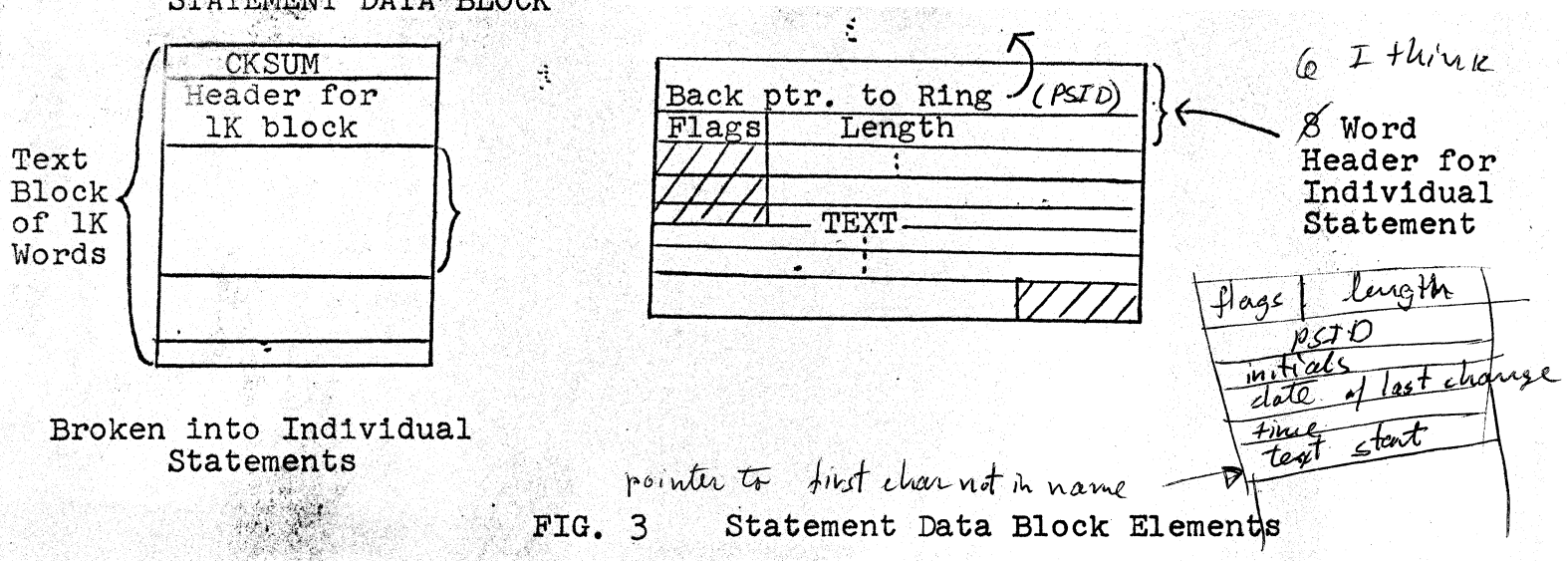

FIG. 3 Statement Data Block Elements

CKSUM: this is a checksum to check against hardware I/O errors in reading the statement data block from drum. Before writing out on drum, the system adds up all words in the SDB and stores the sum in CKSUM. On read-in, it re-adds the words and checks to see if the sum is the same.

All 1K file blocks are checksummed in this way

Header: this contains the initials, date, and time of last user and change, fields which can be used as a later means of retrieval.

Back pointer to ring: this is an internal pointer (of type 'P') to the ring element representing this statement in the text hierarchy, i.e., the ring element which points to this statement.

Flags: the first bit indicates whether this SDB element is garbage, and is used when compacting the SDB. Other bits indicate whether it is difficult to format the statement on the display, that is, if the statement contains things like underlining or flicker. If it is difficult to format, the low speed scanning/formatting routine is used. Otherwise the high-speed routine is used. This saves up to 50% on time. (Other bits for other things.)

Text: the text is stored in the statement data block as follows: there are two kinds of characters, (1) 8 bit character, with the high order bit off, and (2) 16 bit characters. If the high order bit is on, this signals that the character is a 16 bit character. The seven next high bits signify font, etc., of the character represented in the second eight bits. The different qualities of each character are underline, blinking, italics, boldface, etc. The user can make up his own special characters and the system will insert it. This is done by giving the special character a number. It takes less than 10 msec. to reformat a display.

1.6 LOCATING STATEMENTS IN THE DATA STRUCTURE

1.6.1 Mapping Statements to Ring Block Elements Through the Internal Name (Pointer) P

When statements are created, they are assigned by the freelist allocator to open positions in a ring block (to a ring element) and assigned to statement data blocks according to the "garbage bits"; they are also assigned an internal (position related) name in the ring block denoted by 'P'. All ring block vacancies are kept on the freelist. The internal name P in the ring block is thus assigned by getting it off the freelist (creating a map from statement name position to internal name, and from internal name to block position) as described below:

Say that we want to retrieve statement P, an internal name (pointer) of the type found in the successor and sub fields of the ring element. It is listed in the file header where it gives the point in the ring where the file starts, i.e., it points to the ring element representing the first statement in the file.

To get statement P (10 bits) we look at P*4, which is 12 bits long (See Fig. 4). The upper 2 bits are an index on the 4-entry ring status table (RST). The entry in the RST points to an entry in the random file status table block (RFSTB). This entry in the RFSTB tells us whether the ring block containg the desired ring element is in core or not, or whether it is unallocated (in which case an error condition exists). The ring block is brought into core if necessary. The lower 10 bits of P*4 then form an index relative to the start of the ring block that bring us to the appropriate ring element. Thus from the internal name of the statement we retrieve the desired ring element.

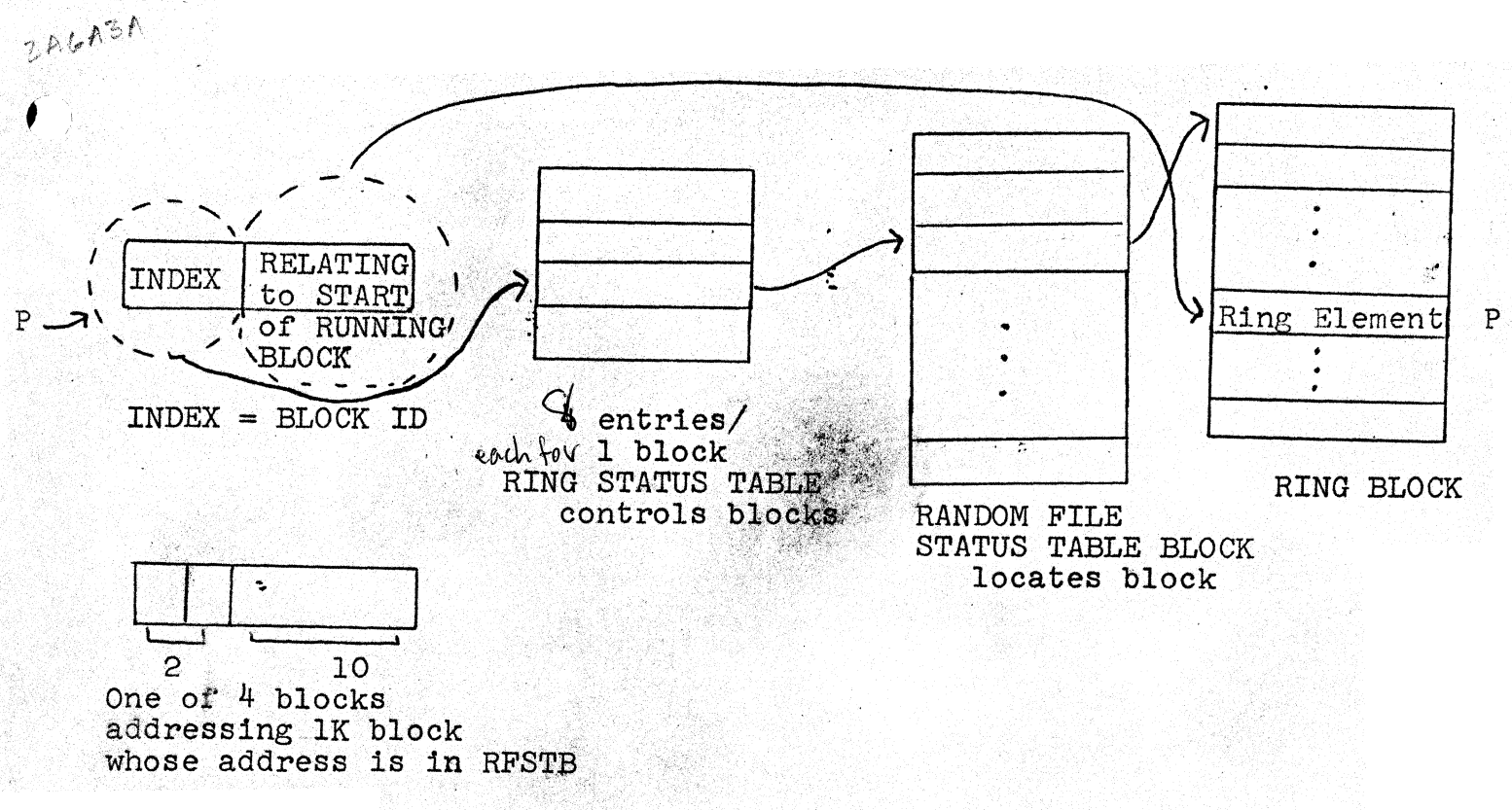

FIG. 4 Structure and Mapping of Internal Pointer 'P'

Notes on the file header: this contains pointers to all status tables and their lengths, and information on the virtual memory map. It also contains bibliographic information which may be used as a means of retrieval: last time written into, username, initials of last user, jump delimiters (these are the marks that delineate a jump, and in the present version are general parentheses), average length of statement (determined by how much activity over periods of time). This information is all contained in the first 1K words. An interesting feature is that the TS system will accept any amount extendable to 1K without using excess drum space.

The first ring element at the start is 'dummy'; and is the start of the file. When the system rewrites the file on drum after use, it searches to the first semicolon and puts in place of what is there the file description: username, initials, date and time, etc. of last use.

1.6.2 Mapping From the Ring Element to the Text of the Statement (See Fig. 5)

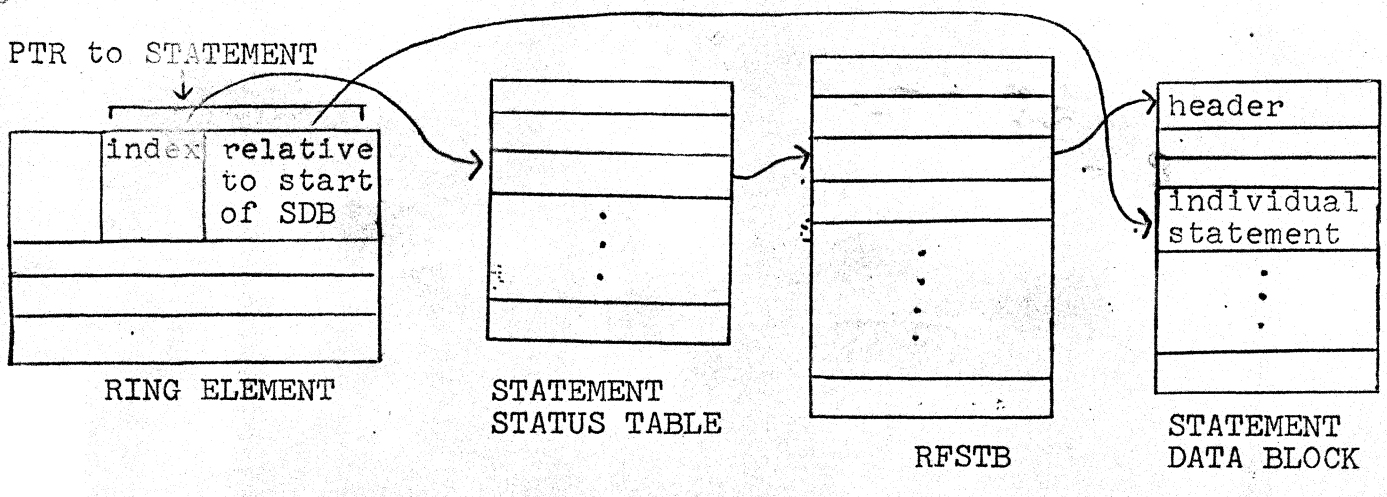

FIG. 5 Mapping from Ring Element to Text

Now that we have the appropriate ring element relating to statement P (see Fig. 2) for the structure of ring element, we can retrieve the statement-text for filter/format/display. The

system takes the "pointer to text" in the first word of the ring element. This pointer is of the same structure with respect to the statement in the statement data block as P*4 is to the ring element of the ring block. Thus the high bits are an index on the statement status table. The entry in the SST points to an entry in the RFSTB, which in turn points to the location of the appropriate SDB in which the desired statement is located. Once we have the appropriate SDB, the 10 low order bits of the original pointer point to the desired statement, relative to the start of the SDB. (See Fig. 3)

1.6.3 Generating a Sequence of Statements

Given the appropriate individual statement P (see Fig. 3 for structure of the statement entity), the sequence generator now takes the statement text for filtering/formating/display, as described in Section 3.

Which statement is taken next depends on the sequence being followed by the sequence generator. If the sequence generator is following the basic hierarchical tree structure, it will look at the ptr-to-sub field in the ring element (Fig. 2), and use that pointer as it used P above. (However, if a filter is set for a specific level and statement P was on that level, the sequence generator will ignore the sub field and take the ptr-to-successor field. ~~The sequence generator can tell the level of the statement P by the flag set the in the ring element.~~) has to find the level of the first st. & keep track of it

The sequence generator, however, may be following an associational trail. If this is the case, the content analyzer will scan the statement-text P for the appropriate trail marker. If it finds the appropriate trail marker in the statement-text, it will hash the name in the trail marker, and scan the name hashes of the ring elements until it finds the correct ring element, and continue generating statements from there. If the appropriate trail marker is not found, it will follow the tree structure as able. (- by hierarchy until a trail marker is found)

2. DATA STRUCTURE MODIFICATION

The data structure is modified through the basic editing commands (delete, insert, replace, move, copy, break/join) which are described briefly below in Section 4. System features and facilities are described more completely in the "NLS User's Guide" (a SRI publication).

We will describe how the data structure is modified for an insert; the other types of edit-modifications are all similar. If the edit is an insert, it is an insert in the middle of a statement. By system definition, all editing is based on the statement. The user types in the appropriate insert command, hits the point of insert with the mouse, and types in the insert. The insert typed appears on the screen in the literal type-in area. If the user decides the insert is complete, he hits the command-accept button. The system then makes the modification of the data structure as follows:

The system computes the new length of the statement by taking the old length of the statement in the statement data block and adding the length of the insert. The system then finds a free area on one of the statement data blocks of sufficient length to put the new statement. It tries to put the updated statement on the same SDB. If the edited statement does not fit in that SDB, the system tries to compact the block. If that would not give enough space, the system goes to the previous ring element and sees what SDB that statement is stored on and tries to fit the newly updated statement on that block. If it doesn't fit there, the system looks through the SDBST to find any free area and fits it in anyplace.

Now that the appropriate space is allocated, the updated statement is constructed. This is done by copying the header of the original statement and the text up to the insert point, adding to this the literal type-in, and copying the rest of the text of the statement. Then the "ptr-to-text" in the associated ring element is changed to point to the new statement, and the garbage bit is set in the original statement.

When any statement is edited, the system checks to see if there is a statement name, or label. If there is, it is rehashed and replaced in the ring element. Thus labels are always updated.

3. REDUCING THE DATA STRUCTURE TO A SCREEN DISPLAY

The process of scanning the data structure to retrieve and display the desired text has four basic parts: (1) the sequence generator (as discussed briefly in Section 1.6.3), (2) filtering, (3) formatting, and (4) display.

3.1 SEQUENCE GENERATOR

The sequence generator is the routine that actually scans the data structure and generates the sequential text. Basically it generates a list of statements. There are three types of sequences that can be generated:

3.1.1 Tree.

This is the default hierarchical structure that is generated and is simply the sequential text of the main associational trail of the text, ordered in a hierarchy of statements.

3.1.2 Trails

The trail feature is used to set up statements in such a way that only a particular set of statements will be displayed and in a particular order. The set of statements is called a trail, and is an associational trail that criss-crosses the default (main) trail; it provides a manner other than the normal sequence in which to read the text. A trail marker is set up for a particular trail of statements; the pattern for this marker can be a complex syntactical form and is followed by the content analyzer (described in an SRI publication).

Trail markers are thus used to mark turning points from the normal sequence of statements, as a signpost to the next statement in the trail. Each time a marker appears in a statements it is followed by a statement name in parentheses that is the name of the next statement. Between trail markers

statements are displayed in normal sequence. The trails can be followed only in the forward direction; there is no capability for inverting the trail when moving backwards through the text. (SRI claims that with the complex content-analyzer, this is unnecessary.)

3.1.3 Keywords

The keyword system permits a user to construct a specially formatted catalog file containing references to other files and capable of being reordered automatically according to some chosen set of weighted keywords. When reordered, the file lists references in order of relevance, according to the choice and weighting of keywords.

The keywords are attached to a statement. The system keeps a list of the keywords containing for each keyword a short description of the keyword, and the labels of statements tagged with this keyword. This list is visible to the user and can be changed by him. The system also keeps a list of the file-reference entries, that is, a file of any statement name tagged with a keyword, and a list of the keywords it is tagged with following it. Thus one keyword can be attached to any number of statements, and one statement may have any number of keywords attached to it.

The keyword system is used mainly as a retrieval-by-keyword system. The user selects desired keywords and weights them according to importance. A negative weight can also be used to blackball any keywords. According to SRI,the weights on the keywords allow more flexibility than straight Boolean retrieval functions on keywords; after the user has selected keywords and weights, the system goes to the list of keywords and picks out all statements tagged with the selected keywords. For each statement selected the system computes the weights of keywords attached to it, and displays the names of the statements in order of highest total weight. Statements with a negative total weight are not displayed. The user may then access the referenced files by using the jump command on the statement names.

3.2 FILTERING

Facilities included are: level specification, branch only, subfile, content analyzer, trail flags, literals, search for trail flags and literal text, etc.

After the main structure is generated and filtered, it is formatted.

3.3 FORMATTING

The formatting sets the following, and other, variables of display:

Statements numbers: the number of lines of each statement to be displayed is variable; headers, time/initials/labels can be on/off.

View change: character size, page size and dimensions, etc.

3.4 DISPLAY

After the statements have been filtered out, they are displayed. The main display of the generated/filtered/formatted structure is in the file area of the screen. There are a number of one-dimensional registers used for man/machine interaction:

1. Echo register. This displays the last six characters typed by the user, for feedback.

2. Command display line. This is a line which says what command is in the process of being executed.

3. Name register. Displays user's name (this is on a multi-terminal system).

4. View specification areas. There are three view spec areas, and these are set according to the formatting variables described in Sec. 4.28.

5. Message area. An area for system messages to the user, such as error messages.

6. Literal type-in area. When the user is typing in an insert or delineating a command, the characters typed are displayed in this area.

There is, in addition to the file area, another two-dimensional area, the freeze area. This freeze area is used to "freeze" statements designated by the user so that they remain

unchanged above the file area, with the file being then displayed in the file area. The freezed statements remain unchanged despite any text manipulations or file searching that goes on in the file area. (In a future version, the freeze area will be done away with, and instead the user will be able to multi-window any number of windows. Each window will be a full file area, with all one-dimensional registers in each window. They can be any shape or size any place on the screen. With multistations, a window can be assigned to a station, giving the users at two different terminals the ability to decide who holds the chalk and who holds the eraser in each window.)

4. SYSTEM FEATURES AND FACILITIES

4.1 EDITING

The basic editing commands are delete, insert, replace, move, copy, set, and break/join. All are self explanatory, except set and break/join. The set commands allow the user to change the font on any text string. The fonts are: capital, lower case, italic, roman, boldface, no boldface, flickering, non-flickering, underline, no underline. The break and join commands allow the user to break a statement into two statements; the join command adds a text string onto another statement. The break and join commands are the only commands that operate across statement boundaries. All the other editing commands are specialized: for example, the insert commands are insert character, insert word, insert text, insert invisible, insert statement, insert branch. The specialized commands make it easier for the system to make the edits; the rationale for specialization is that since you have to type the command in you may as well specialize, and economies in data structure manipulation may be achieved (e.g., moving an entire branch of the tree).

4.2 OTHER FEATURES

4.2.1 Invisibles

When editing, invisibles such as spaces and tabs can be displayed by marks, and thus can be deleted.

4.2.2 Labels

Labels are statement names and are used for retrieval purposes by jumps, links, and keywords. They are inserted as part of the text, that is, with an insert command. A label is simply a variable-length character string that appears at the beginning of a statement in parentheses. These labels can be changed or deleted as if they were regular text.

5B2B Duplicate labels can be created. A jump to a label results in a jump to the first occurrence of that label, since the system sequentially scans the name-hash field of the ring elements. A feature contemplated for incorporation in the system is a "look for next occurrence of this label" jump to resolve duplicate labels.

5B3 ## 4.2.3 Links

5B37 A link is an association to another statement, i.e., it is a jump to another statement that can be taken at the option of the user. The link can be in the current file or in another file. There are four parameters to a link: three (the user name, filename, and label) define the point linked to. The fourth is the view specifications on the text linked to. This is an interesting feature: that view specifications can be changed on all links.

5B3B The link structure is a regular text string inserted in the text as if part of it, and is in parentheses in a syntactic form. Like labels, the link is just regular text until it is used. It can be edited at any time. When the user decides to take a link, he hits a character with the bug. The system scans forward with the content-analyzer until it picks up the nearest link structure in that statement, and jumps to the label. The link is taken by use of a jump command.

5B4 ## 4.2.4 Intrafile Return Ring

5B4A Whenever any jump is made within the file, a new entry is made in a list called the intrafile ring. Each of these entries gives a display start and a set of viewspecs. A pointer indicates the current view on the list. Each time a jump is executed, the new information is written ahead of the pointer and the pointer is moved forward. On a jump return or jump ahead, the pointer is simply moved backward or forward and no new entries are made or any deleted. The list holds a maximum of six entries, and is circular.

4.2.5 Interfile Return Stack

This works much like the intrafile stack except that it is concerned with jumps between files. The differences with the intrafile stack are: (1) the length of the list is variable, and depends on the amount of information in the links used, (2) the list is not circular, a new entry is made on the stack whenever any interfile jump is taken or whenever a new file is loaded with a load file command. (See section on multifiles for more details.)

There are no backpointers from a link, the same as with trail markers. Thus if a label that is linked to is deleted, there is no user notification that a link has been made inoperable. Also, since link structures are entered as simple text, the label in a link structure does not necessarily exist. A link or jump to a non-existent label results in an error condition.

4.2.6 Jump

The jump command brings the desired statement to the top of the display.

There are four basic types of jumps: (1) jumps to a specified label name, (2) jumps to links, (3) jumps through the tree structure, and (4) jumps among different files.

In case 1, the label or statement name to be jumped to can be specified by either a word-selection via the mouse or a literal entry from the keyboard.

In case 2, the statement defined by the specified link is placed at the top of the display. More detail is given in section 4.2.3.

The case 3 commands allow jumps to the next substatement, the next successor, the statement of which the selected statement is a substatement, the previous statement, the head of the file, the end of the file, the end of branches, and many other links on the basis of tree and file structure. For more details see the "NLS User's Guide."

The case 4 commands allow the user to load a number of files into the system and to jump freely among them. These will be discussed in Section 4.2.9.

There is one other type of jump, the jump-ahead/return. Whenever any type of jump within the current file is executed, the sytem keeps track of it, and a ring is maintained keeping a sequential track of all views that have been used. These commands allow the user to return to a previous view or to move forward after a jump return to the latest view. (See Section 4.2.3 on links for a description of this intrafile ring.)

A special feature of jumps is that almost all jumps allow the user to change the view specifications of the area jumped to from those of the current text. In addition, each jump saves the viewspecs of the area jumped from in the intrafile ring, so that on a jump return the text is viewed as before.

4.2.7 Pointers

Pointers make it possible to select entities that are not on the display. The entity has a pointer fixed on it while it is on the screen of not more than three characters. To select the entity at any time, a mouse button is depressed and the name of the pointer is entered from the keyboard. This is exactly equivalent to making a direct bug selection of the character that has the pointer on it.

The list of pointers can be displayed and one may use it to jump to the individual pointers.

4.2.8 View Specifications

The view specifications (viewspecs) are parameters that control the way in which statements are displayed. The parameters are: indenting on/off; names on/off; display file as tree/normal text; keyword reordering on/off; display of statement signatures on/off; branch-only on/off; content analyzer on/off; trail feature on/off; pointer display on/off; number of lines displayed; number of levels of statements displayed and a few others. These can be set in three ways: with the view set command, from the special keyset, or during certain commands such as jump.

These parameters are always displayed in the upper left corner of the screen with a single letter denoting each. When they are capable of being changed by the user, they are displayed with larger letters.

There is a relative level control, which allows changes to the level parameter set by the user to be interpreted relative to the level of the first statement in the display. The user can also change other viewing parameters. These include the type of mark the cursor leaves, the number of characters in a line of text, the number of spaces indented for each level, the number of lines in the text area, the spacing between lines, size of characters, etc.

4.2.9 Multi-files

When a file is loaded or jumped to, it is "opened" and displayed; no copy is created, rather the file is viewed directly from the disk. For reasons of file protection, if any changes are made, it becomes impossible to continue direct viewing, so the system creates a working copy when an edit is made. In fact, this working copy is not created until all core is filled and not necessarily on the first edit. In this way the system does not make a working copy until it definitely has to. When the system creates the working copy it copies the displayed file to it, closes the displayed file, and from then on all work is done in the working copy. No working copy is created when the user is just browsing. This is done since most users just look at files and do no editing.

Files are loaded by the load command or by an interfile jump command. Entries are made in the interfile stack as files are loaded (see Section 4.2.5). The working copy and the checkpoint file are never entered in the stack.

One feature of the multi-files is that the user can create a checkpoint file at any time. This writes the present working copy out on the drum under the name checkpoint.

The interfile stack can be used like the intrafile stack to go back and forth among views on different files. Only one working copy at a time can be created, and can be looked at any time, even if a file other than the one of which a working copy was made has been currently loaded.

4.2.10 Freeze

The freeze feature freezes a single statement with the present view. The frozen statement will appear at the top of the screen

whenever frozen statements are being shown, with the main text display on the under part of the screen. A fixed number of statements can be frozen, and are displayed in the freeze area in the chronological order frozen.

5B11 4.2.11 Tree-display Feature

5B11A This allows the user to see the file as a tree structure, or in the hierarchy form, instead of normal text. The tree structure shows the relationships of statements in the file to each other. This is done by indenting the differing levels of the tree to different depths, much like an outline form. This can be turned on or off by the view specifications.

5B12 4.2.12 Statement Numbering

5B12A The system numbers each statement Dewey Decimal fashion according to the tree structure. This numbering is computed at display time. The numbering can be turned off by the view specifications.

5B13 4.2.13 Vectors

5B13A The vector package allows the user to create simple line drawings, with labels for jumps. The vector is drawn by specifying the endpoints with the mouse. Either endpoint of a line can be translated, and the entire drawing and any label can be translated. These vector labels can be used as jumps to that statement name.

5. FUTURE FEATURES

5.1 MULTIWINDOWS

This may have been inspired by our multiwindows. Theirs, however, is fancier in conception. This would allow any size and shape windows to be defined, and each window to be a self-contained viewing area with all the parameters as described for the single screen display. Their multiwindow facility could also assign different windows to different users. This assignment is done by the time sharing system, though; the only programming problem is the protocol: who holds the eraser in each window.

5.2 VARIABLE SYMBOLS

This would allow the user to define a variable symbol for text, links, etc. The symbol would be filled in with text at display time, like an assembly time variable. Alternately, the variable symbol could simply be permanently defined at a later time.

5.3 WEIERSTRASS ALGORITHM

Currently the system uses a display map technique for detecting bug hits. A future plan is to use the Weierstrass Algorithm of continually subdividing the screen to find the line closest to the bug mark, which would be the line hit.

6. A FEW IMPRESSIONS

The hierarchical structure allows the text to be set out in a tree form very easily. The question of advantage of this over traditional text was discussed with Engelbart. He said that the hierarchical statement-oriented structure was selected just as a starting point and empirically has proven to be more helpful to users in terms of visualizing the text. He insisted there is no premeditated reason toward this structure, nor need it be imposed on the user.

The statement oriented quality limits the flexibility of editing somewhat. From our point of view, there is no editing across statement boundaries, for instance. Jeff said that this limitation is of no real importance since as users gain familiarity with the statement oriented system, they learn to make statements complete thoughts, and so editing across statement boundaries is not really necessary; the limitation is only on traditional thinking with traditional text. This is the same reason Engelbart stated for using hierarchy: the user quickly adapts to the structure provided him.

One advantage of the statement oriented structure is that to move a branch or a statement requires no actual movement of text, but just the changing of a few pointers.

There is great effort not to let the user hurt himself when he cannot see the entire tree structure due to filters. For example, a user cannot delete an entire statement. There might be substatements below that are filtered out that he might inadvertently delete: he must give a delete-branch command and delete the entire branch.

AHI DATA STRUCTURE

A.M. 276 CLASS NOTES

Compliments of the

HYPERTEXT EDITING SYSTEM

CENTER FOR
COMPUTER & INFORMATION SCIENCES
BROWN UNIVERSITY
PROVIDENCE, RHODE ISLAND

30 March, 1969