MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS

DOCUMENT RESUME

ED 168 462   IR 006 856

TITLE   Individualized Instruction for Data Access (IIDA).
                   Quarterly Report No. 1, June, 1978.
INSTITUTION   Drexel Univ., Philadelphia, Pa. Graduate School of
                   Library Science.; Franklin Inst. Research Labs.,
                   Philadelphia, Pa.
SPONS AGENCY   National Science Foundation, Washington, D.C.
PUB DATE   Jun 78
GRANT   NSF-DSI-77-26524
NOTE   83p.; For related document, see ED 145 826

EDRS PRICE   MF01/PC04 Plus Postage.
DESCRIPTORS   *Computer Assisted Instruction; *Computer Programs;
                   Computers; Data Bases; *Information Retrieval;
                   Information Systems; Models; *Program Design;
                   Relevance (Information Retrieval); *Search
                   Strategies; Systems Analysis
IDENTIFIERS   *Individualized Instruction for Data Access

ABSTRACT
                   This report provides a brief general summary of
progress in a 2-year project renewing earlier work on the development
of Individualized Instruction for Data Access (IIDA)--a system for
teaching online searching techniques for bibliographic data bases--as
well as an in-depth report on the IIDA computer system design. The
Connector for Networked Information Transfer (CONIT), from which the
IIDA project has borrowed software, is briefly reviewed and the
specific CONIT software used by IIDA is examined. The overall flow
and a detailed description of the IIDA software are discussed. The
report also examines the data structures, including specific contents
and routines, and diagnostic programs, used to analyze the search in
progress. Exercise and assistance mode, which provide the student
instruction in the use of DIALOG through exercises controlled to
varying degrees by the computer, and plans for computer program
production are outlined. Appendices include context strings in IIDA
and a list of IIDA special features. (CWM)

****
*   Reproductions supplied by EDRS are the best that can be made   *
*   from the original document.   *
****

ED16 ...

INDIVIDUALIZED INSTRUCTION FOR DATA ACCESS (IIDA)

Quarterly Report No. 1
June, 1978

Drexel University, School of Library
and Information Science
Franklin Institute Research Laboratories

NSF Grant No. DSI 77-26524

2

TABLE OF CONTENTS

1. GENERAL SUMMARY OF PROGRESS

This project represents a renewal of earlier work on Individualized Instruction for Data Access (IIDA). Begun in July 1976, with initial funding for one year, the project was resumed in April 1978 and is to be completed in two years. This series of quarterly reports is planned to report in depth on selected aspects of the project and to contain a brief, overall progress statement in each report.

With the renewal of the project in 1978 came a change in its goals. Initially, we intended to produce a fully operational service, i.e. not only a functioning system but a system made available to a wide body of users on an eventually self-supporting basis. In view of the reduction of emphasis on user training by NSF's Division of Information Science and Technology, we are now restricting ourselves to proving the concept with emphasis on what can be learned from it. Functionally, this means a reduced emphasis on computer programming and on planning for marketing the system. There is no change in the basic logic of the system. For experimental purposes, it is not necessary to be able to serve more than a single user at a time, and this considerably reduces the computer programming effort of the project.

The project staff are divided into two groups. The behavioral group is concerned with studying the search process, developing a model of the process, and developing and carrying out plans for testing and evaluation of IIDA. The computer group is concerned with the design, implementation and testing of the requisite computer programs. The bulk of this report is concentrated on program design.

We are pleased to have available to us an oversight committee, composed of distinguished persons active in the design, use, or research into interactive information systems. A first meeting was held on June 16, 1978 and additional meetings are planned at approximately six-month intervals thereafter. Committee members are:

Professor Frances J. Reintjes, Massachusetts Institute of Technology
Ms. Barbara Lawrence, Exxon Research and Engineering Company
Professor Jerry S. Kidd, University of Maryland
Mr. Mario C. Grignetti, Bolt, Beranek and Newman

Project members are:

Professor Charles T. Meadow, Principal Investigator, School of Library and Information Science, Drexel University
Professor Thomas T. Hewett, Department of Psychology and Sociology, Drexel University
Mr. David E. Toliver, Franklin Institute Research Laboratories
Ms. Janet V. Edelmann, Franklin Institute Research Laboratories
Ms. Sari Scott, SLIS, Drexel University
Mr. Jerry Warren, SLIS, Drexel University
Ms. Carol Fenichel, SLIS, Drexel University

Assisting as a consultant on evaluation is Dr. Robert Rich, Princeton University.

II. IIDA COMPUTER SYSTEM DESIGN

1. Introduction

The IIDA computer system has been previously described as consisting of three processors: the instructional processor, the communications processor, and the data base processor.[1]

The instructional processor is a set of programs which mediate the data base search. It both controls the sequence in which commands may be issued and executes diagnostic programs which analyze both particular commands and the overall search strategy. New software written for the IIDA project will be exclusively for the instructional processor.

The communications processor is a set of programs which perform all functions related to the handling of messages between users at terminals and the instructional processor, and between the instructional processor and the data base processor. The IIDA project has borrowed software developed by the Connector for Networked Information Transfer (CONIT) project[2] at M.I.T. to perform all communications processor functions. CONIT also provides the framework for a rule-driven interpreter -- a software construct that will be used extensively by IIDA.
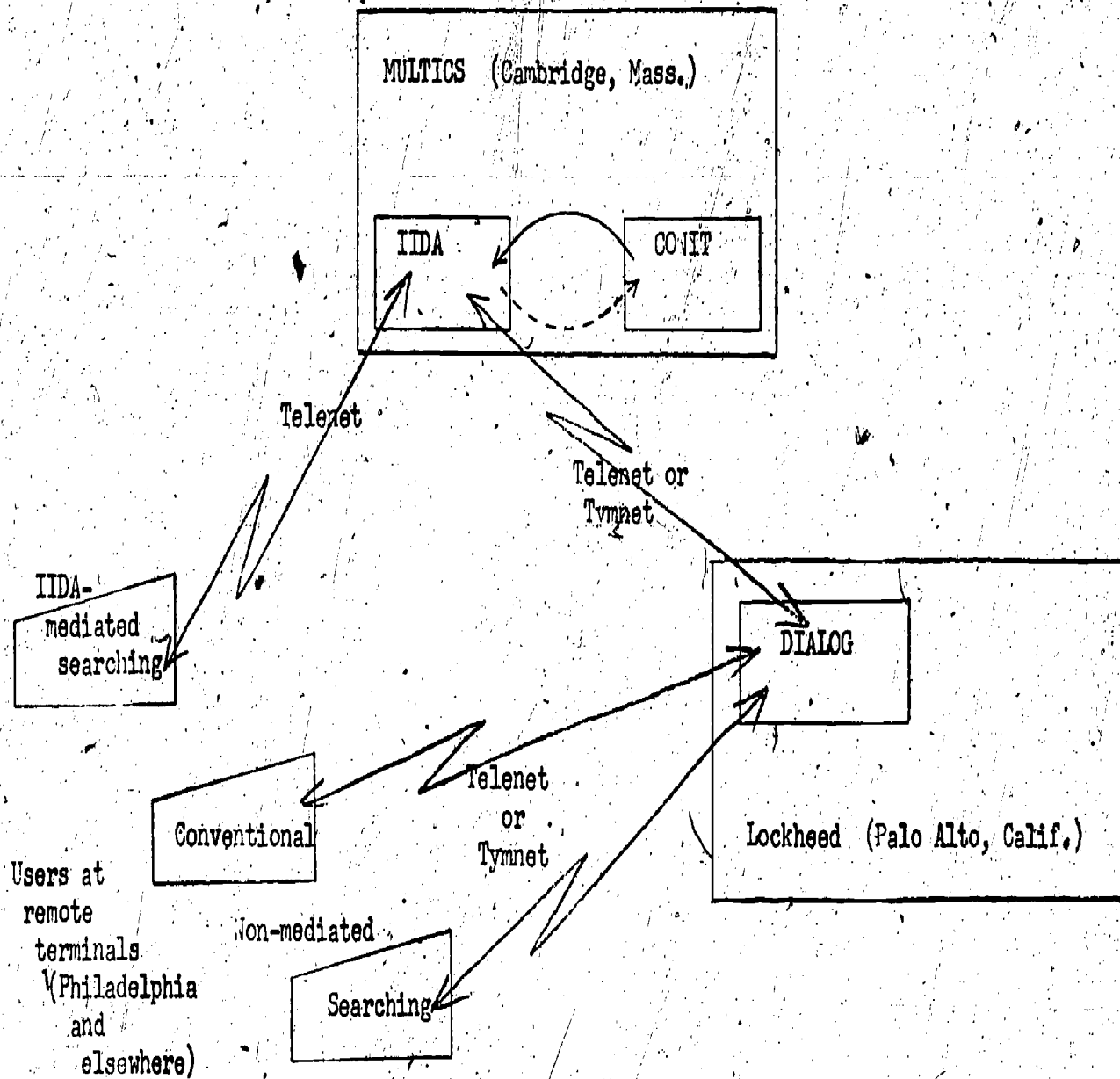
The data base processor performs information retrieval functions and has access to bibliographic data bases. There will be a single data base processor for the IIDA project, the DIALOG search service operated by Lockheed Information Systems. IIDA will use the DIALOG system without modification. To DIALOG, the computer in which IIDA resides will appear as just another terminal.

Development of IIDA software will be done on M.I.T.'s Honeywell 6180 computer, running under the MULTICS operating system. This computer was chosen largely because CONIT was developed on it and resides in it now. Some changes were made to MULTICS to accommodate CONIT's telecommunications requirements. IIDA will benefit from these changes and will not require similar system programming on another computer. Also, software interchange between the two projects will be achieved by issuing a few monitor commands, rather than by physically transporting tapes of CONIT software to another IIDA site. Finally, a dial-out peripheral was installed on MULTICS for the CONIT project. In its early development, IIDA will be able to share this unit with CONIT.

---

[1]Meadow, Charles T., et al, Individualized Instruction for Data Access (IIDA): Final Design Report, NSF Grant No. DSI-76-09737, Philadelphia: Drexel University, Graduate School of Library Science, July, 1977, p. 31.

[2]Marcus, R. S., and Reintjes, J. F., Computer Interfaces for User Access to Heterogeneous Information Retrieval Systems, M.I.T. Electronic Systems Laboratory Report ESL-R-739, NSF Grant No. SIS-75-22946, Cambridge, Mass.: Massachusetts Institute of Technology, April 1977.

MULTICS (Cambridge, Mass.)

IIDA ⟲ CONIT

Telenet

Telenet or
Tymnet

IIDA-
mediated
searching

Conventional

Telenet
or
Tymnet

DIALOG

Lockheed (Palo Alto, Calif.)

Users at
remote
terminals
(Philadelphia
and
elsewhere)

Non-mediated

Searching

Figure 1.1.   The system can be considered as consisting of three processors: Instructional,
Communications, and Data Base.   IIDA software makes up the Instructional Processor.   Software borrowed
from CONIT and the MULTICS Auto-Call device make up the Communications Processor.   DIALOG software
makes up the Data Base Processor.   MULTICS is the operating system of M.I.T.'s Honeywell 6180 computer,
the intermediate computer in which both CONIT and IIDA are located.

Refer to Figure 1.1 for an overview of the entire IIDA configuration. Note that users will be able to access IIDA in MULTICS through the Telenet communications network; IIDA and CONIT will be able to interchange software; and IIDA will be able to access DIALOG through either the Telenet or Tymnet communications networks, as CONIT does presently.

The remainder of this chapter will discuss: an overview of the CONIT system; what IIDA will be using of the CONIT software; the overall flow and a detailed description of the IIDA software; the data-structures used by IIDA; and a detailed description of the diagnostic programs.

8

2. Overview of the CONIT system

CONIT's overall configuration is much like IIDA's as represented in Figure 1.1. While IIDA is tied into DIALOG alone, though, while CONIT can serially access several search systems in a single session. The primary purpose of CONIT is to allow searchers of on-line bibliographic systems to access several such systems using a single command language. After users log into CONIT, they are allowed to select any data base available through any of the participating vendors. These vendors include: Lockheed (DIALOG), SDC (ORBIT), NLM (ELHILL), and the SUNY Biomedical Network.

The user enters search commands specific to CONIT. These commands are translated into commands with the same function in the language of the vendor's search system. Responses from the search system are usually translated back into formats specific to the CONIT language. Thus, "one-stop-shopping" in a large number of bibliographic data bases is possible. In order to make itself useful to relatively naive users, CONIT provides extensive explanations of the use of its commands, the data bases available, how to correct errors, etc.

The most significant features of CONIT for the IIDA project are: automatic dial-out capability, handling of communication to-and-from users and to-and-from search systems, and a rule-driven interpreter for parsing, translation and syntax verification.

The dial-out capability involves both software and hardware. The hardware peripheral is an "Auto-Call" device which allows MULTICS "under program control, to call up another computer over regular switched-telephone-line circuits."[1] Some MULTICS operating system software had to be modified in order to adopt this device for the CONIT application. Finally, CONIT software contains procedures which bring the dial-out capability directly under its control.

Communication between the CONIT system and its users and between the CONIT system and the data base search systems involves calls to MULTICS system subroutines from CONIT procedures. These procedures prepare the input streams for processing. In their turn, these procedures are called by special-actions (subroutines) invoked by rules in the rule-table and by CONIT'S control procedure.

The rule-table is a list structure of single-line rules, each of which is made up of as many as seven parts. These are delimited from each other by special characters. These seven parts are:

1. Context-string (CS): a character string that specifies the context or situation;
2. Match-string (MS): a character string to be matched in the designated input stream;

---

[1]Marcus, p. 9.

3. Host-message (HM): message to be sent to the host system;
4. User-message (UM): message to be sent to user;
5. New-context (NC): revised context put into effect after rule execution;
6. Special-action (SA): particular function to be performed;
7. Comment (CM): has no effect on rule execution.[2]

The syntax of a rule, with its delimiting special-characters, is:

/<CS>/<MS>:.::<HM>::,,<UM>,,//<NC>//..<SA>.. **<CM>

Control of the entire search session at the most general level can be per-
ceived as residing in the rule-table. At all times a pointer identifies a
rule whose context-string matches at least the first few characters of the
context-string describing the current search-status. Another pointer indexes
the input stream from both the user and the retrieval system in turn. When-
ever the context-string changes, the rule table is searched for a match of
both the longest most explicit context-string and the longest initial sub-
string of characters beginning at the pointer to the input-stream.

When such a match is found in a rule, the rule is executed. This involves
one or more of the following: (1) appending the string found in the host-
message part of the rule to the host-message output-stream; (2) appending the
string found in the user-message part of the rule to the user-message output-
stream; (3) setting the current context of the string to the new-context part
of the rule; and (4) performing the special-action (PL/1 subroutine) specified
in the special-action part of the rule.

The pointer to the input-stream is repositioned if the match-string was
not empty. The special-action may involve the isolation of a lexeme, sending
a message, building a message, setting communications parameters, and other-
wise controlling the session.

An example of eight related rules follow, together with an explanation of
each rule. These rules are written for IIDA and are not to be found in
CONIT. The rules of this example are selected from among those in which IIDA
help is requested by and given to the user. The user can call for such help
with the command: /HELP. A more complete discussion of the processing of
this command can be found in Section 4.3.

   (1)   /u/..-/H:.//h::::::9//..timerec..

This rule applies after the user has entered a command beginning with
"/H". The code "u" in the context-string indicates input from the user. (See
Appendix A for a description of codes in context-strings.) All input streams
are prefixed by "..-" and suffixed by "-.." by the CONIT control software.
The match-string here thus indicates an initial "/H". The new-context is
changed to begin with an "h" indicating the /HELP section of

---

[2]Marcus, p. 13.

rules. The previous connect status (positions 2-4 of the context-string),
network indicator (position 5), and search mode indicator (positions 6-7) are
all preserved by the universal replacement code, the colon. A protocol
sequence number "9" is set. The special-action "timerec" records in the help
history data structure the time at which the user requested help. This is one
of six data structures maintained by IIDA on student commands, search status,
and progress as a whole.

   (2) /h------9/:.,,37,,//h::::::89//..senmes..

Because of the new context set above, this rule will be matched next. The
dash symbol in the context-string is the universal match code -- any code will
match this position. In this rule, the match string is empty -- also a
universal match condition when there is no explicit match. The user-message
string contains a code indexing a message in a table of messages. This code
is used by the special-action "senmes" to send that indexed message to the
user at his terminal. In this case, the paragraph consists of a menu of help
alternatives. Only the protocol sequence number "89" is changed by the new
context.

   (3) /h------89/:.//h::::::88//..send..

"Send" is a special-action which means either "send-a-line" or "end-a-
sentence." It has the effect of next seeking a response from the user in
either case.

   (4) /h------88/:.//h::::::87//..hupdate..

Because of the protocol sequence number and the empty match string, this rule
will be executed next. The only context change made by the new context is a
decremented protocol sequence number. The special-action "hupdate" will up-
date the help history data structures with the user's response to the last
menu.

   (5) /h------87/..-3:.//h::::::7//..help3..

This rule will be executed if the user selected the third option from the last
menu by entering the number "3" because the input-stream matches the match-
string. The protocol sequence number is decremented by the new context. The
special-action "help3" will format the history table requested as option 3 and
send it to the user.

   (6) /h------7/:.,,55,,//h:::::6//..senmes..

Like rule (2) above, a message indexed by 55 in the message-table is sent to
the user. This message indicates that further help is available. The user is
asked to make a selection from any of the menus he has already seen.

   (7) /h------6/..//h::::::5//..send..

Like rule (3) above, a response to the last menu is sought from the user.

(8) ,/h-------5/..-21:.//f+//..htime..

Suppose that "21" is the code that indicates that the user is finished with
assistance from IIDA. When this is found in the input-stream, the first
character of the context is changed to "f" to indicate that the front-end
instruction section of the rules is to be executed next. The symbol "+" saves
all codes from the old context beginning in the position of the "+". The
special-action "htime" records the exit time in the help data structure and
tests the threshold to see if the user has spent an inordinate amount of time
for his skill-level getting help from IIDA.

3. IIDA's Use of CONIT Software

In some respects CONIT provides more facilities than IIDA requires. CONIT emphasizes translation of its own commands into those of the several vendors, making all data bases available. IIDA, though, will begin teaching a single search system, DIALOG, and will restrict itself to one or two data bases. Valid commands in the DIALOG language will be transmitted verbatim from user to DIALOG.

In other respects, IIDA requires facilities not provided by CONIT. IIDA will maintain extensive d▓ ▓▓▓▓ student searches in elaborate file and data structures. Furthermore ▓▓▓▓▓▓▓ ly allows commands in the exercise modes so that the com▓▓ ▓▓▓▓ ed gradually and in a meaning▓▓ sequence. CONIT doe▓ ▓▓▓▓▓▓▓▓ either of these requiremen▓

The two systems still have much in common; both are search-mediating programs on an intermediate computer. IIDA will build upon most of the CONIT software in one way or another. Software that handles communications will be borrowed intact and without change. These procedures include those which enable dialing out to the network, logging into DIALOG, receiving messages from and sending messages to both the search system and the user, and handling timeouts and drops by the host computer. Also kept will be the "system-software" for rule-maintenance-and-processing.

Other software will be borrowed from CONIT by IIDA as far as concept and structure are concerned, although the detailed content will be completely rewritten. For instance, CONIT gives significant amounts of on-line assistance outside the context of any particular search. IIDA assistance, however, will largely be within the context of a particular search. Much of the text of CONIT assistance is stored in tables that are called by special-actions. The content of these tables will be completely rewritten to reflect both the DIALOG search language and the IIDA approach to assistance.

Also in this class of borrowed software are the rules in the rule-table and many of the special-actions. Although IIDA will use the same general form of the rules, most of the rules will be rewritten explicitly for the IIDA application. CONIT rules which deal with logging into DIALOG and communications will be maintained intact, however. Revision of the rule table will be taken as far as changing many of the codes and positional significance of the context strings in the rules. See Appendix A for a list of both the new and borrowed codes in the context strings.

Some CONIT software can be dropped altogether for the purposes of IIDA. Procedures developed to inform the user about the entire set of data bases available throughout all the systems can be dropped since users of IIDA will be able to access only one or two data bases on the DIALOG system. All CONIT rules and special-actions which deal specifically with other systems can be dropped, including those which enable logging into other systems. Since IIDA will not be translating from one language to another, the purpose of the rules will shift from translation to control and identification. As a result, the host- and user-message parts of the rules will not be used as extensively by IIDA as they were by CONIT.

Finally, much new software must be written for the IIDA application. This includes all diagnostic and other search analysis subroutines, the command and response parsers, the IIDA data structure (search history) updates, and much of the control sequence as managed by the rules.

14

4. Overall Flow of IIDA Software

Refer to Figure 4.1 for a graphic representation of the flow of IIDA processing from a functional point-of-view. The various functions are performed by control modules, rules, and special-actions. Control modules consist largely of unchanged CONIT code; rules will largely be written for the IIDA application; special-actions will be a mixture of CONIT and IIDA code. See Figure 4.2 for a list of functions, code sources, and types of procedures. Both Figures 4.1 and 4.2 are keyed to the numbers of subsequent subsections of this report which discuss the functions and procedures in detail. A summary of this discussion follows, with subsection numbers in parentheses. Again, these numbers correlate with those in Figure 4.1.

(4.1) Logging into MULTICS and calling for the IIDA-system involves conventional protocols to be spelled out for the user. A greeting will elicit from the user some ∎∎∎ial information to help guide the course of IIDA's analysis and ∎∎ ∎ user will choose one of four modes for his search. These roughly ∎∎∎ to the user's level of skill in searching. Dialing the network a∎ ∎ nto DIALOG is provided by existing CONIT software.

(4.2) CONIT software handles getting the command from the user. The user will be cued with 'D?' for commands directed to DIALOG and with 'I?' for responses directed to IIDA.

(4.3) If the user enters "/HELP", assistance from IIDA will follow in the form of menus from which the user may select the type of assistance he needs. Help consists of explanation of commands and reviews of the search from several different perspectives. The latter involves formatting the search history data structures.

(4.4) If the user doesn't enter "/HELP", the command is assumed to be directed to DIALOG. The verb of the command is first identified. If the verb is not a DIALOG command, this information is recorded. The user is told what commands are acceptable at this stage in his chosen mode. Control transfers back for the input of another user command.

(4.5) If the verb is a DIALOG command, rules determine if the verb is acceptable for the current search mode at this point in the search. If not acceptable, a message is issued to the user to this effect and control transfers back for the next user input.

(4.6) If the verb is acceptable for the current search mode, the entire argument of the search command is parsed by special-actions for validation and isolation of its lexemes. If the syntax is invalid, a message is issued to this effect. IIDA will be very specific as to what exactly is wrong with the argument syntax. If the syntax is correct, lexemes are stored in the various IIDA data structures and the command is relayed to DIALOG. Again, the communications software is provided by CONIT.

(4.7) Between sending a command to DIALOG and receiving a response, significant delays in term of computer processing time can be expected. What are essentially "background jobs" are executed during these delays. These jobs (procedures) perform micro-analyses on the user's search strategy. CONIT software is used to retrieve the DIALOG response from an input buffer.

START

4.1 Log into MULTICS/IIDA via Telenet

4.1 Greet user; initialize mode; DIA

4.2 Get command from user

4.3 Is Cmd=/HELP ? — Yes → **4.3** Give user /HELP menus that he requests — No ↓

4.4 Identify verb of command

4.4 Is verb OK ? — No → **4.4** Update data structures with error indication — Yes ↓

B

4.13 Mode-specific instruction

A

4.5 Message that command in error for this mode

4.6 Message that command syntax is in error

B

4.5 O.K. for mode ? — No → / Yes ↓

4.6 Parse full argument of DIALOG command

4.6 Is syntax OK ? — No / Yes ↓

4.6 Update data structures from parse

4.6 Transmit command to DIALOG

C

Figure 4.1. Overall flow of IIDA programs. Numbers in the upper right hand corner of the decision and action boxes refer to the section of this report in which the decision or action is discussed.

C

4.7 Delay-time micro-analysis processing

4.7 Get DIALOG/ Telenet response

4.8 DIALOG error msg ? — Yes → **4.8** Notify user; Update data structure

No

4.8 Other system err. ? — Yes → **4.8** Several pos. responses to system problem.

No

4.9 Parse DIALOG response

D

4.9 Update data structures from response

4.9 Transmit response to user

4.10 Evaluate diagnostic thresholds

4.10 Pass thresholds ? — Yes

No

4.11 Detailed analysis and discussion of problem.

4.12 Is cmd=END ? — Yes → **4.12** End of search processing

No

A

STOP

Figure 4.1. (cont.) Overall flow of IIDA programs.

17

Discussed in Section	Function	Code Source	Type of Procedure
4.1	Log into MULTICS, IIDA & DIALOG	CONIT	Control & rules
4.2	Get input from user	CONIT	Control
4.3	Give IIDA /HELP assistance	IIDA	Rules & s.a.
4.4	Identify basic type of command	IIDA	Rules
4.5	Evaluate validity of mode	IIDA	Rules
4.5	Messages for invalid mode	IIDA	Rules & s.a.
4.6	Parse argument of DIALOG commands	IIDA	Special actions
4.6	Update data structures	IIDA	Special actions
4.6	Messages for invalid syntax	IIDA	Rules & s.a.
4.6	Transmit command to DIALOG	CONIT	Special action
4.7	Delay-time processing	IIDA	S.a. & control
4.7	Get response from DIALOG	CONIT	Control
4.8	Process DIALOG error message	IIDA	Rules & s.a.
4.8	Process other system errors	CONIT	Rules & s.a.
4.9	Parse DIALOG response	IIDA	Special actions
4.9	Update data structures	IIDA	Special actions
4.9	Transmit response to user	CONIT	Special action
4.10	Evaluate diagnostic thresholds	IIDA	Special actions
4.11	Detailed analysis & discussion	IIDA	Special actions
4.12	Handling the "END" command	IIDA	Rules & s.a.
4.13	Mode-oriented instruction	IIDA	Rules & s.a.

Figure 4.2. Function, code source, and type of procedure in IIDA software. This chart corresponds to the flowchart of Figure 4.1. "S.a." means "special actions."

(4.8) If the response is an error message from DIALOG, this is analyzed and transmitted to the user. Other system problems may develop such as being dropped by the host. CONIT software generally handles these problems. Several responses may be in order including just a return to the user for his next input.

(4.9) problems were encountered, the response from DIALOG is parsed and its lexemes are stored in the various data structures. The response is transmitted verbatim to the user.

(4.10) Internally stored and generated data on the search strategy is evaluated to determine if strategic problems are developing. These are basically formulae which calculate if the current search is within bounds experimentally derived that represent the range of a well-formed search.

(4.11) If strategic problems are developing, they will be brought to the user's attention and an interactive discussion may follow. In the course of this discussion further analyses are applied to the search strategy, again are it to parameters developed to represent a well-formed search. The results of these analyses will be used to guide IIDA's discussion of the search strategy with the user.

(4.12) In any case, if the DIALOG command "END" was entered, the session will terminate with wrap-up routines such as saving the data structures on an external file.

(4.13) If the command was not "END", control will pass to the front-end, giving the user instruction specific to his search mode. If it is one of the exercise modes, this instruction will tell him which commands are allowed at this stage of his search.

4.1 Logging into MULTICS, IIDA, and DIALOG

The user will know or be assisted with the procedure for logging into IIDA under the MULTICS system. This will be done for the duration of the project through the Telenet communications network. The procedure for beginning execution of the IIDA programs will be simplified to essentially entering the word "iida" after MULTICS accepts the user's password. The IIDA programs contain all the information necessary thereafter for conducting a search.

IIDA greets the user with an introductory paragraph in which the user is asked for the following: his personal identifier, his type of terminal (printing or CRT), his search mode (user skill-level), and his approach to strategy. If, in a previous search, the user quit with the intent of resuming (see Section 4.12) he is asked if he wants to resume his previous search now. The type of terminal determines if TYPE or DISPLAY is the appropriate command for output. The search mode determines what sequencing controls are to be applied during the session. The approach to strategy will be used by the diagnostic programs to determine if search objectives are being accomplished.

IIDA then proceeds automatically to dial a local node of a data communications network and to log into the DIALOG system. CONIT rules, special-actions, and procedures exist for both dial-out and login. On the CONIT system, these were user-initiated. Automating them for IIDA requires little more than appropriately setting the new-context in the rule at the end of the welcome paragraph. IIDA will normally dial the Telenet node. If it is not available, IIDA will attempt to call DIALOG through Tymnet.

The IIDA account with DIALOG will be arranged so that the default data base will be one of those for which IIDA is written. DIALOG responses during login will be relayed to the user. The user is responsible for any change of data base by issuing one of the normal DIALOG commands for this function, BEGIN and .FILE. The command will be accepted and processed like any other command to DIALOG through IIDA, except that the data base is must be acceptable to the IIDA system.

If the user indicated that he wished to resume an earlier search, the IIDA data structures for the previous search are restored from the external file on which they were saved. All DIALOG commands in that previous search will be reissued to DIALOG in order to restore their files on the search.

4.2 Input from the User

CONIT control programming which allows input from the user will be used by IIDA largely intact. The last rule in the login group of rules takes a special-action which sets a flag which passes control to a mainline segment labelled "luser", acronym for "listen to user." In this segment CONIT sends to the user's terminal a blank line, the time of day, and a user cue. IIDA will send a blank line and one of two user cues: "D?" for user commands that may be directed to either DIALOG or IIDA, and "I?" for user input in response to IIDA questions. A MULTICS subsystem routine, "iox_$get_line", is called to get a line from the user's terminal.

Some delay can be expected between the user's receiving the cue and his completing the input of a command by entering the carriage-return. Delay-time processing of micro-analysis subroutines, described more extensively in Section 4.7, can be carried out during this delay. Two approaches are described in that section: (1) carrying out a fixed set of analyses, or (2) carrying out scheduled analyses until either the schedule is completed or the input has been terminated by the carriage-return. The approach selected can be applied at this point also.

When a line comes back from the user, carriage-return/line-feed is dropped from the end of the input-string. The substring "..-" is placed at the front of the input-string and the substring "-.." is placed at the end. These substrings are used by the rules to recognize the beginning and the end of the input string. CONIT ignores blank lines entered by the user. IIDA, however, will count sequential blank lines. They probably represent some misunderstanding on the part of the user. If this count is in excess of some small number, the user will be notified that these entries are pointless.

The CONIT segment labelled "luser" prints "++++CONIT:" to indicate that a
system response follows. IIDA will print no such header if a DIALOG response
follows, but will print "++IIDA:" if an IIDA response follows. Control then
passes to a procedure which searches the rule table for a match of the current
context and initial characters in the input-stream. When a match is found,
control is managed by the matching rule.

4.3 /HELP command: Assi   ('IDA

The first r es encountered after input from the user are:

      /u/..-/H:..//h:::::::9//..timerec..
      /h-------9/:.,,37,,/h:::::::89//..senmes..

In the first of these rules, the message string identifies the input of any
user command beginning with "/H". It is assumed that the user is seeking help
from IIDA. The special-action in the first rule records the time that help
was requested. This rule also changes the primary search context, i.e.,
position 1 of the context-string, to "h" for "help mode". The second rule
sends a message indexed by 37 in the table "messen" to the user.

Two basic types of help are given to the user: mini-tutorials on DIALOG
commands, and reviews from various perspectives of the current search up to
the present point in the search. Message block 37 (arbitrarily picked for
this example) contains six options which are displayed at the user's terminal
by the special-action "senmes." These options are: (1) a brief description
of the syntax and function of each of the DIALOG commands; (2) a summary of
the commands given in the current search in the order in which they were
given; (3) a summary of the sets created in the order in which they were
created; (4) a summary of all the records viewed; (5) a summary of all the
errors made; (6) a summary of the descriptors used; and (7) a return to main-
line. The history summaries, (2) through (6), correspond to five of the IIDA
data structures being maintained.

At the end of this menu the user is asked to enter a number selected from
the menu. This number is retrieved, saved and evaluated by the following
rules:

      /h------89/:.//h:::::::88//..send..
      /h------88/:.//h:::::::87//..hupdate..
      /h------87/..-1:.,,38,,//h:::::::78//..senmes..
      /h------8/..-2:.//h:::::::66//..help2..
      /h------8/..-3:.//h:::::::66//..help3..
      /h------8/..-4:.//h:::::::66//..help4..
      /h------8/..-5:.//h:::::::66//..help5..
      /h------8/..-6:.//h:::::::66//..help6..
      /h------8/..-7:.//f+ /..htime..
      /h------8/:.,,39,,//h:::::::89//..senmes..

The first rule sets the flag which passes control to "luser" and so picks up the content of the user's input buffer. The second rule updates the help history data structure with the response entered by the user. The third rule transmits message block 38, for example to the user. This gives him a menu of all DIALOG commands. The user is asked to enter a number from the menu indicating which command he wants described.

The fourth through the eighth rule in the above table print out formatted tables of the data structure corresponding to the types of perspectives on the search offered in the menu. The next to the last rule updates the help data structure with the time at which the user asks to return to the mainline, performs the help-use threshold analysis, and then, by the new context, returns him to the front-end mode-oriented instruction. The last rule catches an invalid entry and request input a second time.

The following section of rules processes requests for further instruction on the meaning and use of DIALOG commands:

```
/h-------78/:.//h:::::r:77//..send..
/h-------77/:.//h:::::::76//..hupdate..
/h-------76/..-8:.,,40,,//h::::::66//..senmes..
/h-------76/..-9:.,,41,,//h::::::66//..senmes..
            .
            .
            .
/h-------76/..-21:.//f+//..htime..
/h-------76/:.,,54,,//h:::::::78//..senmes..
```

Again, the first rule picks up the user's response. The second rule updates the data structure on help usage with the response. The third through the fifteenth rules send information about the syntax and function of specific DIALOG commands. The last two rules perform functions analagous to the last two rules of the section previously described.

Finally, the section of rules which allow the user to recycle through the help options are something like:

```
/h-------66/:.,,55,,//h:::::::65//..senmes..
/h-------65/:.//h::::::64//..send..
/h-------64/:.//h::::::63//..hupdate..
/h-------63/..-1:.,,38,,//h::::::78//..senmes..
/h-------63/..-2:.//h::::::66//..help2..
            .
            .
            .
/h-------63/..-7:.//f+//..htime..
/h-------63/..-8:.,,40,,//h::::::66//..senmes..
/h-------63/..-9:.,,41,,//h::::::66//..senmes..
            .
            .
            .
/h-------63/..-21:.//f+//..htime..
/h-------63/:.,,56,,//h::::::66//..senmes..
```

Message block 55 prints a message about the recycle option indicating that any number from the previous tables is acceptable. If there is no match on any of the numbers through the entire range, an error message indicates this and requests user input again. If the user chooses either exit number (7 or 8), the exit-time is recorded, the help-use threshold is checked, and control transfers to the front-end instructions.

The help-use threshold-check, part of the special-action "htime," applies an algorithm to determine if the user of the /HELP command is commensurate with the level of user proficiency indicated by the mode chosen by the user. The algorithm is based on two factors: (1) the number of /HELP commands as a percent of all commands and as a function of the total number of commands and mode; and (2) the time spent in /HELP as a percent of the total search time, and as a function of the total search time and mode.

If the threshold is crossed according to either part of the algorithm, an in-depth analysis of the user's need for help is made. If an in-depth analysis is necessary, the special-action "htime" will change positions 3 and 4 of the context string to "id" (for "in-depth"). A subsequent rule will call the in-depth analysis routine:

    /h-id---9/:.//h:id:::8//..inhelp..

Any user-system interaction is now managed by the rules whose context-string begin with "h-id...". When the in-depth analysis is complete, the user is returned to the front-end instruction by a rule something like the following:

    /h-id---2/:.//f+//.

The in-depth analysis may determine that a less demanding mode is in order for the user. Perhaps a particular command needs further review, or a particular perspective on the search needs to be explained. These situations are identified by information in the help-usage data structure.

4.4 Identifying the Basic Type of Command

A preliminary identification of the command type is necessary to determine if the command is allowed in the current search mode and submode. This identification also allows the command type to be incorporated into the context-string. The rules discussed here (1) perform preliminary general actions on all commands and (2) identify particular commands, moving that identification to the context-string. Table 4.4.1 presents a flowchart of these actions.

The rules given in Table 4.4.2 perform the preliminary general actions. The first rule removes all space characters from the front of the command. The second rule stores the entire command, as it is "left-justified" by the first rule, where it belongs in the data structure array, C_TEXT(C HIST). This includes all characters up to the first ";" or "-..". The third rule copies the entire command as entered by the user up to the first ";" or "-.." to the host-message buffer. This buffer will be sent later to the host if and only if it is identified as being both syntactically valid and valid for the user's search mode. The semicolon is considered a delimiter since the user may have entered several DIALOG commands in a stack.
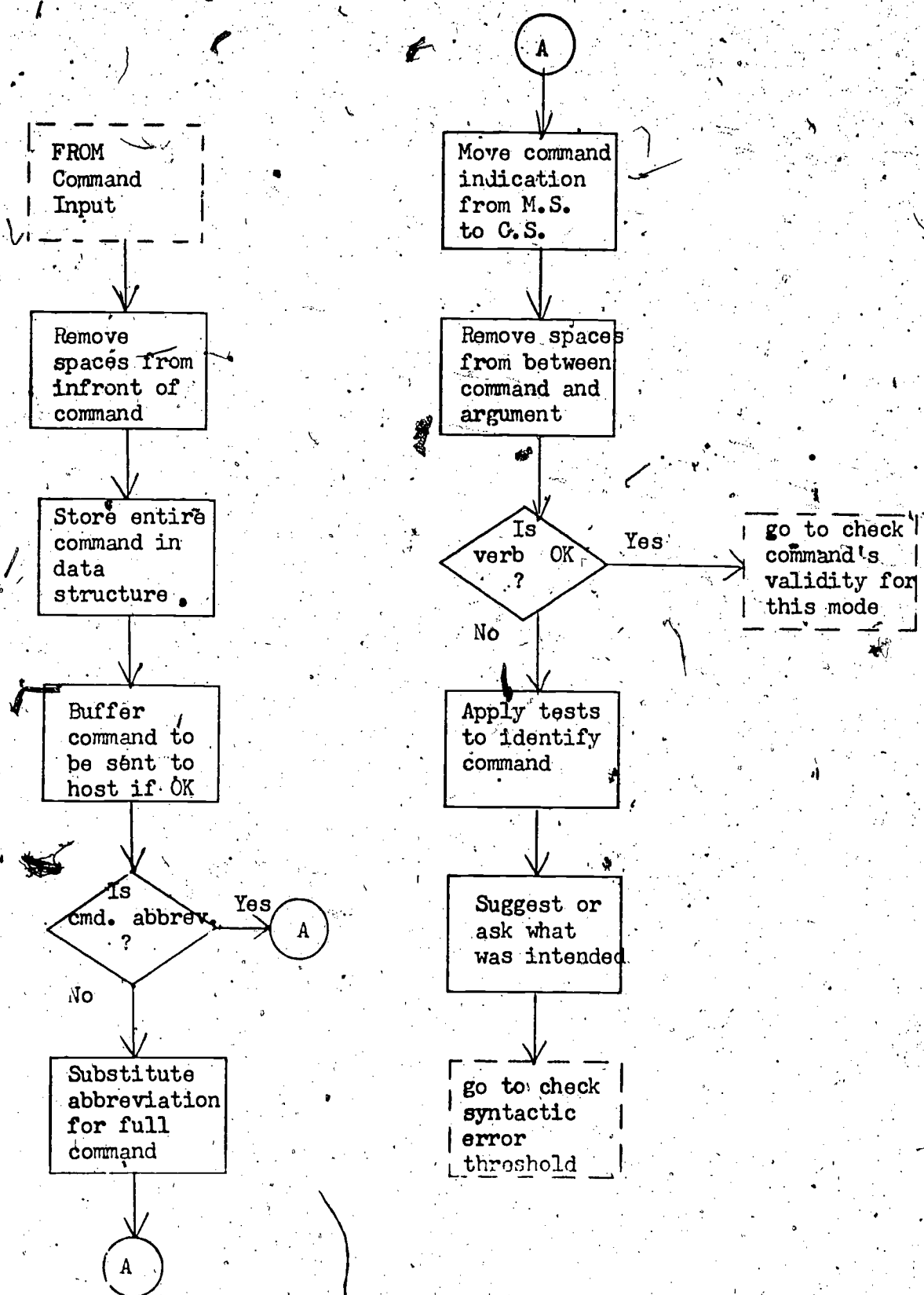
FROM
Command
Input

↓

Remove
spaces from
infront of
command

↓

Store entire
command in
data
structure

↓

Buffer
command to
be sent to
host if OK

↓

Is
cmd. abbrev.
? —— Yes —→ (A)

No

↓

Substitute
abbreviation
for full
command

↓

(A)

(A)

↓

Move command
indication
from M.S.
to C.S.

↓

Remove spaces
from between
command and
argument

↓

Is
verb OK
? —— Yes —→ go to check
command's
validity for
this mode

No

↓

Apply tests
to identify
command

↓

Suggest or
ask what
was intended

↓

go to check
syntactic
error
threshold

Table 4.4.1.   General Flow of the Basic Identification of Command Type.
"M.S." stands for "match string".   "C.S." stands for "context string".

rul /s/ :. ** skip all spaces that precede command.

rul /u/:...stcom.. ** special-action stores command as a string in its

        ** place in the data structure: C_TEXT (C_HIST).

rul /ucon/:...copyc.. ** copy user command to host message; command

      ** will be sent later to host only if it is without error.

Table 4.4.2. General section of rules for syntactic analysis.

---

rul /ucon/..-type:.,,..-t,,..sub.. ** substitute ."t" for "type"

rul /ucon/..-select:.,,..-s,,..sub.. ** substitute "s" for "select"

rul /ucon/...-t:.//ucon:::ot// ** move indication of TYPE to context-string

rul /ucon---ot/ :. ** remove any and all spaces between command and argument

rul /ucon/..-s:.//ucon:::s// ** move indication of SELECT to context-string

rul /ucon---s/ :. ** remove any and all spaces between command and argument

rul /ucon/..-out:.,,34,,//mcsy+//..senmes.. ** perhaps "type" was intended

rul /ucon/..-typ :.,,34,,//mcsy+//..senmes..

rul /ucon/..-get:,,,35,,//mcsy+//..senmes.. ** perhaps "select" was intended

rul /ucon/..-retrieve:.,,35,,//mcsy+//..senmes:.

rul /ucon/..-merg:.,,36,,//mcsy+//..senmes.. ** perhaps "combine" was intended

Table 4.4.3. Typical rules of the identification section. The first group
substitutes an abbreviation for the full command. The second group moves the
identification of the command verb to the context-string and removes all
blanks between verb and argument. The third group tries to identify command
verbs in error.

Examples of three types of rules typical of the command identification section are given in Table 4.4.3. These rules: (1) convert all full commands to their abbreviations; (2) match the command verb in the message-stream and put its code into the context-string; (3) remove any and all spaces between the command and its argument; and (4) attempt to identify invalid command verbs.

Converting full commands to their abbreviations involves the CONIT special-action "sub" which substitutes what is in the user-message part of the rule (in this case, the abbreviation) for what is in the match-string part of the rule (in this case, the full command). Moving the command into the context-string involves matching the abbreviation in the match-string with the initial part of the message-stream and setting the new-context to include an indicator of the command verb. (See Appendix A for a list of context-string command indicators.) Removing spaces involves an iterated match on the space character.

If user commands directed to DIALOG are not identified in either their full or abbreviated forms, a set of rules applies various tests to try to determine what command was intended. These tests include: (1) scanning a list of possible but illegal synonyms; (2) scanning a list of common misspellings; (3) applying letter-frequency algorithms to identify misspellings; (4) identifying commands by characters and constructs in the arguments; and (5) giving up and notifying the user that his command is uninterpretable.

In every case, the invalid command is brought to the user's attention with a message. This may include a suggestion to get help on the syntax and use of the command through /HELP. Since it is likely that the error was a keying problem at the terminal, the first time such an error is made the user will be told how to correct a character and how to correct a line in the MULTICS system. After checking the syntactical error threshold (discussed in Section 4.6), control returns for the next input from the user.

If this process identifies the command entered by the user to be "END", IIDA will request at this point a verification that the user indeed wishes to end his search. The conversation may proceed as follows:

D? END

++++IIDA:

DO YOU REALLY WANT TO END YOUR SEARCH NOW?

                                                                    ++++IIDA.
I (YES OR NO)?

If the user enters "YES" the search will soon be terminated as described in Section 4.12. If the user enters "NO" he will be cued by the mode-oriented instruction for the input of his next DIALOG command. If the user instead enters another DIALOG command, the cue will be bypassed and control will be transferred to the beginning of command-analysis. When the user returns to the search with either "NO" or another DIALOG command, the fact that he entered END but continued with the search will be recorded in the command history data structure.

4.5 Exercise and Assistance Mode Control

When he logged in, the user decided to search under one of four modes, corresponding roughly to his expertise in searching. The three exercise modes control the sequence of searching, that is, the set of commands that may be used at each point in the search. In the first exercise mode, complete search commands are prompted throughout, giving the user experience with a "canned" search. In the second exercise mode, a subset of the full language is allowed, with the user searching a subject of his own choosing. In the third exercise mode, the full search language is allowed with the user's own search, although the sequence is still controlled. In the fourth or assistance mode, all search commands are allowed with no controls placed upon the sequence of commands.

When the user selects the mode he wishes to search under, the mode's code is stored in position 6 of the current context string. This code will remain unchanged throughout the search. At the same time the code for the first submode, the letter "a", is stored in Position 7. The submode code determines precisely which commands are allowed at each point in the search in the chosen mode. The submode code is incremented in the mode-specific instructions (See Section 4.13) when: 1) the previous command is not in error, and 2) a new command (or form of command) allowed in the submode has been successfully issued.

The validity of a command for a given mode and submode is determined by the section of rules in the rule-table that will now be discussed. If a command is identified as invalid for the mode, a message to that effect is sent to the user, control transfers to the mode-specific instructions (See Section 4.13), and the user is asked to enter a command valid for the current mode. If a command is identified as valid with respect to the mode, position 1 of the context string becomes "m" to indicate that the mode-check was passed. Control then passes to the syntax-evaluation section of the rules.

Context positions 6 through 9 provide all the information used by the rules to determine command validity for mode and submode. As mentioned earlier, positions 6 and 7 indicate mode and submode; positions 8 and 9 indicate the current search command and type of command. See Figure 4.5.1 for examples of rules evaluating the validity of commands for the first three submodes of the second and third exercise mode.

For the first "canned search" mode, a match is sought on the explicit command, both verb and argument, since both are required for an entry to be acceptable. Thus the rules for the first mode will include match-strings for the expected commands in full. The new context of those commands which pass the test is "rcon-----99". Control passes to the rule which sends the command to the retrieval system (see Section 4.7).

Note in Figure 4.5.1 that all commands issued under the second and third modes, whether valid or invalid, are covered by each submode section of the rules. This coverage may be either explicit or by default. The subset exercise mode will tend to be explicit about what commands are acceptable, while the full-command-set exercise mode will tend to be explicity about what commands are unacceptable.

rul /uc---sabe/:.//mc+// ** BEGIN only allowed in mode 2, submode 1

rul /uc---sa/:.,,5,,//fcer+//..senmes.. **Anything else is an error

rul /uc---sbs/:.//mc+// ** SELECT and EXPAND only are

rul /uc---sbex/:.//mc+// **allowed in mode 2, submode 2

rul /uc---sb/:.,,6,,//fcer+//..senmes.. ** Anything else is an error

rul /uc---scs/:.//mc+// ** A second SELECT and EXPAND are

rul /uc---scex/:.//mc+// **allowed in mode 2, submode 3

rul /uc---sc/:.,,7,,//fcer+//..senmes.. **Anything else is an error

rul /uc---fabe/:.//mc+//. ** BEGIN is O.K. in mode 3, submode 1

rul /uc---fas/:.//mc+// ** SELECT is O.K. in mode 3, submode 1

rul /uc---fae/:.//mc+// ** EXPAND and EXPLAIN and END are all O.K.

rul /uc---fa/:.,,21,,//fcer+//..senmes.. **Anything else is an error

rul /uc---fbbe/:.,,22,,//fcer+//..senmes.. **BEGIN not O.K. mode 3 sub 2

rul /uc---fbo/:.,,23,,//fcer+//..senmes.. ** Output commands not O.K.

rul /uc---fb/:.//mc+// ** All other commands are O.K. in mode 3 sub 2

rul /uc---fcs/:.,,24,,//fcer+//..senmes.. ** SELECT not O.K. mode3 sub3

rul /uc---fcex/:.,,25,,//fcer+//..senmes.. **EXPAND not O.K. mode3 sub3

rul /uc---fcbe/:.,,26,,//fcer+//..senmes.. **BEGIN not O.K. mode3 sub3

rul /uc---fc/:.//mc+// **All other commands O.K. in mode3 sub 3

Table 4.5.1. Examples of mode-control rules for the second and third modes.
The code "er" in positions 3 and 4 of the context-string is the general error
indicator.

28

4.6 Parsing Arguments of DIALOG Commands

Parsing arguments of commands entered by the user will be achieved mainly by PL/I special-actions called by rules. A typical rule, calling the COMBINE command parser, might be:

            rul /mcon---c/:.//mcon:::::5//..parsec..

"Parsec" is the name of the COMBINE command parsing special-action. The command's argument is isolated from the pointer in the input-stream to either the first semicolon (stacked commands) or the terminating substring "-..".

Several functions are performed in these special-actions. These include: (1) ignoring all spaces where DIALOG would ignore a space; (2) isolating all lexemes required by the data structure and storing them where they belong; (3) determining what errors, if any, are present and fixing an error code in context-string positions 3 and 4; and, sometimes, (4) generating auxiliary data for later use. These functions are performed in an order which is most efficient for the particular command involved.

The lexemes required by the data structures are discussed in Section 5. Identification of errors will be presented below in two ways: (1) a list of major syntactic features of every command will follow, indicating the types of errors which may develop in each command; and (2) flowcharts for the four commands with the most complex arguments: EXPAND, SELECT, COMBINE, and the output commands. Particular error codes have yet to be assigned and error messages have yet to be designed. An instance of the generation of analytic data appears in the COMBINE command argument where the extended argument is built. It is done here to take advantage of a single control structure interlocking this task with parsing and verification. Because it is most complex, a further explanation of the COMBINE argument parse is also given below.

First, the list of the major syntactic features in which errors may be identified:

BEGIN

- do not check for space between the command and its argument
      (such a space is not allowed by DIALOG)
- argument must be a single numeric
- number must be a valid data base code

.FILE

- same conditions as specified under BEGIN

EXPAND

- if argument contains a tag-delimeter (= or /),
  the tag must be a valid one
- if argument is of the form En:
    - nothing more is allowed in the argument
    - there must have been a prior EXPAND
    - n may not exceed the number of entries in the prior E-table
- if argument is of the form Rn:
    - nothing more is allowed in the argument
    - there must have been a prior "relate" (EXPAND EXPAND)
    - n may not exceed the number of entires in the prior R-table
- argument may not be simply a number or a stop word or contain
          no alphabetic characters

PAGE

- no argument at all is allowed

LIMIT

- first entry of the argument must be a valid set number
- a slash-delimeter (/) must be present and contiguous with set number
- argument following the /-delimeter must be valid for the data base

DISPLAY SETS

- must be exactly "DISPLAY SETS" or "DS"

SELECT

- if argument contains a tag-delimeter (= or /),
  the tag must be valid
- if argument is of the form $En_1, En_2, En_3 \ldots$
    - there must have been a prior EXPAND
    - no n_j may exceed the number of entires on that E-table
    - for every range entry, $En_k - En_{k+1}$, n_{k+1} must exceed n_k
    - no other type of entry, e.g., index term, may be on list
- if argument is of the form $Rm_1, Rm_2, Rm_3 \ldots$
    - there must have been a prior "relate" (EXPAND EXPAND)
    - no m may exceed the number of entries on that R-table
    - for every range entry, $Rm_k - Rm_{k+1}$, m_{k+1} must exceed m_k
    - no other type of entry may be on list
- if argument is a string search type:
    - no spaces may be embedded in the argument
    - parentheses for links must match
    - contents of the parens must be valid link codes
- truncation symbol (?) may not be embedded in an argument

COMBINE

- if /-delimeter is present, argument must consist of
    - valid set number followed by:
    - a dash (-) followed by:
    - a larger valid set number followed by:
    - the slash delimeter (/) followed by:
    - a valid operator: AND or OR
- Boolean operators and operands must be conventionally ordered
- the only alpha characters allowed are: AND, OR, NOT
- set numbers may not exceed the largest set created
- parens must match meaningfully

EXPLAIN

- argument must be on current list of explainable items
- check any argument not matched for misspellings and synonyms

DISPLAY, TYPE, AND PRINT

- null argument to TYPE is acceptable, but not to PRINT
- argument to left of first / is a DIALOG access # or valid set #
- if argument is an access #, at most one more / may follow
- first / must be followed by a valid format #
- second / must be followed by a valid record #
- if a range of record #s, n-m, they must be valid and n<m
- PRINT may take the SORT argument
- PR- is acceptable for cancelling last off-line print command

---

The analysis of the COMBINE argument does the following: (1) it recognizes the argument as being either valid or invalid, and if invalid, in what respect; (2) it isolates the set numbers in the argument and stores them for cluster analysis; and (3) it builds the extention of the COMBINE argument, i.e., the string in which all of the argument's set numbers have been translated into index terms.

As shown in the flowchart in Table 4.6.1, the input stream is scanned to isolate numerics (set numbers); left and right parenthesis; and Boolean operators, "AND", "OR", and "NOT". Three variables are used in the analysis: s counts sets, t codes the last type of character encountered, and d measures parenthetical depth. When the end of the input stream is reached, d must equal 0, t must equal 1 (a set number) or 4 (a right parenthesis), and s must be greater than 1. Sequences allowed by the analysis are: a set number must follow a Boolean op or a left paren; a Boolean op must follow a set number or a right paren; a right paren must follow a set number or a right paren; a left paren must follow a Boolean op or a left paren.

Code	Begin Value	Name	Assigned Values
s	0	Set count	for each set, $s = s + 1$
t	3	Char. type	1 = set number
			2 = Boolean operator
			3 = left parenthesis
			4 = right parenthesis
d	0	Paren depth	for left paren, $d = d + 1$
			for right paren, $d = d - 1$

If a syntactical error is identified in the command argument, positions 3 and 4 of the context string are changed by the parsing special-actions for the various commands to a numeric code. This code represents the index of the message in the table "messen" which discusses that particular syntactical error. If no syntactical error was identified, the first four positions of the context-string remain "mcon". The code "m" stands for having passed mode-control; the code "con" stands for the unexceptional connect status (See Appendix A). A protocol number is always placed in position 10 of the context-string.

The first rule following those which call for the parsing special-actions handles all commands without syntactical errors:

        rul /mcon-----5/:.//rcon::::::99//..rsendc..

This rule sends the command in the host buffer to the host by the special-action "rsendc". Position 1 of the context-string is changed to "r" to indicate that a response from DIALOG is expected next. The special-action "rsendc" allows control to continue processing rules without immediately looking for a response from the retrieval system. By the protocol code "99" in the new context, control is transferred to the delay-time processing section of the rules (see Section 4.7).

Every error condition identified in the syntax parse and analysis triggers an error message in this section by rules like the following:

        rul /mc23/:.,,23,,//mcsy+//..senmes..
        rul /mc24/:.,,24,,//mcsy+//..senmes..

Errors, previously coded 23 and 24 in context-string positions 3 and 4, now call for messages indexed by 23 and 24 in "messen", the message table. These messages address themselves to particular syntactic errors. "Senmes" is the special-action which takes these paragraphs out of "messen" and sends the message to the user. The new context string replaces the numeric code with "sy". The context "mcsy..." is found at the end of this section of rules as follows:

        rul /mcsy-----5/:.//mcsy:::::4//..thsyn..
        rul /mcsy-----4/:.//fc+//
        rul /mcid-----4/:.//fc+//..idsyn..

The first of these ruls calls the special-action "thsyn", that is, syntax threshold analysis. A formula in this special-action determines if the user is encountering special difficulty with the syntax of commands. The formula
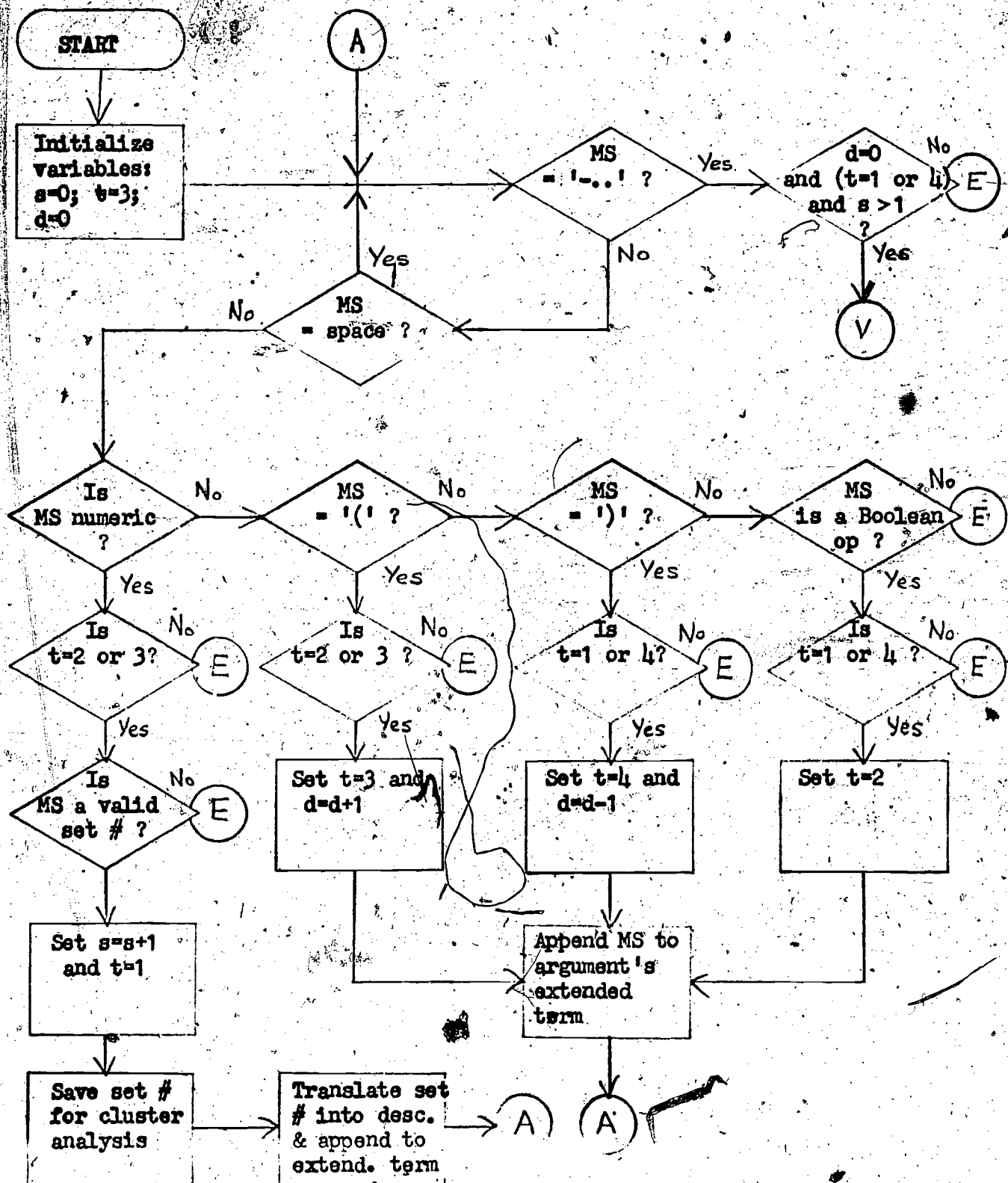
Table 4.6.1. Flowchart for the analysis of COMBINE command arguments. (E) indicates an error condition to be made explicit to the user. Control transfers from (E) to the cue for the next user command. (V) means command is valid; continue rule interpretation. MS means 'match string,' i.e., the substring beginning at the pointer in the input stream. Assume that the pointer is automatically incremented after a match to the character just beyond the match. The letters s, t, and d are variables which stand for 'set count,' 'type of lexeme,' and 'parens depth,' respectively.

33.

START

Set output default parameters

MS = '-..' ? — Yes → V

No

MS numeric ? — Yes → A

No

Take MS up to first / else up to the '-..'

Vald access # this db ? — No → E

Yes

MS = "-.." ? — Yes → V

No

Valid fmt. # ff. '/' ? — No → E

Yes

C

A

Is it a valid set # ? — No → E

Yes

MS = '-..' ? — Yes → V

No

MS = '/' ? — No → E

Yes

MS Is valid fmt. # ? — No → E

Yes

MS = '-..' ? — Yes → V

No

MS = '/' ? — No → E

Yes

B

B

Is 1st record # valid ? — No → E

Yes

MS = '-..' ? — Yes → V

No

MS = '-' ? — No → E

Yes

Is 2nd rec # <= max ? — No → E

Yes

2nd rec # > 1st # ? — No → E
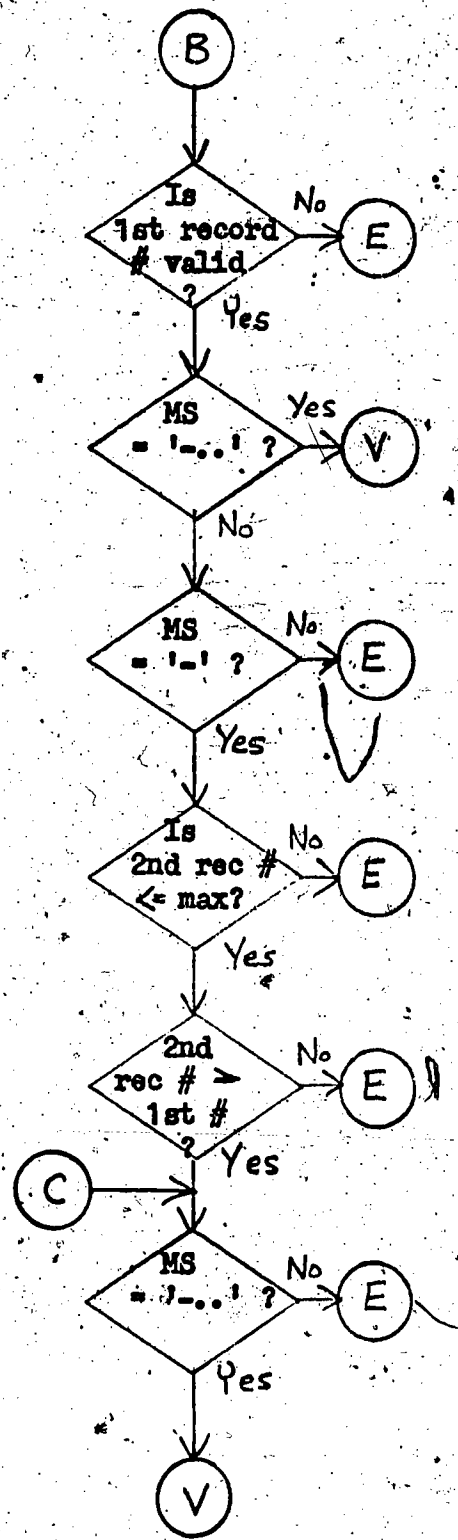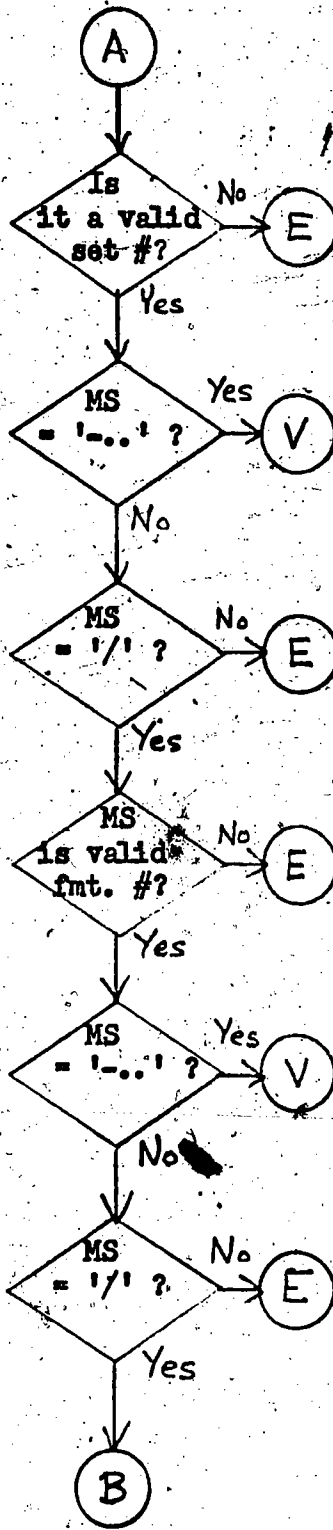
Yes

C →

MS = '-..' ? — No → E

Yes

V
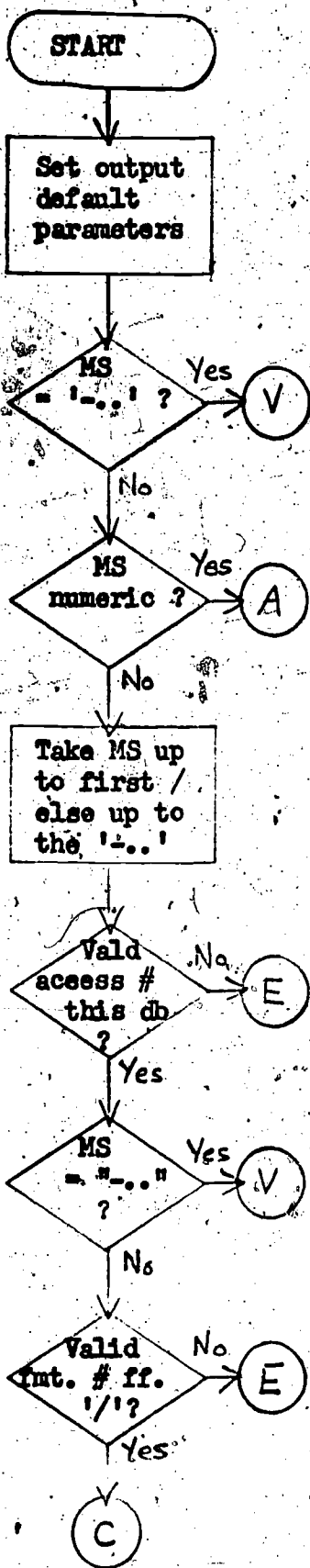
Table 4.6.2. Flow of parsing output commands. Codes and assumptions are those indicated in the note under Table 4.6.1.

34

START

MS include '='? — No → MS include '/'? — No → MS begin with 'E'? — No → MS begin with 'R'? — No → (A)

MS include '='? — Yes → substring prior to '=' valid tag? — No → (E)
substring prior to '=' valid tag? — Yes → (V)

MS include '/'? — Yes → substring following'/' valid suf? — No → (E)
substring following'/' valid suf? — Yes → (V)

MS begin with 'E'? — Yes → Does a numeric follow? — No → (A)
Does a numeric follow? — Yes → Was there a prior EXPAND? — No → (E)
Was there a prior EXPAND? — Yes → Is num <= highest E-num? — No → (E)
Is num <= highest E-num? — Yes → MS = '-..'? — No → (E)
MS = '-..'? — Yes → (V)

MS begin with 'R'? — Yes → Does a numeric follow? — No → (A)
Does a numeric follow? — Yes → Was there a prior "relate"? — No → (E)
Was there a prior "relate"? — Yes → Is num <= highest R-num? — No → (E)
Is num <= highest R-num? — Yes → MS = '-..'? — No → (E)
MS = '-..'? — Yes → (V)

(A) →

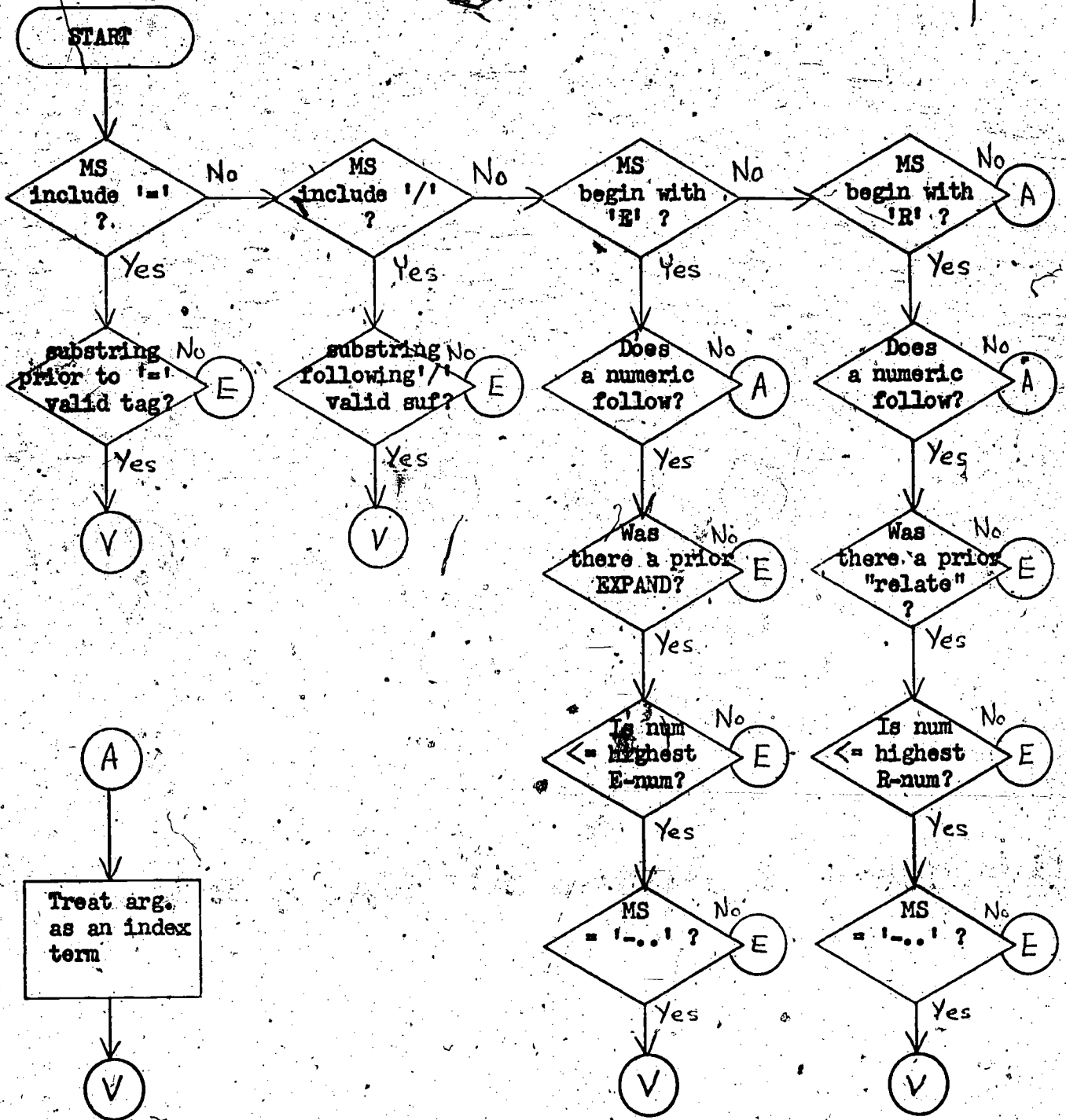Treat arg. as an index term

→ (V)

Table 4.6.3. Flowchart for syntactic analysis of *EXPAND command argument. Codes and assumptions are the same as those indicated in the note under Table 4.6.1.
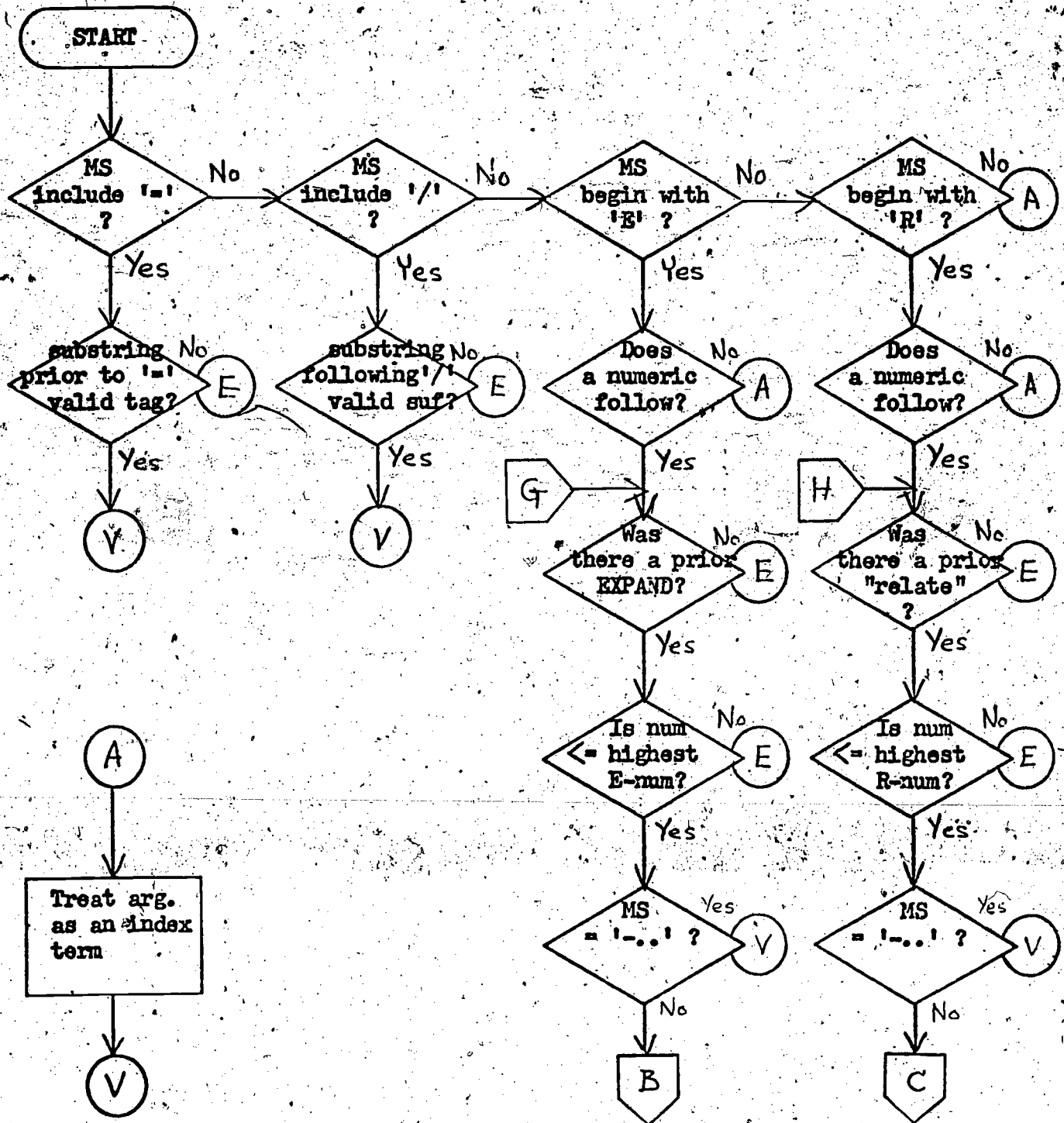
Table 4.6.4. Flowchart for syntactic analysis of SELECT command argument. Codes and assumptions are the same as those indicated in the note under Table 4.6.1.
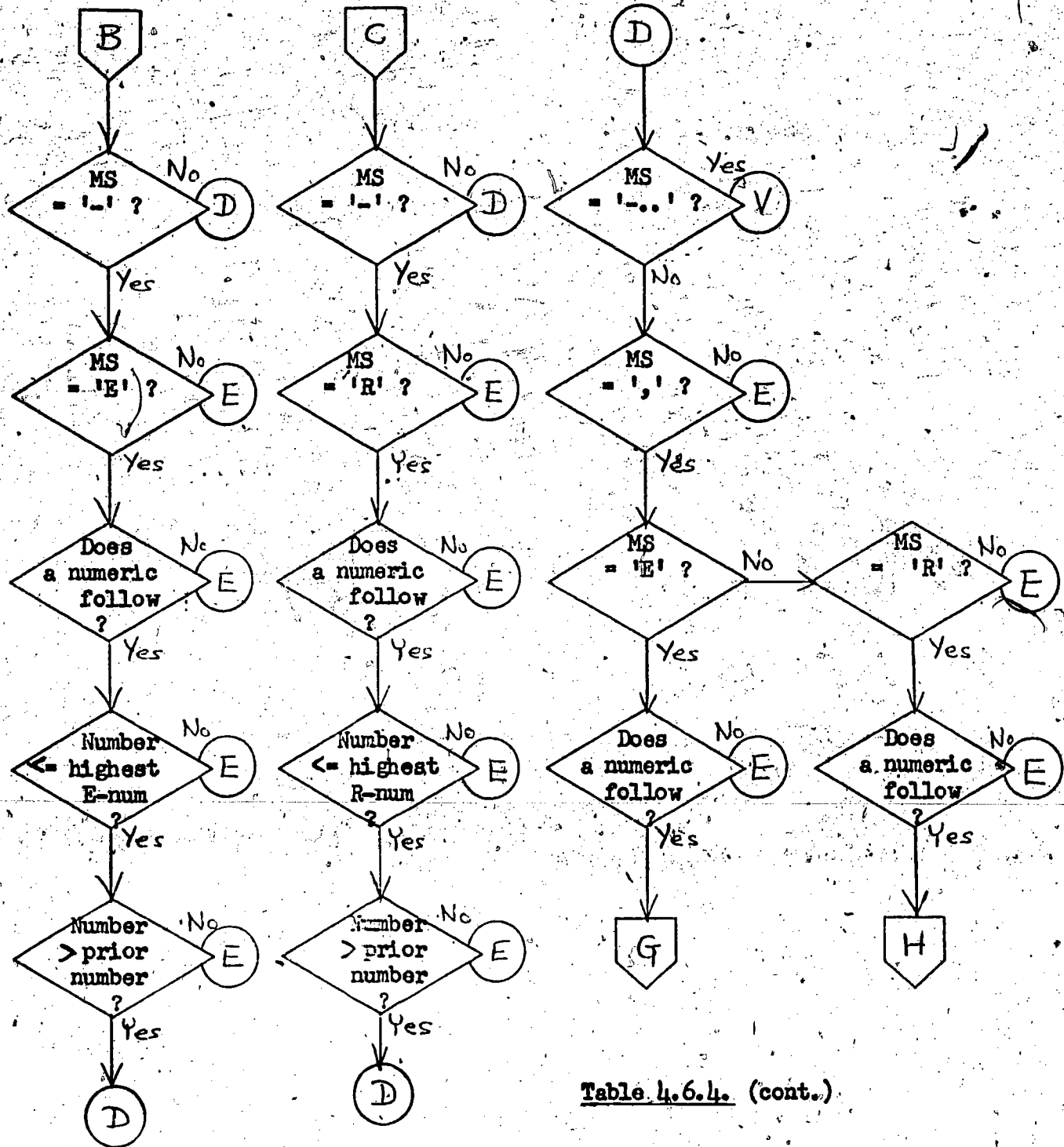
Flowchart — Column B:

B → MS = '-' ? — No → D
Yes ↓
MS = 'E' ? — No → E
Yes ↓
Does a numeric follow ? — No → E
Yes ↓
Number <= highest E-num ? — No → E
Yes ↓
Number > prior number ? — No → E
Yes ↓
D

Flowchart — Column C:

C → MS = '-' ? — No → D
Yes ↓
MS = 'R' ? — No → E
Yes ↓
Does a numeric follow ? — No → E
Yes ↓
Number <= highest R-num ? — No → E
Yes ↓
Number > prior number ? — No → E
Yes ↓
D

Flowchart — Column D:

D → MS = '-..' ? — Yes → V
No ↓
MS = ',' ? — No → E
Yes ↓
MS = 'E' ? — No → MS = 'R' ? — No → E
'E' Yes → Does a numeric follow ? — No → E ; Yes → G
'R' Yes → Does a numeric follow ? — No → E ; Yes → H

Table 4.6.4. (cont.)

involves: the total number of commands thus far in the search, the time thus far in the search, commands with syntactic errors as a percent of all commands, and time spent making commands with syntactic errors as a percent of the entire search time. If the threshold by this formula is exceeded, postition 2-3 of the context-string is changed to "id" for "in-depth analysis." The particular parameters of the formula are yet to be determined.

In either case, following a syntactic error threshold analysis, the next matched rule changes the context to begin with "fc..." for the front-end mode-oriented instruction. If the threshold has been crossed, however, the special-action "idsyn", short for in-depth syntactic analysis, is also executed. Here the problem with the syntax of commands is pinpointed by the application of various analyses. Is the problem general with all commands or specific to a particular command? Perhaps a review of a specific command with /HELP is in order. The special-action "insyn" will make an appropriate suggestion or may actually take the further action needed.

4.7 Delay-Time Processing and DIALOG Response.

All commands which reach this point have passed the syntax and mode test and have been sent to the host by the special-action "rsendc" in the rule discussed in Section 4.6. The special-action "rsendc" is an IIDA variant of a CONIT special-action. Its effect is to send the command to DIALOG, but then to continue processing rules in the rule-table. This continuation allows IIDA to utilize the delay between sending a message to the host retrieval system, DIALOG, and receiving a response. The rules that logically follow the one containing the special-action "rsendc" will specify various micro-analyses as special-actions. These procedures will both generate data reflecting analysis of the search strategy as a whole and store it in appropriate places in the data structures.

The details of the micro-analyses are discussed in Section 6. Two approaches to micro-analysis are being considered: a fixed set of analyses called by rules in sequence; and (2) a few fixed analyses which also create a stack of calls for further analyses which are then done one at a time as long as the input buffer from the retrieval system remains empty. It has not yet been determined whether the IIDA processing is sufficiently time-consuming to warrant the second approach. Programming will begin using the first approach. If noticeable delays attributable to IIDA analyses are experienced, the "system programming" involved in the second approach will be undertaken.

A few definitions are now in order. A "string" means a succession of commands of the same type. A "string-type" means the type of the commands which make up the string. A "cycle" or "group" means a succession of strings in which each string-type is greater than the previous string-type. A "cluster" means a set of sets with similar or associated origins or bases. Refer to Section 6 for precise definitions of the micro-analyses mentioned and underlined in the discussion that follows.

For every type of command, membership in the current group will be evaluated. If the current command belongs to the current group, it is recorded as such in the data structures and a dwelling-check is either

executed (approach 1) or schedules for execution (approach 2). If the current command does not belong to the current group, a new group is noted in the data structure, and a group-change-analysis is either executed or scheduled. Also, for every type of command, a regression-check is either executed or scheduled.

For every set-generating command, i.e., SELECT or COMBINE, a cluster-check will be made. The cluster check consists of determining if the current set belong to the cluster of the immediately previous set. If the current set belongs to the cluster, a dwelling-check is either executed (approach 1) or scheduled (approach 2). If the beginning of a new cluster is identified, an inter-cluster-check is either executed or schedules. In both cases, a thrashing-check is also either executed or scheduled.

For every SELECT command argument, the descriptors-used list will be scanned for similar arguments. If any are found to begin with the same five letters, for instance, this similarity is noted in the descriptors-used list.

Several extension routines are necessary for amplifying the arguments of SELECT and COMBINE for later comparisons. Whenever an item is SELECTed from an E-table or R-table, the extension routine looks up the item's descriptor equivalent on the latest E-table or R-table (saved in a data structure) and saves it as the extended form of the argument. If the SELECT argument consists of a list of such items, each one is thus extended and connected by the Boolean 'OR'. In this way it is made to look like an argument of a COMBINE command, with which it can later be compared.

The initial extension of all COMBINE arguments has already been presented in Section 4.6. This micro-analysis is interlocked with the syntax analysis there, taking advantage of an already-existing control structure. This argument, however, must also be reduced to a normal-form for effective comparison with other arguments. The normal-form-reduction: (1) drops all spaces from within the extended argument; (2) drops all parentheses from the argument by applying the distributive law; and (3) alphabetizes all scopes of the resulting argument -- first by NOT-groups, then by AND-groups, and finally by the remaining OR-group (the whole argument).

When the delay-time micro-analyses are completed (approach 1) or when the buffer from the retrieval system contains data (approach 2), a rule will call for the CONIT special-action "rsend" which sets the flags which transfer control to pick up the contents of the retrieval system buffer. The rule will be something like:

        rul /rcon------99/:.//rcon:::::95//..rsend..

The first line, often the only line from DIALOG, is now in the input stream.

4.8. Processing DIALOG error messages.

The input-stream now contains a response from the retrieval system or possibly from the telecommunications network. This response may be either expected for the given command or unexpected. An instance of the former is information about a set. An instance of the latter would be an error message

or some other condition which could not be evaluated before-hand by the syntax analysis (See Section 4.6). Examples of such errors and conditions are: disk-storage overflow, external file I/O error, no core available, and dropped by host.

Immediately after receiving a DIALOG response, rule interpretation will begin on a section of rules which seek matches on all unique prefixes of known error messages. If a match is found, the DIALOG or network error message is printed at the user's terminal. A more detailed explanation of the problem is then called from the message-table "messen". This section of the rule-table is fairly straightforward. Example rules are:

```
rul /rcon------95/.-NO CORE:.//rcon:::::94//..copys..
rul /rcon------94/:.,,64,,//rcon::::2//..senmes..
rul /rcon------95/.-DISK:.//rcon::::94//..copys...
rul /rcon------94/:.,,65,,//rcon:::::2//..senmes..

      .
      .
      .

rul /rcon------2/:.//rcon+//..syerr..
```

If a match is found on an error-message, a special-action "copys" will copy the full error-message to the user's buffer and will send the message to the user. The context is also changed to match the rule immediately following it. This rule takes a descriptive paragraph about the problem from "messen" and sends it to the user. The context is now changed to match on a rule whose special-action "syerr" notes the system error in the command and error data structures. The last rule in the above set of examples also changes the context to next match on the first front-end instruction rule applicable to the current mode.

CONIT has extensive system programming for identifying timeouts and other unforseeable breaks in telecommunication. The rules, special-actions, control structures and procedures which manage this aspect of system errors will be maintained and used intact by IIDA. Rules and special-actions will be added in these sections to record such events in at least the error history data structure. In these cases, control usually passes back to the user after he is informed about the problem. If the problem involves a break with the host system, though, the user may be given a choice of (1) waiting on-line while IIDA tries the alternate network, (2) waiting on-line for IIDA to try reestablishing communication in three to five minutes, or (3) temporarily quitting the search to pick it up again at a later time (see Section 4.12).

4.9. Processing Expected DIALOG Responses.

If no match is found on any of the error prefixes, it may be assumed that the response from DIALOG is a normal and expected one. The rules in this section are, again, straightforward although the special-actions which they call may be, fairly involved. The rules identify, by the command code in the context string, which response parsing special-action should be called. The special-actions do the following: (1) relay the response to the user, (2) parse the response, and (3) update the data structures with the lexemes isolated by the parse. The rules of this section are:

```
rul /rcon---be/:.//acon+//..rsvpbe.. ** BEGIN
rul /rcon---s/:.//acon+//..rsvps..  **SELECT
rul /rcon---en/:.//acon+//..rsvpen.  **EXPLAIN
rul /rcon---zz/:.//acon+//..rsvpzz.  **END
rul /rcon---ex/:.//acon+//..rsvpex.  **EXPAND
rul /rcon---c/:.//acon+//..rsvpc..  **COMBINE
rul /rcon---od/:.//acon+//..rsvpod..  **DISPLAY
rul /rcon---ot/:.//acon+//..rsvpot..  **TYPE
rul /rcon---op/:.//acon+//..rsvpop..  **PRINT
rul /rcon---ds/:.//acon+//..rsvpds..  **DISPLAY SETS
rul /rcon---lt/:.//acon+//..rsvplt..  **LIMIT
rul /rcon---pa/:.//acon+//..rsvppa..  **PAGE
```

Each of these rules calls for the special-action corresponding to the command and then changes the context to "acon+", that is, the context of that section of the rules in which the diagnostic threshold analyses take place. Each of the above special-actions will now be described in further detail.

rsvpbe. If BEGIN'S argument indicates a data base which is not on-line, the DIALOG error response section of the rules (See Section 4.8) would have identified a message to this effect. If the argument indicated a data base not handled by IIDA, this would have been identified by the syntax-checking section of the rules (See Section 4.6). Special-action "rsvpbe" will relay the standard DIALOG response lines to the user. No parsing of the response is necessary. The special-action will first save on files most of the internal data-structures of the immediately previous search if this is other than the first search in the session. In this case, it will the reinitialize the data-structures for a new search.

rsvps, rsvpc, & rsvplt. The responses to SELECT, COMBINE, and LIMIT are simple and very similar. First, the response will be relayed to the user. Then the following three elements will be isolated from the input string: the set number, the number of postings and the echoed argument. The expected set number will be compared with the actual set number. These should not differ. The isolated argument may be a truncation of the previously isolated argument, but this should not really matter. The number of postings is the only significant new datum which is stored in the data structures by these special-actions.

rsvpen. The response to a request for a DIALOG explanation should be relayed in full to the user. Nothing about the response needs to be parsed or saved.

rsvpzz. The response to END is relayed in full to the user. See Section 4.12 for the end-of-search activity which includes: saving the data structures for later analysis, giving the user an opportunity to BEGIN again, and logging the user out of DIALOG if that is indicated.

rsvpex and rsvppa. The response to EXPAND and PAGE is relayed, one line at a time, to the user. Each line relayed is also parsed and stored in a temporary data file, to be used should IIDA need to make a reference to an E-table entry. All descriptors will be saved in the descriptors data-structure. Parsing consists of isolating substrings by position from the input string. Other E-table attributes, e.g., highest entry, beginning entry, and ending entry, may be saved in other permanent parts of the data-structures.

rsvpod and rsvpot. Responses from DISPLAY and TYPE are handled exactly alike. Their parsing depends upon the command format, which was isolated by the parse of the argument. The function of parsing here is to save the accession numbers, should the user later wish to review records viewed. The response is relayed verbatim to the user. If the format makes it a meaningful question, (i.e., not simply a list of accession numbers) the relevance of the output records is elicited from the user. This probably will be done after every few records have been viewed. These responses will be stored in the sets-viewed history data-structure.

rsvpop and rsvpds. Responses to PRINT and DISPLAY SETS are relayed verbatim to the user. The contents of these responses are predictable. In the case of display sets, the response can be reconstructed directly from the set history data-structure. They will therefore not be parsed and data which they contain will not be stored.

4.10. Evaluation of Diagnostic Thresholds.

Threshold analyses are made to determine if the search is making progress (by IIDA's definition) or if the search is developing strategical errors which should be further analyzed and called to the attention of the user. Two threshold analyses have already been discussed: (1) following /HELP calls (Section 4.3); and (2) following syntactic errors (Section 4.6).

The behavioral characteristics that threshold analyses will identify at this point include:

A. Failure to Use Facilities

1. SELECT

   - relatively few sets selected compared to the
        total number of commands or sets
   - many sets result in 0-postings indicating failure
        to use EXPAND to identify valid index terms

2. COMBINE

   - few sets COMBINEd relative to many sets created
   - arguments excessively use one type of logical operation
   - the contracted form, Cm-n/op, could be used but wasn't

3. Output commands (TYPE, DISPLAY, PRINT)

   - these commands are seldom or never used
   - only one format of output is ever used
   - the first few records of very large sets are viewed
   - only SELECTed sets, not COMBINEd sets, are viewed
   - sets are PRINTed off-line without having been viewed
        first by the TYPE command on-line (no sampling)

B. Excessive Use of Facilities

1. SELECT

- too many SELECTs are issued relative to the total
  number of commands or sets
- excessively large sets are regularly SELECTed

2. COMBINE

- used too often relative to the number of sets with which it
  is working

3. Output commands

- too many output commands relative to the total number
  of commands in the search
- excessive on-line output of large sets relative to total
  search time or number of commands

C. Erratic Behavior

1. REPEATING

- exact repetition of a command sequentially or with
  intervening commands
- exact repetition of a series or group of commands
- repetition of a SELECT or COMBINE command in that arguments
  reduce to the same normal form

2. EMPTY-SET GENERATION & USE

- empty sets generated mainly by one type of command
- empty sets used in arguments of subsequent commands

3. REGRESSION

- viewing records twice with TYPE, but using a less
  informative format the second time
- viewing sets created much earlier in the search
  (no continuity with current search activity)

4. THRASHING

- changing clusters too often
- erratic group and string patterns
- creating many non-empty sets, but never COMBINing
  or outputting them
- using complex logical arguments rather than
  equivalent simple ones
- carrying out separate searches that are never
  COMBINEd and thus never converge

43

5. DWELLING

- COMBINing a small subset of all sets created in
  a variety of ways
- COMBINEs resulting in ever larger sets with no
  convergence
- overly refining a search with alternate logical
  expressions

When the context string begins with the code "acon", (the new context in rules presented in Section 4.9), control is transferred to the section of the rules which evaluates diagnostic thresholds. This evaluation is selectively based upon the last command issued. For example, evaluation of excessive use of facilities will be done on the last type of command issued only; evaluation of failure to use facilities in the general sense will be done on all types of commands except the last issued. This selective application of threshold tests is executed by matches on the command-type code in the context-string.

The threshold analyses themselves will consist of comparisons with formulas determined by the experimental group to represent the tolerable limits of an effective search. Comparison with these limits is done in special-actions. If the conditions of the present search are within the specified limits, the current context string will remain as it is to represent a pass, conditional on a pass of all other thresholds. If all thresholds are passed, the first four positions of the context-string will be changed to "fcon" indicating that control is to be transferred to the mode-oriented-instructions (see Section 4.13). If the conditions of the present search exceed the specified limits, the current context string will be modified to represent a failure condition. The modification will indicate explicitly which threshold test was failed by changing positions 3 and 4 of the context-string to a code representing the error condition, e.g., "ac63...".

4.11. Detailed Analysis and Discussion of Search Problem

When a strategic problem becomes evident by the search's failure to pass a threshold, further analysis and discussion of the problem with the user is necessary. The details of these analyses are spelled out in Section 6. If micro-analyses are done by the "scheduling" approach (see Section 4.7), all pending analyses will be executed. If done by the "direct" approach, those micro-analyses which produce information pertinent to the current problem are called for and executed, if they were not recently executed in the delay-time processing.

When program control reaches this section of the rules, the context-string includes the following information: a code for the threshold which was not passed, the last command issued, and the current mode and submode. The appropriate micro-analyses will be called by the first rule with a matching context-string. These micro-analyses each involve a set of rules which are distinguished largely by protocol numbers. The rules call for paragraphs from "messen" or other special-actions that format statements to be sent to the user. These paragraphs and statements will point out specifics on the search strategy problem to the user.

The user may or may not be asked to respond to the paragraph or message. Required responses may be either code-selections from a menu or free-text. If responses are selections from a menu, they can be matched as alternate match-strings in the rules, much like responses were matched in the /HELP section of the rule table. Free text will simply be recorded in the log. This process may be iterated. Eventually the conversation will be terminated by IIDA. The context string will then be set to "fcon+", transferring control to the mode-oriented instructions.

The entire conversation will be recorded by the log kept on the IIDA search session. This log is made available by simply enabling the Multics audit facility before the session begins. A code indicating all IIDA-to-user messages in this section may also be stored in the help history data structure, as well as any user-to-IIDA menu-selections or statements received in response. If the same assistance must be given twice, it can be stated in a more forceful way the second time since a special-action in the series of rules can search the data structure for earlier in-depth analyses caught by the same threshold analysis.

4.12 Processing the END Command

If the user enters "END" and confirms his wish to end the search (See Section 4.4), the command will be sent to DIALOG and the DIALOG response will be relayed to the user. IIDA will have intervened and asked the user by a menu display: (1) whether he wishes to do another search using IIDA, (2) whether he is altogether finished with his present search, or (3) whether he would like to save his present search and be able to return to it later. In all three cases, the student data structures are written to a disk file for later research and analysis, and, if the user chose the third option, for later restoration as well.

If the user chose to do another search using IIDA, he is again offered at this point, a menu of the four modes in which IIDA searching can be done. After he chooses the mode, IIDA will reinitialize its counters, etc., and begin cycling in the chosen mode. If the user indicates that he is altogether through, IIDA logs out of DIALOG and exits to the MULTICS monitor level. From here the user is expected to log out of MULTICS. If the user indicates that he wishes to pick up his search at a later time, IIDA will mark the output file to this effect, log out of DIALOG and exit to the MULTICS monitor level. When the user logs into IIDA the next time, IIDA will check if a continuation search is pending for him. If so, IIDA will restore all data structures internally. All valid DIALOG commands are reissued in order to reestablish the DIALOG search files. Searching then can commence where it left off in the previous session.

4.13. Mode-Oriented Front-End Instruction

The input stream from the user at this time may contain either commands which were entered as part of a stack of commands or the terminating substring "-..". The first rules in this section determine which of these is in the input stream. If commands remain in the input stream, control transfers to evaluate if the next command is "/HELP". That is, the context string is changed to "u" to possibly match the first rules discussed in Sections 4.3 and 4.4. If the input stream contains only the terminating substring "-..", mode-specific instruction is necessary to cue the user for the input of commands. In this latter case only, the following discussion applies.

For the three exercise modes, the function of the rules in this section is as follows: (1) inform the user as to which command or commands may be entered next, (2) increment the submode code in the context-string if the previous command was without error and is a new command to the current submode, and (3) keep the current submode code in the context string if the previous command entered involved an error or is not new to the current submode. Errors in the previous command are identified by positions 3 and 4 of the context string being anything other than the two letters "on".

The information provided by exercise mode 1, the "canned search", is more extensive than that provided by the other exercise modes because it intends to teach from the example of the very particular commands which it allows. This teaching involves rule-based calls for messages from the table "messen", which have been exemplified several times already. These messages both explain the results of the previous command and provide motivation for entering the next command. If the previous command involved an error, a rule in this section will re-request the command in another way. That is, the discussion the second time around will try to provide a different perspective on the command to insure that it is entered correctly.

The information provided by exercise mode 2, "the subset of the full command set", should be sufficiently terse that it can be included in the user-message part of the rule. When the rule's submode and mode codes in the context-string are matched, the rule's user-message is sent to the user. If the previous command was without error, the submode is incremented by the new context. Otherwise, it is kept the same as the old context. See Table 4.13.1, for an example of how this might be done for exercise mode 2. Note how the submode is advanced only if the command entered was correct and was a new command for the submode.

The information provided by exercise mode 3, the "full command set", is much like exercise mode 2 in terseness. It differs, however, in that it will require the user to explicitly attempt every type of DIALOG command being analyzed by IIDA. The rule controls are similar to those shown in Table 4.13.1, although they may be more involved.

For the assistance mode, no mode-control front-end is needed other than a rule something like the following for changing the context string:

```
rul /fc----a/:.//ucon:::://
```

/fcon-sabe/:.,,Enter SELECT, EXPAND, or /HELP,,//ucon:sb//..sendu..

/fc---sa/:.,,Enter BEGIN or /HELP,,//ucon+//..sendu..

/fcon-sbs/:.,,Enter SELECT, EXPAND, or /HELP,,//ucon:sc//..sendu..

/fc---sb/:.,,Enter SELECT, EXPAND, or /HELP,,//ucon+//..sendu..

/fcon-scs/:.,,Enter SELECT, EXPAND, or /HELP,,//ucon:sd//..sendu..

/fc---sc/:.,,Enter SELECT, EXPAND, or /HELP,,//ucon+//..sendu..

/fcon-sdc/:.,,Enter SELECT, EXPAND, COMBINE, or /HELP,,//ucon:se//..sendu..

/fc---sd/:.,,Enter SELECT, EXPAND, COMBINE, or /HELP,,//ucon+//..sendu..

/fcon-seot/:.,,Enter COMBINE, TYPE, or /HELP,,//ucon:sf//..sendu.

/fc---se/:.,,Enter COMBINE, TYPE, or /HELP,,//ucon+//..sendu..

/fcon-sf/:.,,Enter any command you have learned,,//ucon+//..sendu.,

Table 4.13.1. Front-end instructions: These examples are possibilities for the second, that is, the subset, exercise mode. The submode is incremented whenever a new command allowed by the submode is first used. The submode, though, is not incremented if the previous command involved an error or problem or if the given command was already entered once within the submode.

4.14. Proctor Access to the User and the Search

There will be times when personnel on the IIDA project will want to interact with the user as he is searching and extract information about his search. In this proctor mode, it will be possible to monitor a user's search, make suggestions to the user, receive remarks from the user, and review a summary of the user's search — all while the search itself is in progress.

The first two of these functions can be handled from the MULTICS monitor level by the MULTICS audit facility and the MULTICS command "send_message". The remaining two functions must be programmed. Receiving remarks from the user will be part of the IIDA-user system. Reviewing a summary of the user's search will be a separate job initiated by the proctor.

The MULTICS audit facility makes it possible to capture on a file all messages that enter and leave a system. For the IIDA system this includes all commands and responses from the user, all responses from DIALOG, and all IIDA-initiated messages. This can be reviewed by another MULTICS user, in this case the proctor, at another terminal. The review can be simultaneous with the IIDA user's activity. Thus the proctor can essentially review the search in real-time.

The MULTICS monitor command "send_message" enables the proctor to send a message to the user at his terminal. Switches can be set so that the message appears after the user's current i/o operation has been completed with an end-of-line code. The user thus will not be interrupted as he is entering a command.

The IIDA system will provide the user with the option of entering a special command to send a comment to the proctor. This command may be some variation of CONIT's "comment" command, perhaps renamed "/COMMENT" or "/SEND" to conform to the syntax of IIDA-directed as opposed to DIALOG-directed commands. Such a comment will appear in the real-time activity log if the proctor is using the audit facility and will appear immediately at the proctor's terminal if it is set to "accept_messages".

Reviewing a summary of the user's search requires a distinct program called by the proctor. A switch can be set in the main IIDA program to write the user's data structures out to a file periodically during the search. The proctor's program will read this file and format the data at the proctor's terminal for him to review. Access to this intermediate file will be provided to both the main IIDA program and the proctor's program.

5. Data Structure

5.1 Overview

The search history data base contains extracted and computed values characterizing a single user's interaction with the DIALOG bibliographic data base service. These data represent a search session from the time the user begins using IIDA to system logoff.

User input, data base responses, and IIDA analyses of the search based on evaluation of input patterns and DIALOG feedback constitute the structure. These data are gathered and stored primarily to provide the diagnostic programs with information it uses to study the search in progress.

IIDA is concerned about the whole search comprised of interrelated commands. The history files make it possible for the diagnostic programs to determine how commands are related as it views the search retrospectively from the current entry to the first command. Actual behavior can thus be compared with ideal behavior. Command sequences are examined for common procedural deviations.

IIDA's primary interest with a single command entry is its eventual transmission in a syntactically correct form to the search system. If the parser recognizes syntactic errors, the command is labeled erroneous, the error pinpointed if possible, the data base information recorded, and the user informed and given the opportunity to reenter the command.

Large segments of the data base are updated when a single correct command is entered. Depending on the type and format of the command, various reduction, expansion, or look-up routines are initiated or scheduled. By the time a command has been transmitted to DIALOG, IIDA has already noted and posted the command text, type, and class; given it a string assignment; normalized it if necessary; and separated it into command related components. Expansion routines have been scheduled or completed, and the response parser has been prepared to receive a particular response format.

When a control command is recognized, additional file values are updated. When thresholds are crossed or in-depth analyses initiated, other elements are entered in the data base.

The details and coordination of the data recording techniques are discussed in section 5.3.

While the diagnostic programs will address the student data structures most frequently, the files (or subsets thereof) are available to both the searcher and proctor. The searcher may wish to recapitulate his search periodically to assess current progress or to assist in formulating further strategy. The arrangement of the data files (see section 5.2) makes it possible for him to view this information from a variety of perspectives. He may be interested in the sets generated, the commands entered, or the records viewed. Using options in /HELP he may request displays of data covering these topics.

The proctor has access to all history data while the search is in progress. Copies of the data files are available in a common area, and current input may be transmitted directly to the proctor's station. A report formatting routine, described in 5.4, organizes, labels, and displays the history data for both the proctor and searcher.

At the conclusion of the search the data structure will contain information of great value in studying long-term search behavior. After the user has logged off, the files are edited and copied to a more permanent storage location. Data can then be analyzed across searchers or over time for a given searcher. The accumulation of detailed data on search behavior will contribute to future refinement of IIDA diagnostic routines.

5.2 Data Structure Contents

The data structure comprises six distinct but related categories. The sub-structures are referred to as history files since each chronicles the progress of a specific aspect of the search. The six categories are: commands used, sets generated, records and sets viewed, errors made, /HELP called, and descriptors used.

Five files reside in memory and are logically interconnected through the use of pointers. The sixth data set is an indexed sequential file. The structure as a whole is organized so that elements are stored efficiently and are easily accessible. The indexed file arrangement, which permits dynamic growth, is used for the descriptors file. A computed abbreviation of the descriptor becomes the key to the file. Each descriptor is thus entered only once, and various details are updated with each usage.

The six files contain values needed by the diagnostic routines. These elements can also be viewed by the searcher and proctor. The files contain the following specific items.

(A) COMMAND HISTORY (contains sequence of <u>all</u> commands entered and encoded descriptive data about each entry

   (1) Command text: user input exactly as entered regardless of classification

   (2) Command class: a parser assigned code classifying the command as valid, erroneous, or control, i.e., an IIDA directive

   (3) Normalized text: a standardized rearrangement of the command to enable comparisons with other commands; especially useful in determining whether commands in different formats mean precisely the same thing

   (4) String type: a parser assigned value based on the type of command; used in grouping routine; differs from command type (6) in that various commands and command formats can be assigned a similar string type depending on function

IIDA SEARCH HISTORY DATA BASE

File	Element	Source
COMMAND HISTORY	(1) Command text	User input
	(2) Command class (valid, error, control)	Parser
	(3) Normalized text	Normalization routine
	(4) String type	String Assignment
	(5) Group assignment	Grouping routine
	(6) Command type	Parser
SET HISTORY	(1) Set number	Response parser
	(2) Set size	Response parser
	(3) Zero set flag	Zero set check routine
	(4) Type of command creating set	Parser
	(5) Number of command creating set	Parser
	(6) Number of times set referenced	Parser
	(7) Expansion of set descriptors	Descriptor expansion routine
	(8) Normalized descriptors	Normalization routine
SETS & RECORDS VIEWED	(1) Set number	Parser
	(2) Range of records viewed	Parser
	(3) Format of records	Parser
	(4) Accession numbers	Response parser
	(5) Position within set	Parser
	(6) Relevance assigned	User input
ERROR HISTORY	(1) Error text	User input (identified by parser)
	(2) Command type	Parser
	(3) Error type	Parser

Figure 5.2

51

File	Element	Source
/HELP HISTORY	(1) Count of types of /HELP called	/HELP routine
	(2) Location of /HELP call	/HELP routine
	(3) Time data	/HELP routine
DESCRIPTORS USED	(1) Descriptor text	Parser or set descriptor expansion or unstacking routines depending on format of argument
	(2) Original argument	Parser
	(3) Number of postings	Response parser
	(4) Count of times used as argument of S,E,R, or C	Response parser
	(5) Number of related terms	Response parser or EXPAND expansion routine
	(6) Flag of descriptors seen in RELATE or EXPAND tables	Response parser

Figure 5.2 Continued

(5) Group assignment: code given to a command during the grouping routine; indicates command's relationship to other commands

(6) Command type: unique value for each possible command

(B) SET HISTORY (indexed by set numbers assigned by DIALOG)

(1) Set number: index to file corresponding to DIALOG assigned set number which has been extracted by IIDA in response to a set-generating command

(2) Set size: number of items in set, computed by DIALOG and extracted by response parser

(3) Type of command creating set: equivalent to (A.6)

(4) Number of command creating set: pointer to entry in command history

(5) Number of times set referenced: initialized zero and incremented every time set number referenced by in any command

(6) Expansion of set descriptors: particularly useful in regenerating the components of a COMBINE; set numbers are translated into descriptors

(7) Normalized descriptors: above descriptors rearranged in standard format to facilitate comparisons

(C) SETS AND RECORD VIEWED

(1) Set number: pointer to set from which records are viewed

(2) Range of records viewed: sequence number of records within a set examined during a single viewing.

(3) Format of records viewed: code indicating the format in which records were seen

(4) Accession numbers: DIALOG assigned identification numbers of the records viewed

(5) Relevance assigned: rating given to the record by the searcher for that viewing

(D) ERROR HISTORY

(1) Error text: pointer to erroneous entry in the COMMAND HISTORY file

(2) Command type: pointer to command type entry (if identifiable) in the COMMAND HISTORY file

(3) Error type: parser assigned error classification code

(E) /HELP HISTORY

(1) Count of time /HELP called: a counter for each type of /HELP is incremented every time a control routine is called by the searcher

(2) Location of /HELP call: pointer to preceding valid command to place call in the context of the search

(3) Time data: time spent in /HELP defined by entry and exit times

(F) DESCRIPTORS USED

(1) Descriptor text: term is transformed to a key and the descriptor entered on the file; this standard entry will be used for various formats representing the same value.

(2) Original argument: may be the same text as above or a variant or abbreviated format, such as an E-series argument or truncated version

(3) Number of postings: number of items in bibliographic data base indexed by the descriptor

(4) Count of times descriptor used as argument of SELECT, EXPAND, "RELATE" or COMBINE: counter incremented depending on command type associated with argument

(5) Number of related terms: count of related terms extracted from EXPAND display

(6) Flag of descriptors seen in "RELATE" or EXPAND tables: bit set to indicate that term was displayed in table to searcher

5.3 Data Recording

5.3.1 /HELP Routine

When the searcher asks for assistance, the /HELP routine is invoked. The data base update segment adds four values to the HELP HISTORY table in the student history file: the time /HELP is entered, the time at which control returns to the main program, a pointer to the previous command, and the specific type of IIDA assistance required. These data are needed by the /HELP diagnostic routine in analyzing IIDA assistance.

54

DIAGNOSTIC SUPPORT ROUTINES

TYPE	ROUTINE	FUNCTION
MICROANALYSIS	(1) Command parser	Establishes syntactical correctness of input line; records values in data base; calls string assignment, normalization, and expansion routines where appropriate
	(2) Cycle detector	Evaluates series of strings to determine relationship to other strings; assigns cycle number
	(3) Expanded Index Viewing Status	Determines format of descriptor; invokes expansion routines if needed; checks DESCRIPTOR HISTORY for viewing status
	(4) Response Parser	Extracts field values and posts to history files
	(5) Thesaurus Look-up	Issues EXPAND arg to DIALOG; performs root word check and returns results to calling analysis routine
	(6) Long-term Data Update	Computes and stores values for long-term search record
EXPANSION	(1) COMBINE Expander	Translates set numbers or ranges into descriptors
	(2) E- & R- Expander	Replaces E- and R- identifiers with corresponding descriptors
	(3) Normalization	Transforms command input to standard form
UTILITY	(1) String assignment	On basis of command and argument type assigns the same or next string number to entry; increments counters; computes averages
	(2) Unstacker	Extracts single commands from multiple command entries

Figure 5.3   55

5.3.2 Microanalytic Routines

(1) Command Parser

The parser examines every non-control entry checking syntax, standardizing format, and updating the history files. After the routine isolates and identifies the command, the parser evaluates the argument. Each command type has one or more acceptable argument formats. The recognition segment of the parser addresses the problem of designating categories to entries depending on various command/argument combinations.

If either the command or argument format is erroneous, the parser calls the update routine which posts relevant data to the COMMAND and ERROR HISTORY files.

If the entry is deemed acceptable, the update routine records the entry as well as various assigned and computed values to the COMMAND HISTORY table. Values originating from the parser or parser-invoked routines include: command, text, command classification and type, and time data.

The parser also sets flags based on argument format scheduling routines to be run at a later time.

If the entry is a series of stacked commands, the parser calls an unstacking routine which isolates the commands. The parser then processes each command separately. Each individual command will have its own data base entry, and a bit will be set to identify it as part of a stacked command sequence.

(2) Response Parser

The command parser sets an indicator which tells the response parser what to expect from DIALOG in response to the command issued. When a message is received, this routine first checks for DIALOG or network messages. If a problem is identified, the user is informed of procedures to be followed.

If no errors or problems are detected, the response parser accepts the return line or table, stripping off headings and capturing significant values. Some of the data are stored temporarily while others are posted to the history files.

Update routines are invoked according to the type of command issued. Set generating commands cause fields in the SET HISTORY and DESCRIPTORS USED files to be updated. EXPAND or "RELATE" commands produce values stored in temporary tables and posted to the DESCRIPTORS USED file. Print generating commands generate data to be recorded in the SETS & RECORDS VIEWED file.

(3) Cycle Detector

The cycle detector is called after the string type has been assigned. If the current string type is not less than the previous string type value, the command is considered within the same cycle and assigned the same group number. If less, the group counter is incremented, and a bit set to note a group change. The group assignment is posted to the COMMAND HISTORY table.

5.3.3 Expansion Routines

(1) E- and R- Expander

This routine is called from the input parser when the command is recognized as EXPAND or SELECT and the argument is identified as E- or R- series. The parser checks to see if requisite flags have been set indicating that a table has been built by a previous command. If not, an error is transmitted to the user. Otherwise, the value is looked up in the stored table captured by the response parser. Each expanded argument is posted to the DESCRIPTORS USED history noting that the original input is in E- or R- format. Each expanded command is listed in the COMMAND HISTORY file with a bit set to show that the argument was abbreviated.

(2) Combine Expander

The parser anticipates that the argument of a COMBINE command will contain set numbers connected by the Boolean "AND", "NOT", or "OR" operators with complex combinations embedded in parentheses. Alternatively the abbreviated form may be used when the sets to be combined are sequentially numbered and connected by the same operator. The expansion routine addresses the descriptor field of the SET HISTORY table using the set number as an index. The referenced descriptors are combined with the Boolean operators in the same order as input, recorded in the SET HISTORY file, and passed to the normalization routine.

(3) Normalization Routine

The function of the normalization procedure is to standardize the arrangement of arguments of the expanded COMBINE command. Section 4.7.3 discusses the steps followed to produce normalized arguments. The output is posted to the normal format field of the SET HISTORY table.

(4) Expanded Index Routine

In response to an EXPAND command the system will return a table to be displayed to the user and stored by the response parser. This table is scanned and descriptors posted to the DESCRIPTORS HISTORY file. A bit is set to indicate that the descriptor was seen in an expanded display.

"RELATE" (i.e., EXPAND E-series) data are recorded similarly. A flag previously set will indicate a second EXPAND has been entered. A bit is set to show that related terms were seen in tabular format by the user.

5.3.4 Utility Routines

(1) String Assignment

The parser calls the string assignment special action for every valid command. The combination of command type and argument format determines the string type to be assigned. If the current string type is the same as the

previous string type, the command is added as a member of that string. Otherwise the string counter is incremented; the current command becomes the first entry in that string. The string assignment is recorded in the COMMAND HISTORY table. When the string type changes, the average number of string members is computed.

(2) Unstacker

The unstacking sub-routine is called when a multiple command line is encountered by the parser as a result of scanning the entry for one or more semicolons. If the value of the position of a semicolon is not equal to zero, i.e., there is a semicolon in the string, the unstacking routine sub-strings the command between delimiters. The procedure continues until the end of the string is reached. As each command is moved to a separate temporary storage location, a command counter is incremented. This counter informs the parser how many individual commands must be processed.

Parsing then proceeds command by command interacting with the user whenever erroneous syntax is encountered. Data related to each command is recorded in the history files. A bit is set to indicate that the command was input in stacked format.

Multiple command lines are entered in the student data base as separate commands since information related to each component may be needed for diagnostic purposes. The original command line will be transmitted to DIALOG if all parts are correct; otherwise separate commands will be issued.

5.4   Report Generation

Both the searcher and proctor have access to values in the performance data structure. The searcher is offered formatted sections of the structure as /HELP options. The proctor can request detailed and summary information on the currnet search by running the proctor report program.

The report generation programs address selected fields of the data structure, perform some computations, format a print file, and display the report to the searcher or proctor. The searcher and proctor reports contain many of the same fields. Summarizing the search for the user is intended to assist him in evaluating progress and formulating further strategy. The reports displayed to the proctor give the same basic information plus more detailed diagnostic data.

Reports can be generated to obtain information on commands issued, sets generated, descriptors used, records viewed, and errors made. The proctor can request reports on the usage of /HELP and performance analysis routines.

The proctor can also view the search in progress. Current inputs and data base responses can be displayed at the proctor's station for real time monitoring of the search.

5.5 Long Term Data Storage

The command history file will be retained at the conclusion of the search. Header data will precede search details. The file will be relegated to off-line storage. The search may be rerun by submitting the command list to restructure the details of the search. A feature of this regeneration program is the ability to interrupt the search at any point and force a listing of values in the data structure.

6. Diagnostic Programs

The diagnostic programs are a set of routines used to analyze dynamically the search in progress. The programs are called when certain qualifying conditions are met. Various routines are invoked for:

- every command input
- specific command types and formats
- DIALOG responses to specific command types
- changes in command type
- content of command arguments vis a vis previous arguments

The main function of the diagnostic programs is to evaluate the search by examining and comparing command entries to detect significant behavior patterns and to alert the searcher when IIDA perceives a problem. Additionally there are some routines which restructure or redefine the data to support the analysis programs. The diagnostic routines can be classified according to function.

The programs referenced in Section 5.3 perform primarily a supporting role. The expansion routines (COMBINE and E- and R-extenders) translate abbreviations or alternate versions of arguments into a more meaningful format for viewing or comparison. Set numbers are transformed to descriptor combinations, and E- and R- series references become single or ranges of terms. The string and cycle assignment routines assign codes to each entry which are needed for more detailed threshold and in-depth analyses.

The normalization routine standardizes the elements of the input line. The sole purpose of these routines is to enable the performance analysis programs to function more effectively. In themselves, these support routines are of little value. The parser, of course, is useful in providing the user with feedback on erroneous entries; but in the process of scanning an entry, it also updates history elements which are essential for procedural and strategic analysis. The response parser also provides important data for the analysis routines. The parser level of analysis is concerned mainly with single inputs and responses.

Performance analysis based on the whole search is the main emphasis of IIDA. The next level of diagnostic programs endeavors to perform the detailed analyses of various search aspects. Performance analysis is a two step process. Preliminary threshold checks are made followed by in-depth analyses.

The threshold checks examine selected data in the history files, perform some computations, and evaluate the results against experimentally predetermined threshold values. The type of command determines which checks are to be made. Some of the aspects evaluated are:

(1) /HELP usage
(2) Errors made
(3) Sets created with zero postings
(4) Duplicate commands entered
(5) Time usage
(6) String and cycle analysis

BEHAVIOR TYPE	BASIS	PROGRAM	ACTION (USER)
Syntactic Error			
Invalid command	Initial letter of command and specific characters found in user command	Parser 4.6	Inform user than command is incorrect; tell how entry was interpreted; elicit new command
Invalid argument	All of message less command	Parser 4.6	"
Invalid set number	Command type (TYPE/ COMBINE/LIMIT); command text; current index to SET HISTORY table	Parser 4.6	Inform user that set number is is out of range; elicit new command
Excess errors	Error code in COMMAND HISTORY table; total error count and error by type count	Error Analysis 6.1	If threshold of same types exceeded, inform user; if threshold of total errors exceeded, offer exercise mode or HELP option

Figure 6.1 Diagnostic Routines

58

61

62

DIAGNOSTIC ROUTINES

BEHAVIOR TYPE	BASIS	PROGRAM	ACTION (USER)
Poor Procedure			
Excessive sets with zero postings without EXPAND commands issued	Zero set threshold exceeded; check whether argument ever seen on EXPAND table	Zero Set Analysis 6.2	Issue thesaurus look-up; scan table for similar terms; suggest use of EXPAND
Repeated commands	Command text; check all normalized commands issued for duplicate entries	Duplicate Command Analysis 6.3	Comment on redundancy; suggest that searcher plan ahead; review "review" facilities
Uninformative format	Command text of type/display format field	Parser 4.6	Inform user that output is not is not particularly helpful; describe HELP facilities
Series of wrong answers	IIDA-User dialog; response required other than new command entry	Parser (following IIDA cue) 4.6	Point out what's happening and recommend alternate answer
Non-use of thesaurus	Check set history for # of non-null sets; check command history for non-use of EXPAND	Expand Usage Analysis 6.4	Issue thesaurus look-up checking for similar terms; present to searcher possibility that similar and related terms may exist in data base; suggest possible use of EXPAND
Excessive time (between commands)	Time threshold #1: computed time between command entries	Time Analysis 6.5.1	Inform user of excessive time usage; query regarding pacing
Excessive total time	Time threshold #2: computed total time	Time Analysis 6.5.2	Inform user; ask if assistance needed; offer in form of HELP or proctor

59

BEHAVIOR TYPE	BASIS	PROGRAM	ACTION (USER)
Poor Strategy			
Unreferenced non-null sets	Set used flag equals zero after several intervening commands (beginning with the third group	Set Usage Analysis 6.6	Inform user that set has never been referenced
Excessive string length	Current string length and average string length	String Analysis 6.7	See dwelling action.
Excessive cycle length	Current cycle length and average cycle length	Cycle Analysis 6.8	"
Dwelling	Length of current COMBINE-type string; referent sets of C-type commands; set sizes	Dwelling Analysis 6.9	Inform user of dwelling status; ask objective and how far from it; offer HELP, review of strategy, review of SET HISTORY (A single command may trigger a number of analysis routines; no more than one such action will be initiated for a command.)
Thrashing	Length of C- string; number of cycles; referent set or commands	Thrashing Analysis 6.10	Inform user; prompt user: (1) which set is closest to objective, (2) how can it be changed; if set in recent group suggest work on it; offer HELP, review of strategy, review of sets

Figure 6.1 Diagnostic Routines (Continued)

(7) Extraneous commands
(8) Dwelling
(9) Thrashing

When a threshold is crossed, an in-depth analysis is undertaken. There are various initiating conditions for each of the detailed analyses, each corresponding to a specific problem or group of causal factors. In some instances, the in-depth programs must perform further analyses to pin-point problem areas. After the more detailed evaluation is done, IIDA converses with the searcher. This interaction is intended to inform the user of the system's diagnosis of the state of the search. One of a series of messages is displayed guiding the user to consider alternate strategies or usage of facilities.

The main categories of threshold and in-depth analyses are discussed in section 6.1. Two support programs not referenced in section 5.3 (since their utilization does not contribute data to the history structure) are described in section 6.2.

6.1 Performance Analysis Programs

6.1.1 Parser

The parser systematically studies every non-control command for acceptable command and argument syntactic structure. The parser may call other micro-analytic or support routines as a result of this examination. It may also interact with the searcher when errors are encountered. Specific diagnosis of errors is a parser objective. Invalid commands and arguments are identified, as are context errors, such as invalid set numbers or uninformative printing formats. The parser is described in greater detail in section 2.4.6.

6.1.2 Error Analyses

Thresholds: count of total errors and errors by type.

This routine is called by the parser following the detection and classification of a syntactic error. Two counters are incremented; the first counts errors according to type, and the other accumulates total errors. These counts are matched against the threshold values. Exceeding the error type threshold indicates that the searcher has committed this error repeatedly; he has thus received IIDA messages diagnosing the error on several occasions. When the threshold is crossed, a more comprehensive message is sent to the user; a detailed explanation is given, and specific procedures are suggested for avoiding the problem in the future.

If the total error count is excessively high, the searcher may lack knowledge of the fundamentals of command construction and usage. Crossing this threshold results in a message to the user offering help in the form of strategy review or assistance in the form of the exercise mode.

6.1.3 Zero Set Analysis

Threshold: total number of successive zero sets.

The zero set check is run after the response parser has evaluated the response to a SELECT command. If the newly created set has zero postings, the zero set counter is incremented. This new counter value is compared with the threshold. If many zero sets have been produced during the search, the argument of the SELECT will be looked up in the DESCRIPTORS USED file to determine whether the term was ever seen in an EXPAND or "RELATE" table. If the argument has appeared in an EXPAND table, IIDA reminds the searcher that the display viewed previously revealed no documents indexed by that descriptor. If EXPAND was never used, a thesaurus look-up will be issued by IIDA. The table will not be displayed to the user, but the descriptors will be scanned for terms similar to the argument. If like descriptors exist in the file, the use of EXPAND will be recommended to the searcher.

6.1.4 /HELP and Time Analysis

Thresholds: time in /HELP
total time in search
time between commands

Time recorded at various intervals is used to measure system as well as search performance. Time data are collected when the following events occur:

Event	Measures
Log In	Search session start time
Command Entered	Time between command entries
/HELP (In & Out)	Utilization of IIDA assistance
Message to DIALOG (Out & In)	Host and network response time
Log Out	Length of session

The searcher's use of time may reflect progress or problems. Three time elements, for which thresholds are checked, focus on performance.

When the user calls for assistance by entering a control command, the time is noted. When the searcher exits the /HELP routine, the time spent in that mode is computed. Total HELP time as a percentage of total search time is compared with a threshold value. Exceeding this limit may cause IIDA to encourage the user to return to exercise mode or prompt the proctor to intervene.

Time between commands as well as total search time is computed. If either time is excessively long, IIDA will query the searcher regarding problems or pacing. If assistance is needed, the /HELP menu will be offered or the proctor connection enabled.

6.1.5 Duplicate Command Analysis

The identification of command repetition by means of two levels of comparisons is the function of this routine. Every valid entry is transformed to a normalized format. The most recent entry is then compared with all preceding commands in original as well as normalized form.

An original entry match means that precisely the same line has already been submitted. Two equivalent lines will be detected by comparing normalized forms. In this way COMBINE 1 * 2 will be recognized as meaning the same as COMBINE 2 * 1.

When redundancy is observed, IIDA will mention to the searcher that he has entered the same command previously in either the same or different formats. The user message will include a survey of review options and the recommendations that the searcher plan his strategy in advance as much as possible.

6.1.6 Expand Usage Analysis

Threshold: number of SELECTs with no preceding EXPANDs

IIDA checks the descriptor history file to see if an EXPAND argument has previously entered whenever a SELECT argument generates few postings. If the online thesaurus has not been addressed, IIDA calls the thesaurus look-up routine which computes the similarity of the argument with nearby terms in the thesaurus. The function of this check is to give the searcher information regarding the presence of like or related terms which may be used as more productive search arguments. This routine is similar to the zero set check.

6.1.7 Set Usage Analysis

Threshold: number of intervening commands

This special action periodically reviews the set usage field of the SET HISTORY file checking for sets never referenced. The difference between the current command number and the command number of the unreferenced set is compared with the intervening command threshold. If the critical value is exceeded, the searcher will be reminded that he has created but never used that particular set. The set usage check will not be invoked until the second group has been processed.

6.1.8 String Analysis

Threshold: absolute string length
            average string length

The string length counter is incremented whenever a command is the same type as its predecessor. The counter is compared with the threshold for the

current command type. If the threshold is exceeded, a command-related message is displayed to the user. For example, if many SELECTS have been entered, the user will be informed that a COMBINE command might appropriately refine the topic. Such a message will not only alert the user to a potential problem but offer him a possible plan of action.

6.1.9 Cycle Analysis

Threshold: absolute cycle length
            average cycle length

A cycle or group is composed of strings which are composed of commands of the same type. To be included in a group, the current string type of the command must be greater than the string type of the previous command. The cycle check increments a value counting number of commands in the cycle. If the counter becomes greater than the threshold value, IIDA sends the searcher a message to help him modify his strategy.

6.1.10 Thrashing Analysis

Threshold: number of cycles

Where sets are never COMBINEd or group assignments are frequently changing, there may be evidence of a phenomenon known as thrashing. The thrashing check evaluates the number of group changes against a critical ratio. If the number of cycles per total number of commands is excessively high, an in-depth analysis of search status is undertaken. The purpose of the detailed evaluation is the reduction of the number of possible causal factors so that the user can be accurately informed of problems. IIDA can view programmatically:

—— sets that have never been referenced.

——— sets that have never been COMBINEd or seen

——— the use of complex rather than equivalent simple
      arguments

——— clusters that change frequently indicating a change
      of topic or separate searches (It is undecided whether
      the clustering evaluation will be implemented in this
      version of IIDA. In the event that clustering based
      on term similarity is deemed a significant measure, a
      second thrashing check will be built into the system.)

——— scattered rather than systematic references to previously
      created sets

When enough thrashing has occurred to be noticed, the user may find intervention helpful. A review of strategy and set history may be useful. IIDA

may prompt the user to decide which set most closely resembles the desired
output and to consider what can be done with the set to improve it.

Further work is being done to refine the methods of diagnosing thrashing.

6.1.11   Dwelling Analysis

Threshold: length of C-type string

Dwelling or overly refining a search is recognized when a few sets are
COMBINEd in many ways or by an increase in the size of sets as a result of
COMBINEs. The dwelling analysis is initiated when the C-string length is
greater than the expected length. The routine looks at the referent sets and
ascertains whether essentially the same ones have been addressed, perhaps in
variant expressions. A further look at the set size assists IIDA in judging
whether the search is not converging but reworking the same concepts.

When IIDA perceives dwelling status, interaction with the user may be
justified. The objective is to encourage the searcher to understand what he
has done by offering him a strategy review and then to prod him to make
decisions which will bring his session to a meaningful conclusion.

The user will be urged to look over his sets (defined in the review as
combinations of descriptors), to reevaluate his search objectives, and to
estimate how far the generated sets are from his goals. He might be en-
couraged to view a few records if he has not already done so.

(A second dwelling check related to similarity clustering had originally
been proposed as part of the IIDA system design. Dwelling status would be
established by frequent use of similar descriptors in COMBINE commands. This
analysis may be included in future versions.)

The evaluation of the dwelling and thrashing thresholds and the subsequent
in-depth analyses of these phenomenon are subject to modification.

6.2   Diagnostic Support Routines

6.2.1   Expanded Index Viewing Status

This program determines whether a descriptor has been seen in an EXPAND or
"RELATE" display. The DESCRIPTORS USED file is addressed by key, i.e., the
descriptor. If the term is indexed, it has at least been seen in some form by
the searcher. The descriptor record is looked up; the usage field is examined
to determine whether the term was the argument of an EXPAND or merely seen as
a member of an EXPAND or "RELATE" table.

In some instances, the absence of the term in the index will be the motiv-
ating factor in calling the thesaurus look-up routine. The viewing status
routine is called from the zero set check threshold program.

6.2.2 Thesaurus Look-Up

A misspelled term, a descriptor in the wrong form, or an argument indexed in an unexpected manner in the data base may be SELECTed with unsatisfactory results. If the searcher has not used the EXPAND command, he will have no knowledge of similar but slightly different terms nor of terms related to the SELECTed one.

IIDA invokes this sub-program when it becomes evident that the online thesaurus is not being used and the search is not progressing as well as the searcher would like. An EXPAND term is issued to DIALOG. The resulting table is not displayed to the user but is subsequently examined for root word similarity. The number of related terms is also noted. (Like terms are determined by the percentage of identical letters from the left of the descriptor string. Related terms are identified by a non-zero entry in the related terms column of the table.)

If like or related terms appear to be present, the searcher is urged to enter his own EXPAND commands. Although IIDA could easily inform the user that this step has been done for him and display the output, doing the searcher's work for him would be system-defeating. The objective here is to assist the user in search strategy formulation by increasing his knowledge of DIALOG facilities.

7. Exercises and Assistance Mode

7.1 Purpose of the Exercises

The purpose of exercise mode is to provide the student instruction in the use of DIALOG in the form of exercises controlled to varying degrees by the computer. While they are generally aimed at users who have had some minimal prior training (as little as an hour) it will be possible for a highly motivated student user who is comfortable with computer use to learn DIALOG from scratch through IIDA.

The first exercise is a "canned" search. The user is given the specific commands he is to enter and simply told to type them in at the appropriate times. Each is preceded by a short tutorial message. The basic purposes of the exercise are: (1) to show the student what are the mechanical results of the use of certain commands (i.e., the kinds of responses provided by the DIALOG system) and (2) to familiarize him with the general structure of a search, and (3) to introduce the IIDA HELP command and its use.

The second exercise has as its purpose to familiarize the student with the sequencing of commands, or search strategy, i.e., how to decide what commands to use. The student is controlled to some extent by IIDA, in that he is not free to use any command he wants, at any time, but he has some freedom to carry out the search as he wants. This exercise involves the use of only a limited number of DIALOG commands.

The third exercise introduces some additional commands, beyond what have been used so far, and insures that the student has some experience with their use. These will be introduced in three groups: (1) Variations on the use of the select command (truncation, selecting by line number from an EXPAND display, and use of infixes in text searching), (2) Use of IIDA diagnostics; and (3) Use of advanced techniques such as stacked commands. IIDA will not explicitly teach the use of other advanced commands, such as SAVE.

The fourth "exercise" is assistance mode, in which the student operates on his own without interference from IIDA unless he invokes it or some problem is detected by IIDA diagnostics.

7.2 Exercise 1

At the time this report is written, the design of the exercises lags the design of the other computer programs. Hence, these sections on detailed design of exercises are not as detailed as those on other aspects of program design.

The first exercise will introduce the following commands, and demonstrate to the student the DIALOG response to them:

        BEGIN
        SELECT (single term only, but showing use of prefix and suffix)
        EXPAND and EXPAND following a previous EXPAND to display
          related terms
        COMBINE
        TYPE or DISPLAY
        LOGOFF

Following introduction of these commands, the student will be introduced to the IIDA HELP command and asked to make use of it to retrieve a set history display. He will also be told about how he can use HELP for definitions of commands and to get details of command formats.

The first exercise will have two cycles. During the second cycle, some additional tutorial material will be presented and the student will be encouraged to use HELP as needed or desired. During the first phase, the user is introduced to the basic DIALOG commands (BEGIN, EXPAND, PAGE, SELECT, COMBINE, TYPE and LOGOFF). The EXPAND command is used both for browsing through the index and to identify related search terms. The SELECT command is used to choose two terms, which are then combined using the COMBINE command with the AND operator. The TYPE command is used to examine three citations.

Next the IIDA HELP facility is introduced and the user is instructed to use it to review previously created sets. Upon exiting the IIDA HELP facility, the user is reminded to call the facility whenever needed, during the remainder of the search.

Review of previously created sets provides a logical introduction to the second phase of the search because one of the previous sets is chosen for a refinement of search terms. The commands used in the first phase are used again to conclude the search.

This exercise does not operate in Socratic mode, asking questions and testing answers. The student inputs are commands, exactly as the student was told to enter them, except for HELP, which, after its introduction, may be used whenever desired. Because the exercise takes the student systematically through the basic commands needed to perform a search and then introduces HELP, those students who want to and have the curiosity to do so can "browse" around in this mode and discover detailed explanations for all commands. We expect that those who are computer literate will find it intriguing to operate this way. It is not, however, a basic objective of IIDA to provide this initial instruction. This is a bonus.

At the completion of the first exercise, the student user will have tried all the basic commands and will have seen the kind of display that each generates. He will have been through a complete two-cycle search. He should remember what the basic commands do, but not necessarily perfectly. He should remember how he can get the detailed information he may need to operate on his own, through HELP.

7.3 Exercise 2

The second exercise presents the student with a search problem and suggests that it be approached in a certain way. The student will be led to make use of commands in the proper sequence, i.e. he may be given a choice of whether to SELECT or EXPAND a descriptor, but will not be permitted to try printing a set before he has explored the use of the term selecting and expanding commands. Then, he must make use of COMBINE and then, if he can get a non-null set with few enough records, he may be allowed to display. Within each command type, he can have discretion on which to use, i.e. whether to SELECT first or EXPAND, whether he needs to look for related terms or not.

The statement of the search requirement will include in it terms which should be looked up in a thesaurus. If the student fails to do this, the IIDA system will have available to it information on what might have been found had he done so, and can use this to demonstrate to the student what he might have gained by EXPANDing or EXPAND-EXPANDing.

The student will be required to complete at least two cycles or groups and will be prevented from completing more than four. Since the purpose of this exercise is to begin teaching strategy, we do not want to risk the user becoming bogged down because of poor choice of terms and having, as a result, an unsuccessful and unenlightening search.

7.4 Exercise 3

At this stage, the student will normally have performed a reasonably successful search in exercise 2. (We cannot guarantee this, nor do we intend to force students to go through the exercises in prescribed order; hence the adventurous may reach this stage in a number of ways.) Before letting him go completely on his own, he will be introduced to some more search commands or variations on those he knows. There will be three groups of variations introduced here:

1. SELECT variations, including: truncation, selection of terms by line number from an EXPAND display, and the use of infix notations to perform text searching. He will be given, in the manner of exercise 1, one command of each type to enter and will be reminded of the services of HELP.

2. IIDA diagnostics. Here, the student is given a complete explanation of all facilities available through help, such as review of set history (already done once), definition of commands, browsing through student performance data base records, and moving to other exercises. He will be reminded that these facilities are not generally available through DIALOG.

3. Shorthand notations for DIALOG commands. Here, he will learn about one- and two-letter command abbreviations, the simplified notation for ANDing or ORing several sets (COMBINE 1 #/5/OR) and about stacking commands (entering several at one time, separated by semicolons).

At this stage, the student should be ready to begin a search of his own choosing, without tight IIDA controls.

7.5 Assistance Mode

As we have described elsewhere[1] the purpose of the assistance mode is to enable the student user to make use of DIALOG without interruption from IIDA. However, the student may invoke IIDA assistance, through the HELP command, and IIDA may intrude itself on him when errors are detected. This mode of operation is one in which we detect and correct errors or ineffective behavior; we cannot guarantee to do so and we cannot tell the student exactly what he should do to solve the problem. As a "course," in the CAI sense, the assistance mode is tolerant and flexible. It allows the student to do whatever he wants and calls upon various monitoring subroutines to detect and correct behavior when necessary; it does not start off telling him what to do or how.

7.6 Approach to Exercise Design

Interactive programs such as these can be, indeed must be, designed in the imagination and then made into operating computer programs. Their impact on users can be determined only after trial, and their impact may be different with different groups of users (especially important distinctions are whether or not the user is computer literate, whether or not familiar with the data base being searched, and extent of prior experience in online searching.)

It can be expected, therefore, that the design of these programs will change after initial user testing. Unlike the diagnostic programs, these are relatively simple as computer programs and their change will not impose major delays on the development of the system.

_____

[1] Meadow, Charles T., et al, Final Design Report.

8. Plans for Computer Program Production

Table 8-1 shows graphically the plans for program implementation.
Major program elements are:

1. Exercise Mode Programs — consisting largely of messages to be
   sent to a student and logic for deciding when to send them.

   1.1 Introductory Exercise. This is entirely tutorial and
       is intended as a quick refresher for those having had some
       brief experience or training.

   1.2 Limited Language Exercise. This exercise proposes that the
       student carry out a search on an assigned topic, using a
       limited subset of the complete DIALOG language and a limited
       version of IIDA diagnostics. It is used to acquaint the
       student with the search as a process and with IIDA diagnostics.

   1.3 Additional Commands Exercise. This introduces or reintroduces
       advanced commands to the student. It is tutorial in tone.

2. Assistance Mode. This is the major mode of IIDA and allows the
   student to carry out any search, of his own choosing, using any
   feature of the DIALOG language.

3. Diagnostic Programs

   3.1 Parser. This program analyzes input from either the student,
       the search service or the communications network, tests the
       validity of student inputs, and posts elements of messages
       to various history files.

       3.1.1 User input parser
       3.1.2 DIALOG input parser
       3.1.3 Network input parser

   3.2 Threshold Analyser. This program makes the simpler checks
       for poor student performance, in some cases triggering
       messages directly, in others triggering more detailed
       analyses.

   3.3 In-Depth Analyses. These are diagnostic programs that take
       longer to execute than threshold checks and are performed
       only when indicated by a threshold check.

4. HELP — consisting of various facilities available to help the
   student understand IIDA, his own performance, or how to proceed
   with a search.

   4.1 Command Explanations

   4.2 Display of Aspects of Search History

   4.3 Change of Mode (into a different exercise)

   4.4 Explanation of Options Available

5. Control Programs and Modification of CONIT. Using the CONIT system is a base, some changes are made to accomodate IIDA's specific requirements.

6. Student Data Base Design — this encompasses all the data that will be recorded about a student's performance. Included is data used for post-search analyses by IIDA staff or during-search analysis by the student.

7. System Testing — this is testing for adherence to specifications. It precedes testing the efficacy of the IIDA system.

   7.1 Design of Test Plans

   7.2 Execution of Tests

   7.3 Revision of Programs

Program Element

1. Exercise Mode

   1.1 Introductory Exercise

   1.2 Limited Language Exercise

   1.3 Additional Commands Exercise

2. Assistance Mode

3. Diagnostic Programs

   3.1 Parser

      3.1.1 User Input
      3.1.2 DIALOG Input
      3.1.3 Network Input

   3.2 Threshold Analysis

   3.3 In-Depth Analysis

4. HELP

   4.1 Command Explanations

   4.2 Search History

   4.3 Mode Change

   4.4 Options Available

5. Control Programs

6. Student Data Base Design

7. System Testing

   7.1 Design

   7.2 Execution

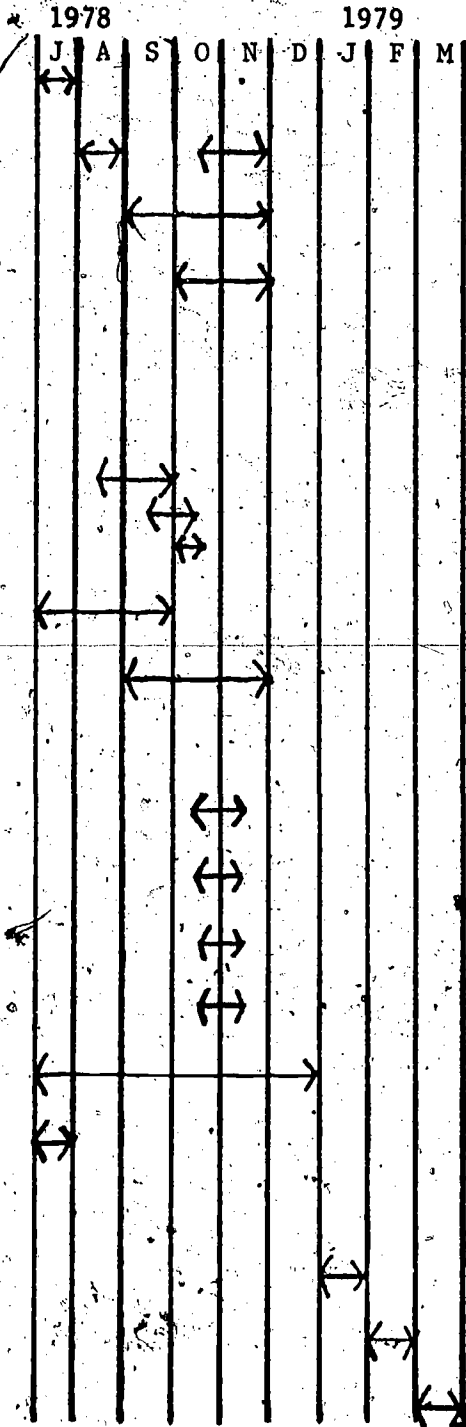   7.3 Revision

Dates

1978   1979

J A S O N D J F M

Table 8.1 Computer Program Production Schedule

III. IIDA PUBLICATIONS

The following is a bibliography of publications on the project
by members of the staff.

Meadow, Charles T., et al. Individualized Instruction for Data
Access (IIDA): Final Design Report, NSF Grant Number DSI-76-09737,
Philadelphia, Drexel University, Graduate School of Library
Science, July 1977.

Meadow, Charles T., Epstein, Bernard. "Development of the Individual-
ized Instruction for Individual Access (IIDA) System." First Inter-
national On-Line Information Meeting, London, December 13-15, 1977.

Meadow, Charles T. "Computer Assistance in the Performance of
Interactive Bibliographic Searching." American Society for
Engineering Education, Vancouver, June 18-22, 1978.

Meadow, Charles T. "Online Searching and Computer Programming:
Some Behavioral Similiarities." Online, October, 1978 (forthcoming)

Meadow, Charles T., Toliver, David E., Edelmann, Janet V. "A
Technique for Machine Assistance to Online Searchers." American
Society for Information Science Annual Conference, New York, November,
1977 (forthcoming)

8ი

APPENDIX A

CONTEXT STRINGS IN IIDA

Each character position in the context strings indicate an aspect of the search context. The particular components of the context strings proposed for use in the IIDA system are listed below. These components are a mixture of those used in the CONIT system and those proposed especially for the IIDA system. The format presenting these, below, will follow that given in Appendix A of the CONIT report (Marcus, p. 50). For each type of component is listed: its symbolic tag (ST), the number of characters it contains (NC), the character position in the context-string (CP), the component name (NAME), and a list of the particular component strings themselves. The project in which the code originated is given (in parentheses) after the code's description.

ST	NC	CP	NAME
A	1	1	Primary Search Context

u: command from user (CONIT)
h: /HELP command rules (IIDA)
m: mode evaluation passed (IIDA)
r: response from retrieval system (CONIT)
a: threshold and in-depth analyses (IIDA)
f: front-end mode-oriented instruction (IIDA)
z: special error condition (CONIT)

B1	1-3	2-4	Connection Codes

not: not connected (CONIT)
log: logging in (CONIT)
con: connected (CONIT)
det: detached (CONIT)

B2	2	3-4	Special and Error Codes

(strings in this group replace the last 2 characters
of B1 under special and error conditions)
eq: user quit - break (CONIT)
eh: phone line hung up (CONIT)
ep: phone connection not made (CONIT)
ei: MULTICS i/o error detected (CONIT)
nr: expected numeral is missing (CONIT)
to: timeout 1 (CONIT)*
tt: timeout 2 (CONIT)*
el: timeout 3 (CONIT)*
99: specific message-pointing error indicator (IIDA)
er: general error indicator (IIDA)
sy: general syntactical error (IIDA)
zp: zero-postings error (IIDA)
th: thrashing error (IIDA)
dw: dwelling error (IIDA)
id: general in-depth analysis required (IIDA)

*See the CONIT Report, Appendix A (Marcus, p. 50) for a further explanation of these timeout conditions.

ST	NC	CP	NAME
C	1	5	Network Connected

n: none (CONIT)
e: Telenet (CONIT)
y: Tymnet (CONIT)

D1	1	6.	Selected Search Mode

c: canned search - first exercise mode (IIDA)
s: subset search - second exercise mode (IIDA)
f: full search - third exercise mode (IIDA)
a: assistance mode (IIDA)

D2	1	7	Program-Controlled Submode

letters in order a-z: submode of search (IIDA)

E	0-2	8-9	Command Indicators

be: BEGIN (IIDA)
s: SELECT (IIDA)**
ex: EXPAND (IIDA)
en: EXPLAIN (IIDA)
c: COMBINE (IIDA)**
ot: TYPE (IIDA)
od: DISPLAY (IIDA)
op: PRINT (IIDA).
ds: DISPLAY SETS (IIDA)
lt: LIMIT (IIDA)
zz: END (IIDA)

**Position 9 may be used to indicate the type of argument in the SELECT and COMBINE commands.

F	0-2	10-11	Sequence Number of Logically Linked Rules

alphanumeric codes (CONIT & IIDA)

NOTE: Refinements on these context codes can be expected when actual programming gets underway.

APPENDIX B

PARTIAL LIST OF IIDA SPECIAL-ACTIONS

Many of the special-actions provided by CONIT will be kept in the IIDA software. See Appendix B of the CONIT report (Marcus, p. 52) for a list of many of the CONIT special-actions. These will be selectively kept or dropped depending upon whether or not IIDA needs them. IIDA will add a large number of special-actions for its own purposes. Those listed and described below are by no means exhaustive of IIDA special-actions. Rather, they are intended to recapitulate special-actions described as examples in this report. They further serve as starting points for developing the full set of IIDA special-actions.

HELP HISTORY MANAGEMENT

timerec: records in the help history data-structure the entry time, i.e., time at which help was requested.

hupdate: updates the help history data-structure with the last entry of the user in the input stream.

help2,...help8:   formats the IIDA history data-structures and sends them to the user's terminal for printing.

htime: records the current time into the help history data-structure and performs the help use threshold analysis.

inhelp: performs in-depth analysis on the use of IIDA help.

MESSAGE STORAGE AND COMMUNICATION

stcom: stores command input by user in its place in the command history data structure, viz., C_TEXT (C_HIST).

copyc: copies the entire input stream to the host output buffer. This will later be sent to the host if the command is determined to be without error in all respects.

rsendc: sends the host buffer to the DIALOG retrieval system and continues processing rules in the table.

copys: copies the entire contents of the retrieval system buffer to the user buffer and sends buffer to user.

PARSING ARGUMENTS AND RESPONSES

pars..:   general form of special-actions which parse the argument of each command type. This includes storing the parts of the argument in the data-structure and, in some cases, generating analytic data.

rsvp...:   general form of special-actions which parse the retrieval system responses for each type of command. This includes updating the various data-structures.

SPECIAL-ACTIONS FOLLOWING ERRORS

thsyn: analyzes the threshold of syntactical errors.

idsyn: analyzes in-depth patterns of syntactical errors.

syerr: updates the data structure with a code indicating that a retrieval system error followed transmission of the last command.