Information Processing & Management Vol. 19, No. 2, pp. 65 75, 1983
Printed in Great Britain.

0306-4573/83/020065-11$03.00/0
Pergamon Press Ltd.

A PARSER FOR ON-LINE SEARCH SYSTEM EVALUATION†

RUSS TREMAIN
National Council on Crime and Delinquency, San Francisco, California, U.S.A.

and

MICHAEL D. COOPER
School of Library and Information Studies, University of California, Berkeley, CA 94720, U.S.A.

(*Received for publication* 10 *December* 1982)

Abstract—This paper describes a computer program which converts the text of user input and system responses from an on-line search system into fixed format records which describe the interaction. It also outlines the syntax of the query language and the format of the output record produced by the parser. The study discusses problems in constructing the parser, the logic of the parser and its performance characterestics, as well as recommendations for improving the process of logging on-line searches.

INTRODUCTION

There are a number of ways in which on-line bibliographic search systems like OCLC, RLIN, DIALOG, MEDLARS, ORBIT and BRS can be evaluated. Computer performance evaluation techniques can be used to determine if the hardware configuration is adequate for the system. This includes evaluating the adequacy of central memory, disk drives, printers, telecommunication ports, and channels, as well as the central processing unit.

Another aspect to the evaluation determines the adequacy of the software for the system. In a typical search system there are four layers of software that work together to provide searching capability. The first is the operating system which coordinates computer system resources and schedules and allocates resources to various programming tasks. A second layer is the set of programs that provide telecommunications support. A user of a system usually interrogates the system by telephone, dialing a number that allows remote access to one of the computer system's ports. All of the extensive control procedures involved in receiving a message over the telephone lines—buffering the message, translating messages from one type of code to another, handling errors, handling the protocols connected with various types of terminals, and transmitting messages—are performed by the telecommunications programs. The third component is the search system programs themselves. These programs accept as input a user query, analyze what needs to be done in the way of an on-line search request, process that request, and send the results to the telecommunication program for transmission to the user.

In the course of processing the user request, the search system uses a fourth component, a data base management system, to access files of bibliographic data. The data base management system provides the tools necessary to create, access, and maintain the bibliographic files. In some cases the data base management system is not a separate program, but rather part of the search system itself.

Each of the four components described above can be evaluated in many different ways. This paper describes a tool to assist in the evaluation of the search system itself. The tool is a parsing program that analyzes the dialog that takes place between the user and the search system. The parser takes the text of messages and translates them into a single record describing a single interaction between user and program. The set of output records

†The support of Kent Smith, Deputy Director of the National Library of Medicine is gratefully acknowledged. Peggy Jarvie's aid in the problem definition stage, Louise Whittaker's initial programming assistance, and Ann Marie Macker's technical advise were invaluable.

produced by the parser can then, at a later stage, be analyzed to determine the nature of the dialog between the user and the system. This can be performed with standard statistical packages like SAS or SPSS.

This type of performance analysis is useful for a number of reasons. First, the computer hardware used to run search systems tends to be very expensive. It is not unreasonable for the purchase price of the equipment in such an installation to be five to ten million U.S. dollars. Thus the incentive to improve the performance of such systems is strong. Another reason for this investigation is to improve user satisfaction. If measurement techniques can be developed to pinpoint bottlenecks in the system, those bottlenecks can be removed and the system's response time improved. Computer systems have a limited capacity. That is, they can only support a finite number of simultaneous users. Evaluation allows decisions to be made about whether and when capacity limits have been reached.

Analyzing a user's dialog with the system is particularly important. By studying the way users conduct on-line searches, a greater understanding of the search process can be obtained, better training programs can be developed, and more responsive and useful systems designed.

An analysis of the user-system dialog provides important information about the performance of the computer system and how users interact with the system. This provides basic data about how to improve the total system from both the perspective of the system manager and the system user.

There are two major software monitoring methods by which the information about user-system interaction can be gathered. The first and most desirable is an extensive logging facility built into the search system. This facility generates log records describing each facet of the user-system interaction as the search system performs the search process. Such a methodology has been incorporated into the MELVYL, the prototype on-line catalog at the University of California[1]. Less desirable but more common is the search system with primitive logging facilities. In this approach the system simply writes out a record of what the user entered and what the system sent to the user. This set serves as the record of what the system did and as a basis for evaluating user-system performance. This approach requires minimal effort to implement and has almost no impact on the performance of the search system itself. However, when performing an evaluation of user-system interaction with this monitoring methodology, sophisticated tools must be developed to assist in the analysis.†

PROJECT DESCRIPTION

The project to evaluate the performance of on-line search systems has a number of phases. In this report a description of the query language used by the National Library of Medicine ELHILL search system is given, and the log records produced by the system are outlined. Then the output record produced by the parser is presented and the logic of the parser is discussed.

Subsequent papers will analyze the data generated by the parser with the purpose of understanding the use of the system. The reports will deal with response time characteristics of the system, characteristics and content of on-line searches, and models of search behavior.

THE PROCESS OF LOGGING TRANSACTIONS

The program described in this paper is called MLPARS4.‡ It operates on log records of transactions produced by the ELHILL search system of the MEDLARS II bibliographic programs. The log records are generated by the ELHILL system in a relatively simple manner. As the user enters a search statement, command, or response into the system, that line of text is date and time stamped, and along with the user's logon code is written onto a file of log records. The ELHILL system processes the user's command

†A review of many additional problems of monitoring and evaluating the performance of on-line systems can be found in the paper by Penniman and Dominick[2].

‡MEDLARS Parser Version 4.

and produces a response which is then transmitted to the user. It is also date and time stamped and written onto the file of log records.

The first task performed on the records is to sort them by user logon code, date, and time. This step is necessary because many users have their requests processed by the machine almost simultaneously, and the records stored in the log file are not arranged sequentially by a particular user's search.

Once the records have been sorted, they are ready to be processed by MLPARS4. Its first task is to group them into *sessions.* A session is defined as one continuous period of time during which one user with a single user logon code performs a search. Once it has found a group of records that comprise a session. it is ready to analyze each record in the session to see if that record is a *user transaction* or a *program transaction.* A user transaction is an input message generated by the user, and a program transaction is a message generated by the ELHILL system. There can be more than one program message generated for each user message.

THE ELHILL QUERY LANGUAGE

The MLPARS4 program analyzes the user and program transactions from the ELHILL system. In order to clarify the functioning of the program and the type of output produced, it is useful to describe the commands and describe their syntax. Table 1 lists the commands and provides a brief description of each. One of the most common commands, a search statement, is not listed. A search statement is equivalent to a FIND command and is composed of terms and operators, but lacks the word 'FIND' preceeding those terms and operators.

The syntax of the query language for selected commands is described in Table 2.† The grammatical notation used in the table is relatively simple. The symbol '+' means that there may be one or more repetitions of the expression it follows. The symbol '*' means there may be zero or more repetitions of the expression. A '?' is used to show that a component is optional in the command, the '|' (or) represents the case where one of a set of items may be chosen, and symbols with single quotes around them are literal values. Parentheses are used to group multiple elements into a single logical entity.

Consider the FIND command as an illustration of the notation. The description begins by defining the command as consisting of two parts:

u_find -> '"' ? 'FIND' ? (fdterm bin_op?) +

The first part says that the command may begin with a double quote followed by the word FIND. The question mark indicates that the entire phrase is optional. The second part indicates that the command will consist of one or more (+) sequences of a required symbol called fdterm followed by an optional occurrence (?) of the symbol called bin_op.

The remaining part of the entry for the FIND command defines the fdterm and bin_op symbols. A fdterm can be a search statement number (⟨ss_number⟩), the words 'LESS THAN', 'GREATER THAN', or 'FROM' ⟨int⟩ 'TO' ⟨int⟩, where ⟨int⟩ is an integer. The last possible meaning of a fdterm is

fdterm -> u_op* ⟨iden⟩ + (('/' ⟨sh⟩)|'(' ⟨cq⟩ ')')*

Here u_op is a unary operator (defined as an asterisk, or the words 'NOT', 'ALL', or 'EXPLODE'). The u_op is followed by an identifier (a search term) and zero or more repetitions of sub headings (⟨sh⟩) and/or category qualifiers (⟨cq⟩).

A sub heading is used in a search to find records that have specific aspects or components. Examples of sub headings along with their two-letter abbreviations include: adverse effects (ae), blood (bl), congenital (cn), and etiology (et). Category qualifiers are used to select certain fields of a record for searching. For example, a searcher may only

†The query language described in Table 2 is the one employed by MLPARS4 to parse user transactions. It does not purport to be an *exact* representation of the ELHILL language.

Table 1. ELHILL commands

Command	Synonym	Description
/LOGIN		Begins login process
CAPS		Forces output to terminal in upper case
COMMENT		Allows user to send messages to Medlars Management staff
DIAGRAM	DIAG	Displays logical structure of search
ELEMENTS APPLY ELEMENTS INCLUDE		Changes type of terms searched for in index
ERASEALL	ERSLL	Clears all previous search statements
ERASEBACK	ERASEBAK, ERSBK, BACKUP	Selectively deletes search statements
EXPLAIN	EX	Elaborates on last program message
FILE		Changes from one file to another
FILES		Gives names of all data bases in system
FIND	FD	Search statement
FINISHED		Signals end of OFFSEARCH, STORESEARCH, or COMMENT
HELP		
MESHNO	MNO	Retrieves a MeSH number for a term or terms
NEIGHBOR	NBR	Finds terms alphabetically adjacent to term supplied
NEIGHBORDETAILED	NBRDET	Lists each form of a main entry term
NEWS		Gives news from NLM
NO	N	Response to many program messages
OFFSEARCH		Initiates a request for off-line search of a data base
PRINT	PRT	Displays search results
PURGESEARCH		Removes a STORESEARCH from the system
RENAME	RNM	Allows commands to be renamed
RESTACK	RSTK, KEEP	Selectively keeps search statements
RESTART	RST	Clears all previous commands and search statements
SENSEARCH	SENS, ABSTR	Searches text of a retrieved set for a word or phrase in same sentence
STOP		Terminates search session
STORESEARCH		Initiates process of storing user search
STRINGSEARCH	STRS	Searches all text of a retrieved set for a given word or phrase
SUBHEADINGS APPLY		Applys subheadings to search statements
TIME		Display cumulative search or connect time
TITLESEARCH	TS	Searches all text of a retrieved set for a given word or phrase
TREE		Displays hierarchical position of term in MeSH structure
USERS		Tells number of users logged onto system
VERSION		VERS
YES	Y	Response to many program messages

Table 2. ELHILL command syntax for selected commands

```
u_diagram     -> u_yesno | '*' ? 'DIAGRAM'

u_explain     -> '*' ? 'EXPLAIN'

u_find        -> '*' ? 'FIND'  ? (fdterm  bin_op ? ) +

      fdterm -> <ss_number>   {integer}
             -> u_op * <iden>  + (('/' <sh>) | '(' <cq> ')' ) *
             -> 'LESS' 'THAN' <int>
             -> 'GREATER' 'THAN' <int>
             -> 'FROM' <int>
             -> 'TO' <int>     {must have 'FROM' <int> first}
      u_op   -> '*' | 'NOT' | 'ALL' | 'EXPLODE'
      bin_op -> 'AND' | 'OR'

u_help        -> '*' ? 'HELP'

u_meshno      ->  '*' ? 'MESHNO'

u_mmresp      ->  'ALL' | 'NONE' | 'EXPAND' | int_list

      int_list-> <int> (',' ? <int>) *

u_nbr         -> '*' ? ( 'NEIGHBOR' | 'NEIGHBORDETAILED' | u_updown)

u_news              -> u_yesno | '*' ? 'NEWS'

u_printcmd    -> 'PRINT' (<const_parm> | cq_list | skip_parm | ssn_parm|
                 ',' || <iden> | <print_request_cnt> ) *

      skip_parm-> 'SKIP' <int>
      ssn_parm -> 'SS' <int>
      cq_list -> <cq>  (',' ? <cq> ) *

u_strsch      -> '*' ? ('STRINGSEARCH' | 'TITLESEARCH' | 'SENSEARCH') ?
                 ('(' <cq> ')' | 'SKIP' <int> | 'SS' <int> ) ?
                 (u_tsterm  u_ts_binop ? ) +

u_tsterm      -> 'NOT' ? (<int> | <iden> | ':' ) + '(' <cq> ')' ?
u_ts_binop    -> 'AND' | 'OR'

u_stop        -> '*' ? 'STOP' ('YES' | 'Y') ?

u_subsapply   -> '*' ? 'SUBHEADINGS' 'APPLY' ? <sh> (',' ? <sh> ) *

u_subscancel  -> '*' ? 'SUBHEADINGS' 'CANCEL'

u_tree        -> '*' ? 'TREE' <iden>

u_updown      -> ('UP' | 'DOWN') | (('UP' | 'DOWN') ? <int>) |
                 ('NO' | 'NONE' | 0)

u_yesno       -> '*' ? 'YES' | 'NO'
```

Note: The grammatical notation used is as follows: '+' - one or more replications of the expression, '*' - zero or more replications of the expression, '?' - an optional component, and '|' - choose one from the set. Symbols with single quotes around them are literal values, and curly brackets delimit comments.

Each command has a different lexicon, i.e. symbols in the input are delimited by different sets of characters. Blanks are always delimiters; others include puncutation marks such as '*', ',', '/', depending on the command.

The lexical analyzer also screens tokens into lexical classes, which vary by command; e.g. category qualifiers, subject headings. These lexical classes are delimited by quoted angle brackets, e.g. '<cq>'.

want to search the title of a record for a particular term. Examples of category qualifiers and their two-character abbreviations include: abstract (ab), author (au), and title (ti).

OUTPUT RECORD DESCRIPTION

The MLPARS4 program analyzes user and program transactions from the log files, and for each set of user transactions and associated program response transactions it creates a fixed format record which can record almost every type of interaction between the user and the ELHILL program. The output record (Table 3) contains a number of sections. The first is general information about the session, e.g. the date and time of the search. The second section contains information about the type of command that the user issued and the type of message that the program issued in response to the user command. Other sections of the record contain information about particular program responses,

Table 3. Output record format

Field No.	Field Name	Description
1.	Session number	Unique number assigned by parser to the session
2	Transaction number	Transaction number within session
3.	Day of year	
4.	Year	
5.	Parse status code	Code indicating whether parse was successful
6.	User time	Time of the user input transaction
7.	Program times	Array of times that each program response was issued
8.	User line length	Number of characters in user input line
9.	Message type	Type of program message issued for this user transaction (cue, error, informatory, system)
10.	File name	The file currently being searched by the user (e.g. MEDLINE, AVLINE, SERLINE,CHEMLINE)
11.	User command code	Code representing the command the user entered
12.	Program response code	Code representing the response issued by the program
13.	Program cue code	The type of cue issued by the system (e.g. none, continue printing, search stmt. cue)
14.	Multi-meaning count	The number of entries in the array in field 15
15.	Multi-meaning terms	Array containing terms found in a multi-meaning response from the system
16.	No postings count	The number of entries in the array in field 17
17.	No postings terms	The terms that the system found not to have any postings associated with them
18.	Postings count	The count of the number of postings resulting from a search statement
19.	Searched count	The number of items searched as part of a SEARCHED/QUALIFIED command
20.	Qualified count	The number of items that qualified as part of a SEARCHED/QUALIFIED command
21.	Beginning search stmt. number	Search statements are numbered. This is the search statement number at the beginning of the transaction
22.	Ending Search stmt. number	Search statement number at end of transaction
23.	String search skip count	The number of items the user requested skipped in a string search.
24.	String search search stmt. number	The search set specified in a string search

Table 3. (*Contd*).

Field No.	Field Name	Description
25.	Category qualifier code	The global category qualifier specified for a search
	Parameters in a User Command	
26.	Multi-meaning response code	The response given by a user to a program multi-meaning message (e.g. all, expand, none)
27.	Multi-meaning count	The number of meanings selected if a count was found
28.	Neighbor response code	A code representing the response given by the user to a program NEIGHBOR command response (e.g. up, down, quit)
29.	Neighbor up-down count	The number of terms to move up or down
30.	Sub headings count	The number of entries in the array in field 31
31.	Sub headings	Array containing the sub headings used in a subs apply or elements apply command
32.	Elements apply count	The number of entries in the array in field 33
33.	Elements apply	Array containing the elements apply codes used in a subs apply or elements apply command
	User Search Term Section	
34.	User term count	Number of terms in user search statement
35.	Search terms	Array containing the terms entered by the user in a search statement
36.	Set numbers	Array containing the set numbers used in a search statement
37.	Subheading codes	Array containing subheading codes used in a search statement
38.	Category qualifier codes	Array containing category qualifier
39.	Truncation codes	Array indicating which terms in a search statement were truncated and how they were truncated
40.	And operators	Array indicating the position of an 'and' operator in a search statement
41.	Or operators	
42.	Not operators	
43.	From-To operators	
44.	Less than operators	
45.	Greater than operators	
46.	All operators	
47.	Star operators	
48.	Explode operators	
	Print Parameters Section	
49.	Skip count	The number of items to skip as found in a PRINT command skip parameter
50.	Print request count	The number of citations to print
51.	Search statement no.	The search statement number used in a PRINT command
52.	Category qualifier ct.	The number of category qualifiers used in the PRINT command
53.	Category qualifiers	The category qualifiers used as part of tailored print or arguments to an exclude or include parameter of a PRINT command
54.	Print parameter count	The number of entries in the array in field 55
55.	Print parameters	Array containing the parameters in a PRINT command (e.g. compressed, detailed, full, indent, standard, only)

arguments found in parameters of user commands, user search terms, and parameters of PRINT commands.

When a user enters a search statement, that statement may consist of a number of search terms, boolean and other operators, category qualifiers, truncation codes, sub heading codes, and other search statement numbers. This information is recorded in the output record by storing each search term in a separate element of an array in a position corresponding to the sequence with which it occurred in the input string. (See field 35 in Table 3). Other operators and codes are stored in their own arrays. (See fields 36 through 48 of Table 3).

In response to a search statement, the user can receive a 'postings' message which tells the number of postings for the search statement, a 'multi-meaning' response which indicates that some search terms have multiple meanings, or a variety of other responses. The information returned by the system is recorded in fields 12–20 of the output record.

Certain statements, such as the PRINT command, have many parameters that can be used with it. Fields 49–55 record the information that the user enters as part of the PRINT command. For example, the user may request that a certain number of citations be skipped before or during the printing of output. The number of citations to be skipped is recorded in field 49. Many parameters other than those specified in fields 49–54 can be present in a PRINT command. They include parameters like compact, indented, exclude, include, compressed, full, and detailed. Print qualifiers are two-character category qualifiers that can be used as part of a PRINT statement. These parameters are stored in an array in field 55 of the output record.

STRUCTURE OF THE PARSER

The MLPARS4 program has five major components: (1) a transaction analyzer, (2) symbol table handling routines, (3) a transaction builder, (4) an ELHILL message recognizer, and (5) a lexical analyzer.

The transaction analyzer controls the processing of log records. When records are needed for processing, the analyzer calls the transaction builder to get a set of user and program transactions. Then it calls the message recognizer to try and ascertain if the program transaction has an ELHILL message in it. The lexical analyzer is used to isolate and screen tokens in the user and program messages and the symbol table handling routines are used throughout the program.

Symbol table handling routines

There are a number of symbol tables used in analyzing user and program transactions. They include a list of subject headings, category qualifiers, keywords used in search statements, data base names, responses to multi-meanings commands, and responses to NEIGHBOR commands. There are also keywords that appear in a program response message (for example the words 'you', 'are', 'now', 'connected' would appear in the symbol table file and would be used by the parser to help determine whether a particular program response during a session was a program message. There are also files that contain the parameters that can be used in a print statement (like 'compact', 'indented', 'full'), and files that contain equivalencies between various forms of commands and parameters (e.g. the equivalences between the short and long form of subject headings ('mi' and 'microbiology') and short and long forms of commands ('ts' and 'titlesearch'). The symbol table handling routines initialize these tables as well as maintain them.

The description of the query language employs a number of lexical category names. The symbol table handling routines also are used to post specific tokens to these categories to create equivalencies between a number of specific tokens and a particular lexical category. They also count the number of tokens posted to a category and use this information for error checking and MLPARS4 program optimization.

The transaction builder

Another component of MLPARS4 is the transaction builder. This program reads input records looking for the start (or end) of a session. Once the beginning of a session has been

found, the transaction builder forms the transactions into groups consisting of a user input transaction and one or more program output transactions. In some cases this is not a simple task. The time-of-day that is recorded on each log record is measured to the nearest tenth of a second. A number of transactions may occur in the same tenth of a second, thus it is difficult to determine the exact ordering of the log records based on time. The transaction builder uses a simple heuristic to supply records to the transaction analyzer in the correct order.

The ELHILL message recognizer

The objective of the MLPARS4 program is to successfully determine what the user said and what the system said, convert that text to a fixed format record, and write the output record onto a file.

An impediment to a straightforward parse of the records is that only the first 57 characters of the user input and the program response are recorded on the log file records. This means that no full recording of a session exists. Because of the (usually) shortened program transaction, it is often difficult to determine what was the prompt that the program issued after it completed responding to one user transaction.

A number of clues, algorithms, and heuristics were ultimately derived which allowed successful parsing to take place. Deriving those rules that could be consistently trusted to yield accurate results was an empirical process which involved implementing the program in a number of intermediate stages, viewing the frequency of success, and refining the program.

The single most important clue used to parse the transactions is the presence of lowercase characters in the ELHILL response to a user PRINT command. Normally, this clue allowed MLPARS4 not to have to look at these text responses and thus usually avoid having to decide whether the text output was a standard system message (which is always displayed in upper-case) or natural language text resulting from a print command.† More importantly, this process also made it possible to determine over time most of the messages legitimately produced by ELHILL and not retrieved from one of the ELHILL data bases (e.g. MEDLINE).

The second major clue used in parsing the transactions was a decision that one of the most stable aspects of the user-system dialog was the program messages generated by the ELHILL system. The process of matching the program messages to the program transaction tokens involves consulting a data base containing all the program messages issued by the ELHILL system. The ELHILL message recognizer compares program tokens to message tokens to find a match. If the match is successful, the parser has found a message that matches the program transaction. The parser then records the message number in the output record and then parses the user input only if of interest to the analysis. The user transaction is much less defined or structured than the program transaction, so the parser defers handling it until it needs to. In many cases it is not necessary to parse the user input because what the user did can be inferred from the program transaction. In other cases this is not true and the user transaction must be parsed.

The Parse action data base

One of the most important design aspects of MLPARS4 is that it is table-driven. Instead of the logic being imbedded within the code of MLPARS4, it has general processes that are invoked as needed depending on values in various tables and results of previous actions. The most important table is the *parse action data base*. Much of the decision-making about what to parse, how to parse, and when to parse is recorded in this data base.

The parse action data base contains a description of each of the more than 100

†There are exceptions to this rule. For example, the output from a TREE command can contain lower-case, a multi-meaning response from the system can have lower-case characters in it, and a MeSH Number can have lower-case characters. But except for these cases, the presence of lower-case characters is an important clue to the parser about how to proceed.

messages that can be produced by the ELHILL program. The messages are numbered and stored in a generalized manner. For example, the data base contains the message

⟨quoted_tok⟩ IS NOT A CORRECT COMMAND NAME.

Here the meta-symbol ⟨quoted_tok⟩ is used to represent any word with single quotes around it. A specific instance of this message might be issued when the user types the word PRUNT instead of PRINT:

'PRUNT' IS NOT A CORRECT COMMAND NAME.

The data base serves as the input to another computer program that generates the programming language statements that comprise the portion of MLPARS4 that recognizes messages (the ELHILL Message Recognizer). That is, there is a separate computer program that takes as input the parse action data base. That program creates a set of PL/1 statements that when executed will detect one of the 107 ELHILL program messages that can occur in the program transaction. The implication of having a separate program generate the recognizer, rather than having the logic of recognizing the statements imbedded in the parser itself, is that as additional program messages are found, MLPARS4 PL/1 statements can be generated to recognize them with ease.†

The data base contains a number of other pieces of information about each program message. To optimize matching between program transaction tokens and messages, the comparison is ordered by frequency of occurrence of the messages in a sample of messages taken in the course of building MLPARS4. The data base stores this frequency and uses it to order the evaluation statements. As an aid to debugging, the data base also records the session, and transaction numbers within the session, where an instance of the message was found. This aided considerably in analyzing the context within which the message was issued.

The data base contains an indicator of the type of the message (normal, error, informatory, cue, or system) issued. It also gives the parser the name of the subroutine that will do a parse of the user transaction, or indicates that a parse of the user transaction is not to be done.

The lexical analyzer

The MLPARS4 program defers as long as possible a decision about whether to analyze the user transaction. Lexical analysis of the program transaction always takes place because the parser needs the program transaction broken down into tokens to match it against the program messages and to scan it for the presence of a program cue. When MLPARS4 decides to actually parse the user transaction, it first invokes the lexical analyzer to break the user transaction into tokens.

The lexical analyzer consists of a scanner and a screener. The scanner breaks up the input into tokens based on the set of look-ahead characters passed to it by one of the user command parsers. (A look-ahead set is a set of characters that delimit the lexical components of the input). The screener uses the set of symbol table names passed to it by the command parsers to classify each token into one of several classes, e.g. category qualifiers, user command keywords. Thus the transaction analyzer can successively apply different grammars to the same input until the input is recognized.

The order in which the grammars are applied is important. For example, the time-overflow message can be generated in response to a search statement (i.e. FIND command), a string search, or a YES response in a long sequence of time-overflow messages. Since the language generated by the search statement grammar contains the

†When the project began we believed that there were 37 program messages from ELHILL. Now we are not sure that we have found them all, but our count is at 107. Regenerating the ELHILL message recognizer was relatively easy as new messages were encountered.

language generated by a user response to a time-overflow message (yes or no), the latter grammar must be applied first.†

PERFORMANCE CHARACTERISTICS OF THE PARSER

The parser is a relatively large program written in PL/1. It is close to 5700 lines long. Tests on a sample of 194,371 input records indicate that the central processing unit time to parse an average session read from tape is 1.87 seconds, and that the time to process one transaction is 0.06 seconds, on an IBM 370/168. For every 2.39 input log records that the parser reads, it produces one output record.

CONCLUSIONS AND RECOMMENDATIONS

This paper has reviewed some of the methods of evaluating on-line search systems. Emphasis has been on analyzing searches from the National Library of Medicine's MEDLARS system. The paper presented a symbolic definition of the query language used in the system and described a fixed format output record that can store search statements and program responses. The paper then described the problems inherent in converting an abbreviated form of log record produced by the system into the fixed format record. The structure of the parser that converted the log records to fixed format was also discusseed and its major components described.

Surprisingly complete descriptions of the user-system dialog were obtained with the parser, but its implementation was a major effort. The output from the parser produces an excellent source from which to analyze search and performance patterns.

The evaluation process could be more comprehensive and accurate if a few changes were made in search logging procedures. These include more accurately recording the time a transaction occurred in the system (more significant digits in the time stamp), and the full text of the transaction rather than only the first 57 characters in the log record. An important feature to add to the logging facility would be one that clearly distinguishes natural language text from legitimate system messages. This would have considerably simplified the development effort. Monitoring other parts of the system, such as telecommunications and operating system performance, would be useful in a broader evaluation since all aspects of the system could then be analyzed together to evaluate performance.

REFERENCES

[1] RAY R. LARSON, *Evaluating public access on-line catalogs: phase 1 development and testing of data collection and analysis tools.* Final Report to the Council on Library Resources, Division of Library Automation, Office of the Assistant Vice-President Library Plans and Policies, University of California, Berkeley, California, July (1981).

[2] W. D. PENNIMAN and W. D. DOMINICK, Monitoring and evaluation of on-line information system usage. *Inform. Proc. Management* 1980, **16**, 17–35.

†This would not in itself guarantee a correct analysis of the transaction except for the fact that the search statement 'yes' never generates a time-overflow.