Lehigh University
Lehigh Preserve

Theses and Dissertations

1-1-1980

A program for storage allocation in DECSYSTEM-20's DBMS.

Tung-Sheng Lai

Follow this and additional works at: http://preserve.lehigh.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation
Lai, Tung-Sheng, "A program for storage allocation in DECSYSTEM-20's DBMS." (1980). *Theses and Dissertations.* Paper 2283.

This Thesis is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

A PROGRAM FOR STORAGE ALLOCATION IN DECSYSTEM-20'S DBMS

by

Tung-Sheng Lai

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computing Science

Lehigh University

1980

ProQuest Number: EP76559

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest EP76559

Published by ProQuest LLC (2015). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

Sept. 18, 1980
(date)

Professor in Charge

Chairman of Department

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A PROGRAM FOR STORAGE ALLOCATION IN DECSYSTEM-20'S DBMS

by Tung-Sheng Lai

This thesis presents a computer package designed to help the data base designer in the proper storage for DECSYSTEM-20's Data Base Management System(DBMS). The DEC's DBMS is based on the 1971 CODASYL Data Base Task Group(DBTG) proposal. Although the Schema Data Description Language(Schema DDL) had been defined in reasonable detail by the CODASYL committees, the Device Media Control Language(DMCL) was left out as this is dependent on the specifications required by the implementor.

Although storage allocation is a very critical aspect in data base design, there is no systematic approach to it. Due to this fact, most of the data base designers arbitrarily assign storage much larger than is really needed for the data base. Obviously, this is not an economical approach. On the other hand, some data base designers simply ignore the storage considerations, and as a result,

eventually face the situation of having to unload the data base to increase the storage, and/or rewrite some or all of the application programs.

The computer package presented in this thesis is designed to help the data base designers allocate appropriate storage for future growth of existing applications and new applications. This package is a PASCAL program, composed of two parts, a parser and an interactive program.

The first part is a parser, which reads the user's Schema DDL as input, examines the syntactical structure of the Schema DDL statements, and stores all the information that will be used in storage calculation. The interactive program helps the user to develop his DMCL statements by asking to provide more information about the data base and its usage in the future.

Information on how to access this package for use is available from Division of Computing and Information Science.

1. INTRODUCTION

1.1 Background

The proliferation of digital computers and the widespread demand for timely access to data has resulted in the need to use data bases. The term Data Base became current in the late 1960s. James Martin(Ref.7,22) defined Data Base as follows:

> A data base may be defined as a collection of interrelated data stored together without harmful or unecessary redundancy to serve multiple appilications; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and in modifying and retrieving existing data within data base.The data is structured so as to provide a foundation for future application development. One system is said to contain a collection of data bases if they are entirely separate in structure.

The stages of data base design have only recently been well defined. Some of the main approaches to data base design-TOTAL, IMS, RELATIONAL and CODASYL(Conference on Data Systems Languages) approaches. There are several commercial

implementations of the CODASYL approach available, the best known of which are Univac's DMS 1100, Honeywell's IDS/2, Cullinane's IDMS (available on IBM machines and also on ICL machines), Siemen's IDMS and DECsystem-20's DBMS. The latter implementation is the main concern of this thesis. The DECsystem-20 Data Base Management System (DBMS) is a group of programs that enable an installation to create, access and maintain one or more data bases. DEC's DBMS is based on the 1971 CODASYL Data Base Task Group(DBTG) proposal(Ref.2).

1.2 Components of a DBMS

DBMS's five major components are identified as:

a. Schema Data Description Language(Schema DDL)
b. Sub-Schema Data Description Language (Sub-Schema DDL)
c. Data Manipulation Language(DML)
d. Data Base Control System(DBCS)
e. Device Media Control Language(DMCL)

Each will be discussed in turn.

1.2.1 Schema DDL

To the DBTG, a schema may be defined as the description of

a data base. The Schema Data Description Language is a free standing declarative language which is used to define the structure of a data base. These descriptions are in terms of the names and characteristics of the DATA-ITEMS, DATA-AGGREGATES, RECORDS, SETS, and AREAS included in the data base, and the relationships that exist and must be maintained between occurrences of those elements in the data base. The Schema DDL does not define the processing to be performed on the data. It does not even allow statements to be made concerning the amount of data to be stored in the data base.

A DATA-ITEM is the smallest unit of named data. It is a subdivision of a record.

A DATA-AGGREGATE is a named collection of data-items in a record.

A RECORD is a named collection of data-items and/or data aggregates. It is the basic retrievable unit of information in a data base.

A SET is a named collection of record types. As such, it establishes the characteristics of an arbitrary number of occurrences of the named set. Each set type specified in the Schema must have one record type declared as its OWNER and one or more record types declared as its MEMBER records.

An AREA is a named sub-division of the addressable storage space in a data base. The occurrences of records are stored in areas. In DEC-20 each area is divided into fixed-length physical units called pages. A page logically consists of some number of lines. Each line is a group of 36-bit words large enough to contain a single record occurrence and its set pointers.

1.2.2 Sub-Schema DDL

A Sub-Schema is best defined as a part of a schema. A useful way to think about the schema and sub-schema is the following. The schema is the overall view of the logical structure of the data base as seen by the central authority responsible for the data base. The sub-schema is the view

of the data base as seen by the Appilications Programmer(Ref.8,9). A sub-schema is defined using a Sub-Schema DDL.

1.2.3 Data Manipulation Language(DML)

DML is the name given by the DBTG to the collection of statement types which must be added to an existing programming language to enable the programming language to be used to process the data in a data base which has been defined using the Schema DDL.

1.2.4 Data Base Control System(DBCS)

DBCS is the interface between the run-unit and the data base; i.e., it is the module that actually performs the actions defined by the DML statements.

1.2.5 Device Media Control Languages(DMCL)

The DBTG report(Ref.2,21:22) identified the role of the DMCL as "the assignment of areas to devices and media space,

and specifying and controlling buffering, paging, and overflow". The above DBMS components have been defined in reasonable detail by the CODASYL committees. However, the DMCL was left out as it is up to each implemenator to state his own specifications.

1.3 Statement of Problem

The development and implementation of a data base system is much difficult than that of a conventional system. The complexity is in both hardware and software. Even if one restricts one's view to just the software, there are many facets to the problem of data base design. Among other things the designer must concern himself with sound logical design, data integrity, privacy control, backup and storage allocation. It is this last aspect, storage allocation in the data base, which is the subject of this thesis.

Although storage allocation is a very critical aspect in data base design, there is no systematic approach to it. Due to the above fact, most of the data base designers

arbitrarily assign a storage which is much larger than is really needed for the data base. Obviously, this is not an economical approach. On the other hand, some data base designers simply ignore the storage considerations, and as a result, quickly face the situation of having to unload the data base to increase the storage, and/or rewrite some or most of the application programs.

The best available technique to overcome the above problem is by using the storage calculation table as shown in Fig. 1. The data base designer uses the table to calculate the data items and/or data aggregates of each type according to its location mode and its relationships with other records.

All these calculations are done manualy. In a simple data base, all these calculations could be done in a few hours. If the data base is huge and the set relationships is complicated, this task would be tedious and time consuming. Moreover, the accuracy of the results could be doubtful.

1.4 Approach to the Problem

It is the aim of this thesis to provide a computer package (named: Data Base Storage Allocation Package-DBSAP) that will help data base designers allocate storage in the data base for future growth of existing applications and new applications. This package is a PASCAL program, composed of two parts, a parser and an interactive program. The first part is a parser, which reads the user's Schema DDL as input, examines the syntactical structure of the Schema DDL statements, and stores all the information that will be used in storage calculation. The interactive program helps the user to develop his DMCL statements by asking him to provide more information about the data base and its usage in the future.

RECORD-TYPE										
Number of Occurrences										
Number of Characters										
Number of Words										
Record Overhead	Calc									
	Owner									
	Member									
	Linked to Owner									
	Linked to Prior									
	Line Header									
	Total Overhead									
Words per Record										
Total Words All Occurrences										
Growth										
Page Organization	Page									
	Page Calc									

Fig. 1. The Storage Calculation Table

2. DESCRIPTION OF THE DATA BASE STORAGE ALLOCATION PACKAGE

2.1 The Content of DBSAP

DBSAP is composed of two parts. The first part is a parser, which reads the user's Schema DDL as input examines the syntactic structure of the Schema DDL statements. This chapter will concentrate on how this parser is developed. The second part is an interactive program, which will be discussed in the next chapter.

2.2 The Parsing Algorithm

Parsing is the process of determining the syntatic strucrs;e associated with an input sentence(Ref.1,56-57). The method used in the parsing algorithm of DBSAP is top-down left to right parsing which consists of reconstructing the generating steps from the start symbol to the final sentence (Ref.5,Ref.6).

There are two essentially different techniques that can be applied to the top-down left to right parsing

```
a. SCHEMA NAME IS schema-name.
b. AREA NAME IS area-name-1
       [AREA IS TEMPORARY]
     [ [PRIVACY LOCK [FOR [EXCLUSIVE  ] {UPDATE   }] IS lock-1 ] .
                     [PROTECTED] {RETRIEVAL}]
c. RECORD NAME IS record-name-1

     LOCATION MODE IS { DIRECT identifier-1 %pseudonym-1                  }
                      { CALC USING data-name-1                            }
                      {            data-name-2 ...                        }
                      {   [DUPLICATES ARE [NOT] ALLOWED]                  }
                      { VIA set-name-1                                    }

     WITHIN area-name-1 [area-name-2...AREA-ID IS
                                identifier-2 [%pseudonym-2] ] .
d. 02 data-name-3 [%pseudonym-3] {PICTURE} IS picture-string
                                 {PIC    }

     USAGE IS [DISPLAY  ]
              [DISPLAY-6]
              [DISPLAY-7] [OCCURS integer-1 TIMES] .
              [DISPLAY-9]
e. 02 data-name-3 [%pseudonym-3] SIZE IS integer-2
                  { WORDS                                  }
                  { USAGE {DISPLAY  }                      }
                  {       {DISPLAY-6}                      }
                  {       {DISPLAY-7}                      }
                  {       {DISPLAY-9}                      }
     [OCCURS integer-1 [TIMES]
f. 02 data-name-3 [%pseudonym-3]
                                  {[FLOAT] [ {DECIMAL} ] [REAL   ]}
                  TYPE IS         {[FIXED] [ {DEC    } ] [COMPLEX]}
                                  {DBKEY                           }

     integer-3 , integer-4 } [OCCURS integer-1 TIMES] .
```

Fig. 2. The General Formats of Schema DDL

g. SET NAME IS set-name-1

```
    MODE IS CHAIN [LINKED TO PRIOR]
                 ┌ ALWAYS  { FIRST }                                          ┐
                 │         { LAST  }                                          │
                 │         { NEXT  }                                          │
ORDER IS         │         { PRIOR }                                          │
                 │ SORTED  [ WITHIN RECORD-NAME                               ]│
                 │         [ BY DATABASE-KEY                                  ]│
                 │         [ DUPLICATES ARE [ FIRST ] ALLOWED                 ]│
                 │                          [ LAST  ]                         │
                 │                          [ NOT   ]                         │
                 └                                                            ┘
OWNER IS { record-name-1 } [.]
         { SYSTEM        }
```

h. MEMBER IS record-name-1 { MANDATORY } { AUTOMATIC }
                           { OPTIONAL } { MANUAL   }

```
    [LINKED TO OWNER]
┌ { ASCENDING           }                                                    ┐
│ { DESCENDING          }  KEY IS data-name-1 [data-name-2]...               │
│ { ASCENDING  RANGE    }                                                    │
│ { DESCENDING RANGE    }                                                    │
└                                                                            ┘
  [DUPLICATES ARE [ FIRST ] ALLOWED]
                  [ LAST  ]
                  [ NOT   ]
┌ [ SET OCCURRENCE SELECTION IS THRU                                         ]┐
│                  { CURRENT OF SET                                          }│
│                  { LOCATION MODE OF OWNER                                  }│
└                                                                            ┘
┌ { USING data-name-2 data-name-3 ...                                        }┐
└ { ALIAS FOR data-name-4 IS identifier-1                                     }┘
                             %pseudonym-1 ...
[.]
```

Fig. 2 (Continued)

algorithm (Ref.9,288). One is to design a general top-down parsing program valid for all possible grammars. In this case, to provide a basis for the operation of the program, particular grammars are to be supplied in the form of some data structure. This general parser is controlled by the data structure and the program is then called table-driven. The other technigue is to develop a top-down parsing program which is specific for a given language and to construct it systematically according to a set of rules which map a given syntax into a sequence of statements, i.e. into a program(Ref.9,288). The later technigue is used in the DBSAP parser. There are two steps in developing the DBSAP parser:

Step 1: Constructing Syntax Diagrams

Fig. 2 lists all the general formats of Schema DDL statements. These formats plus their technical notes(Ref. 3,4-1:4-23) form the syntax of Schema DDL.

A better way to represent the syntax of Schema DDL is

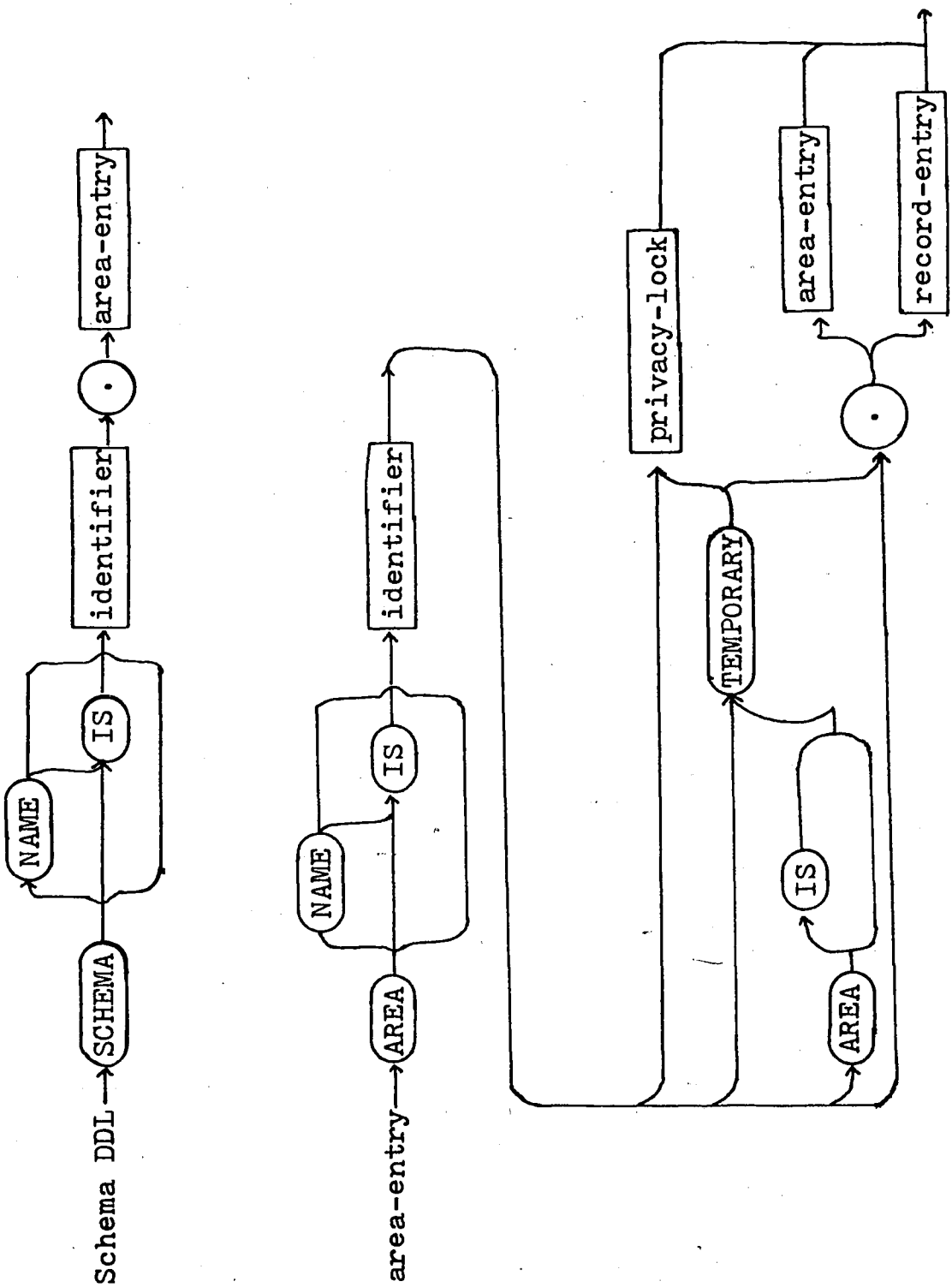

Fig. 3. The Syntax Diagrams of Schema DDL

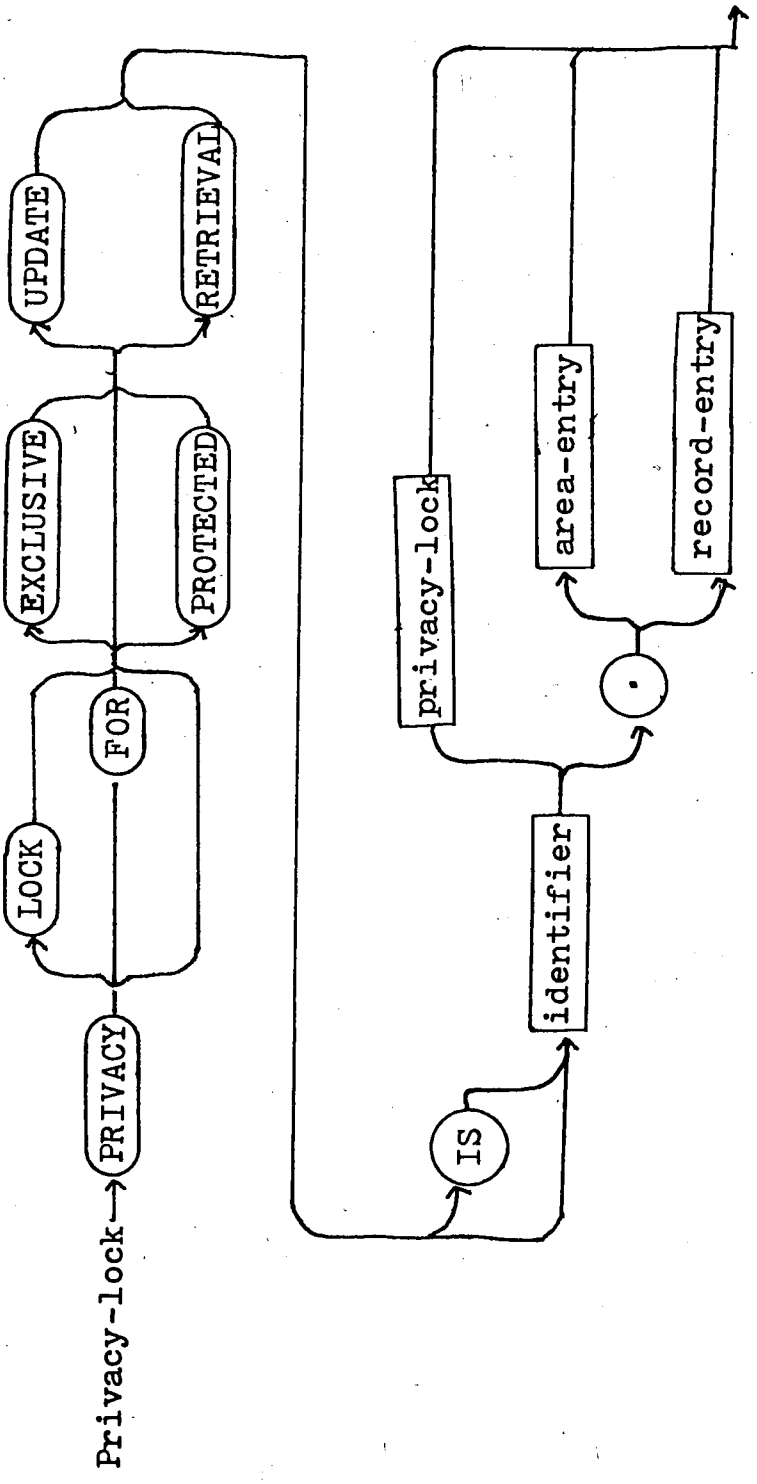

Fig. 3 (Continued)

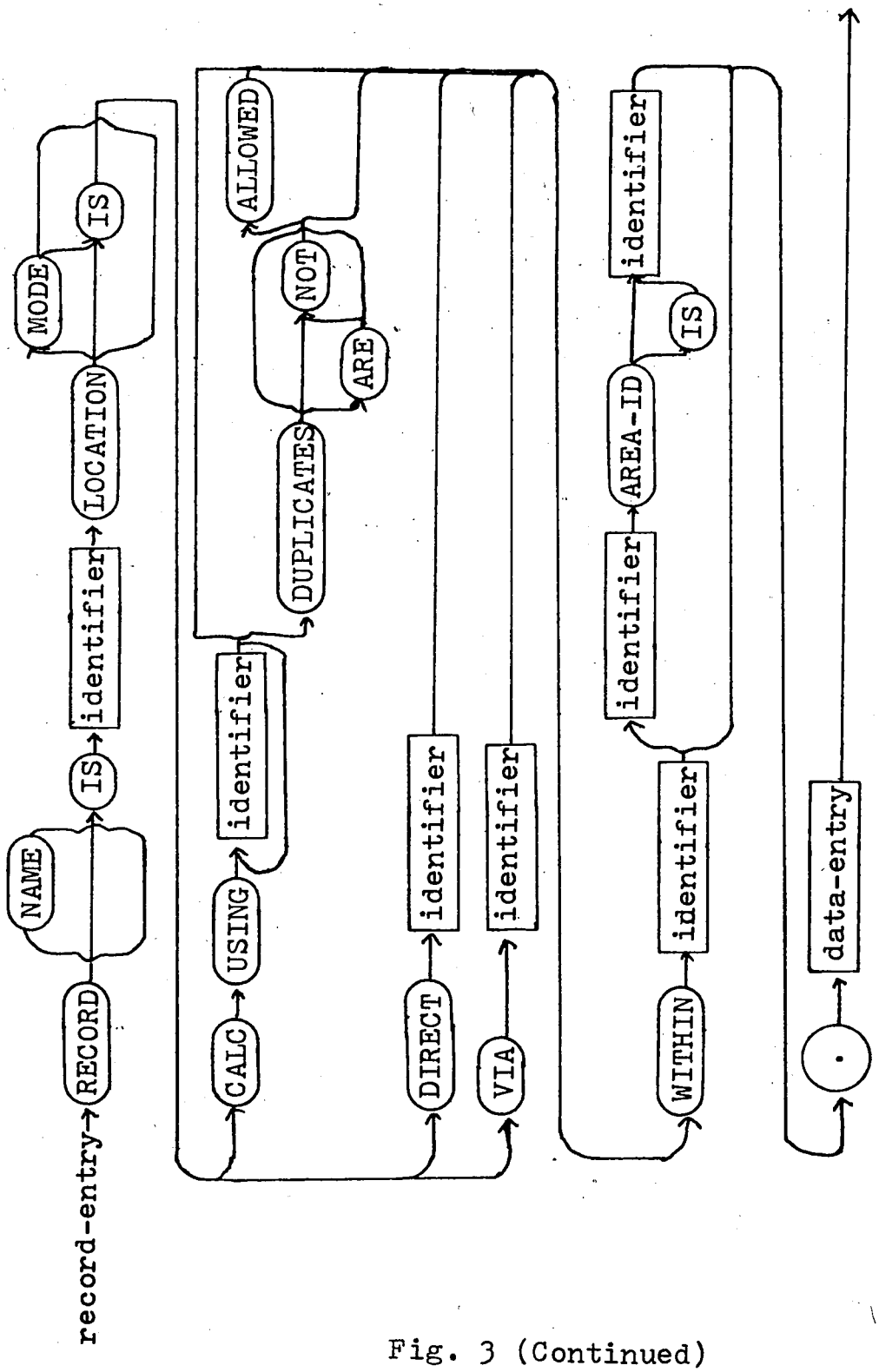

Fig. 3 (Continued)

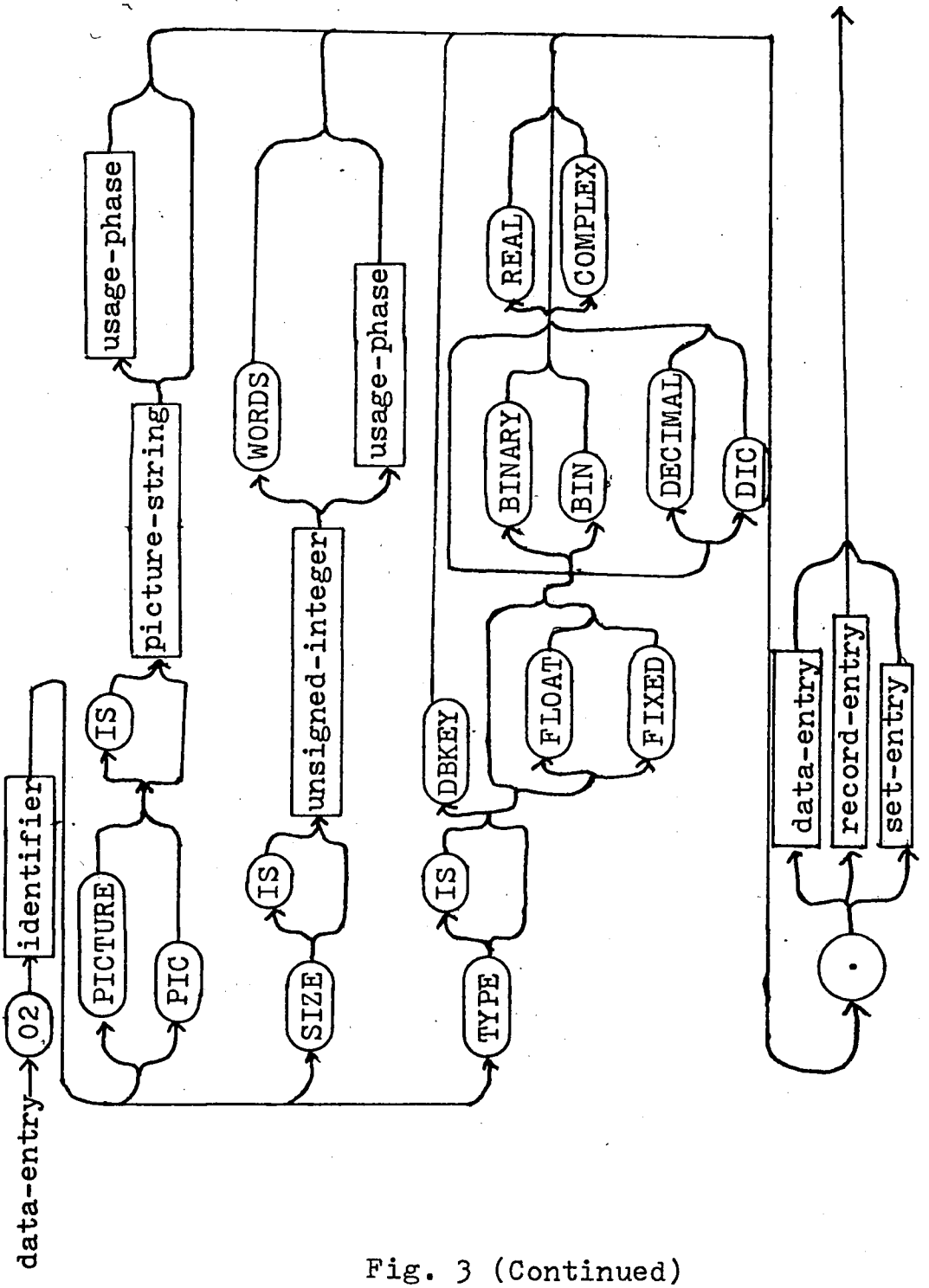

Fig. 3 (Continued)

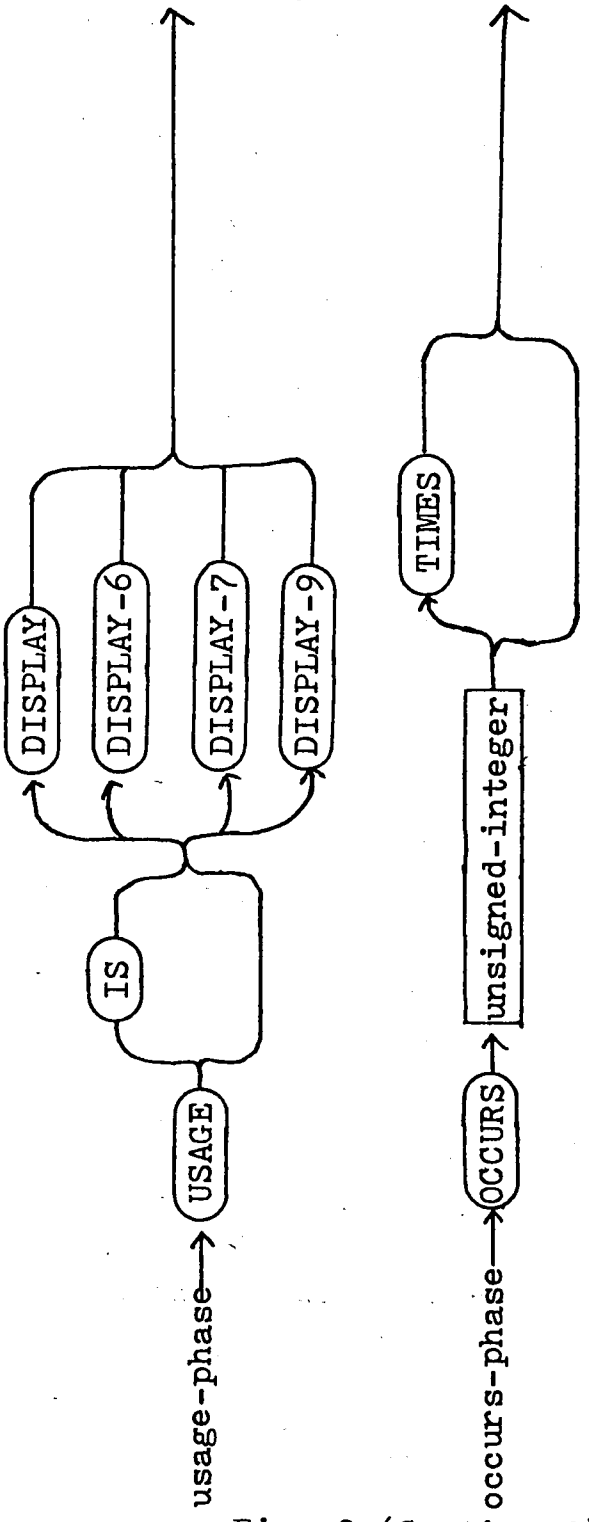

Fig. 3 (Continued)

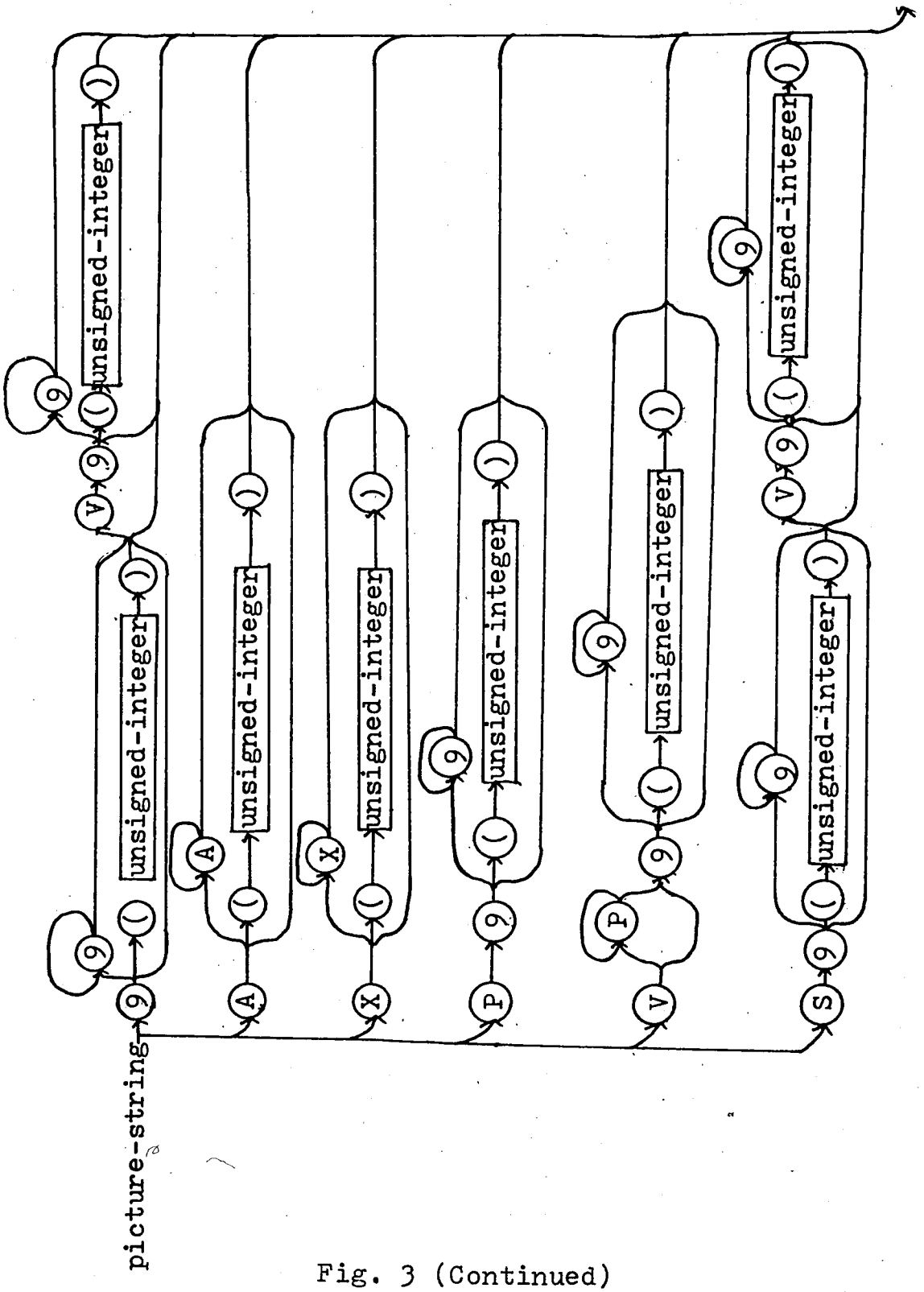

Fig. 3 (Continued)

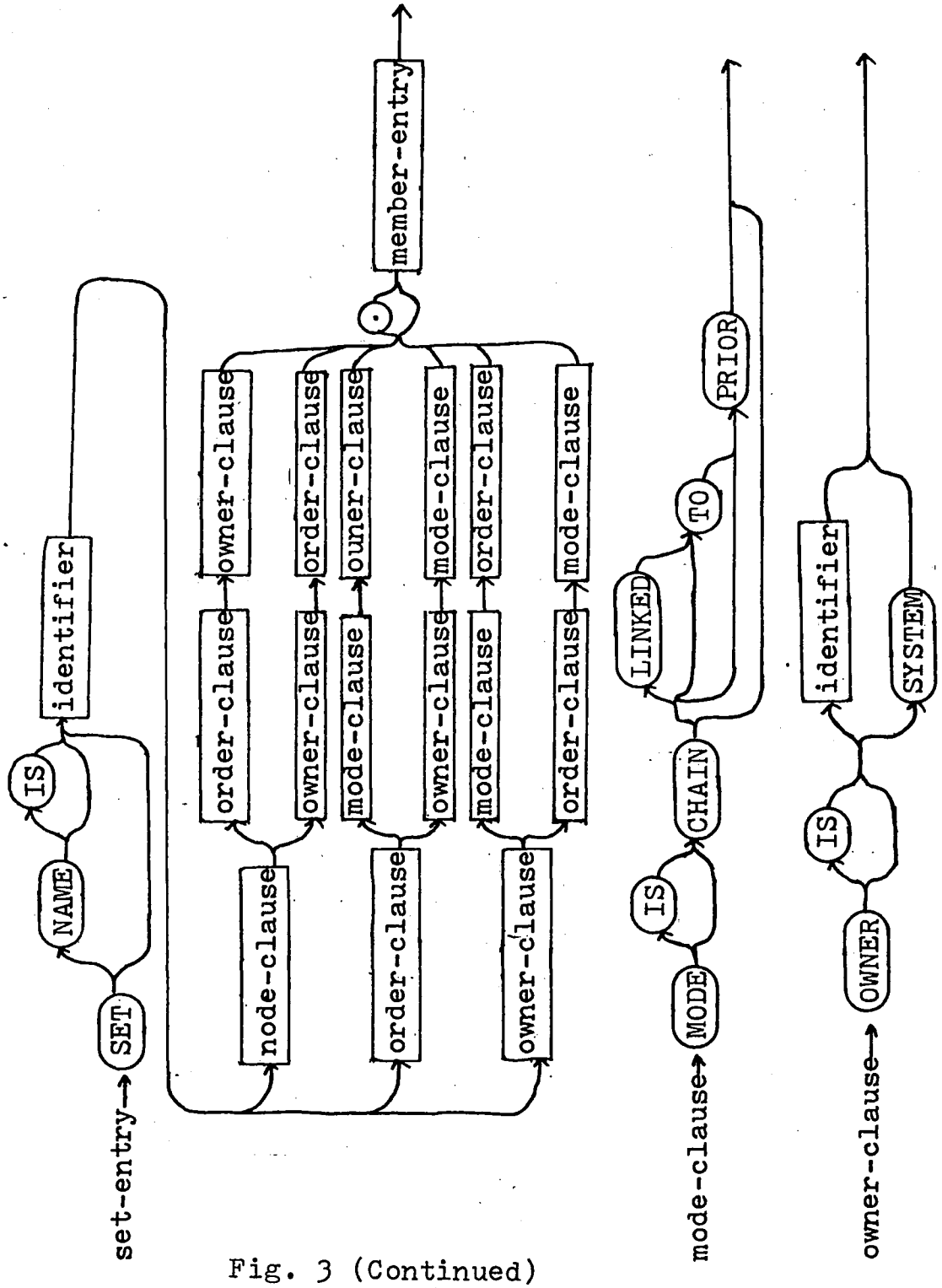

Fig. 3 (Continued)

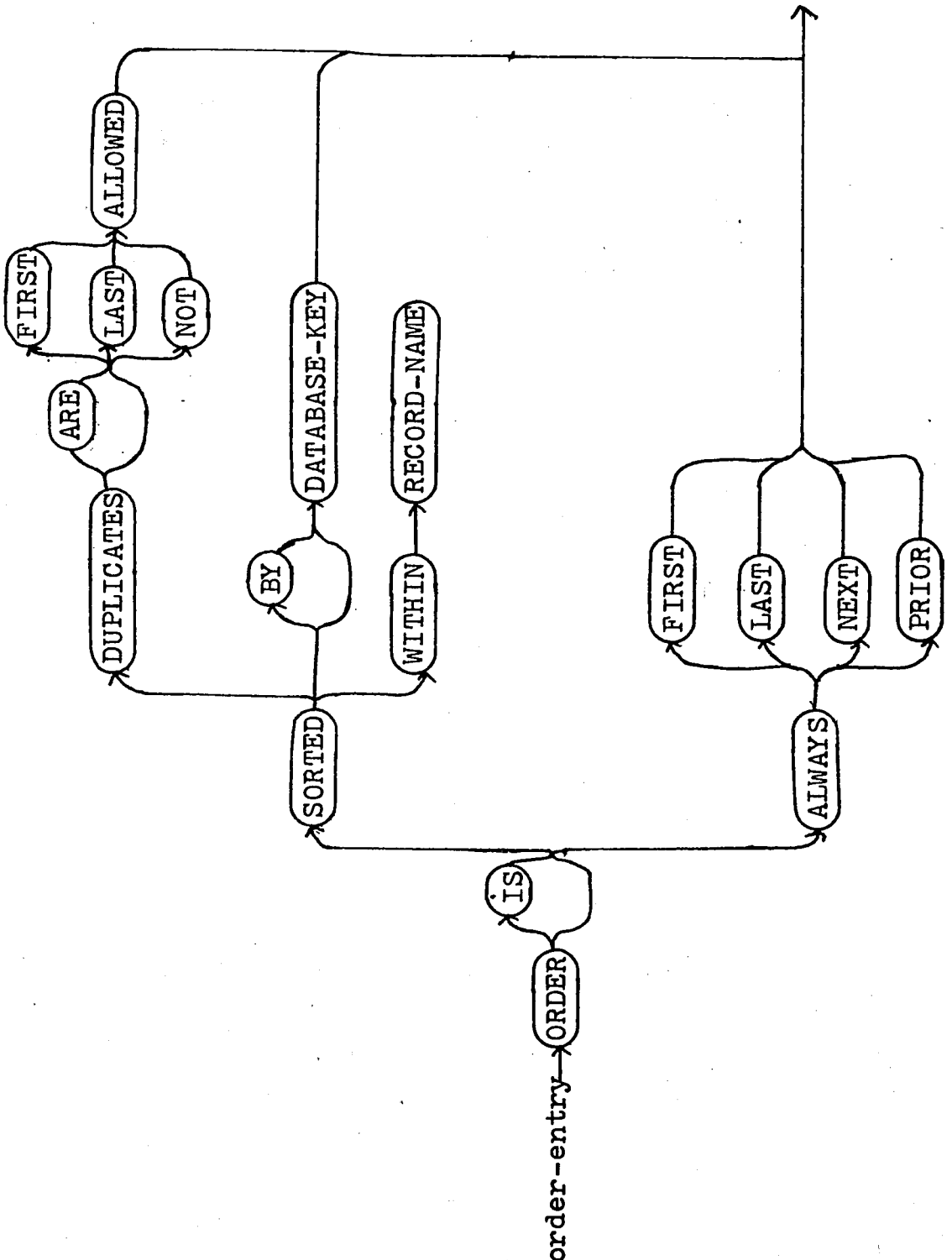

Fig. 3 (Continued)

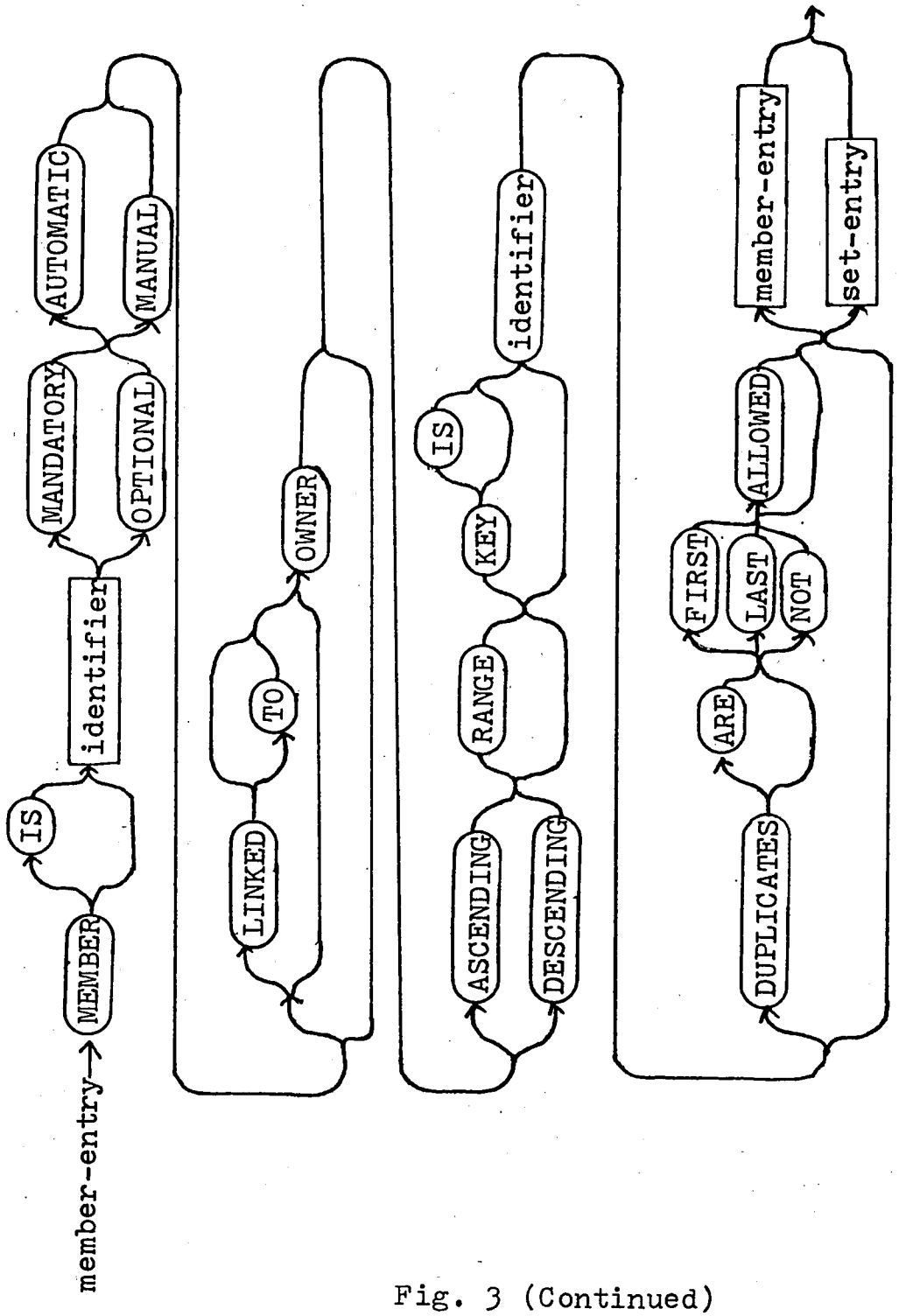

Fig. 3 (Continued)

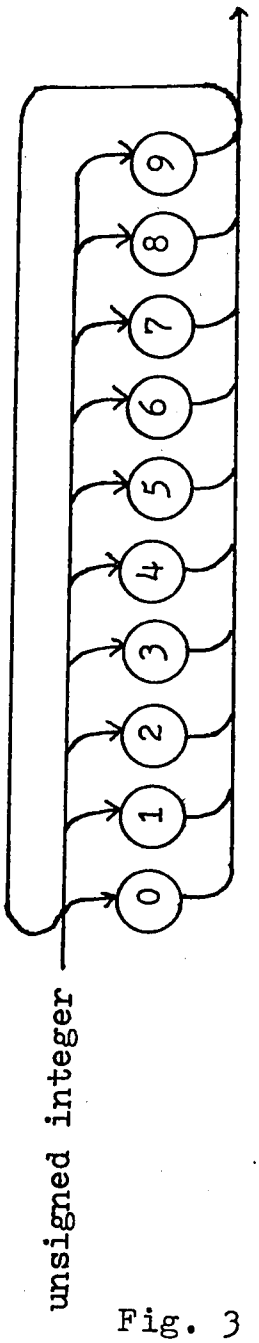

Fig. 3 (Continued)

presented in Fig. 3 in the form of 15 diagrams. Fig. 3 is a convincing example of the expressive power of these graphs which allow formulation of the syntax of an entire Schema DDL in such a concise and readable form.

STEP 2: Translate The Given Syntax Into a Program

A program which accepts and parses Schema DDL is readily derived from its deterministic syntax graph. The graph essentially represents the flowchart of the program. In developing this program the seven given translation rules(Ref. 9, 292) were strictly followed.

```
PROCEDURE GETSYM;

   VAR
      I, J, K: INTEGER;
```

Fig. 4: The Scanner used to get next symbol

```
PROCEDURE GETCH;

   BEGIN
      IF CC = LL
      THEN
         BEGIN
            LL := 0;
            CC := 0;
            WHILE NOT EOLN(INPUT) DO
               BEGIN
                  LL := LL + 1;
                  READ(CH);
                  WRITE(CH);
                  LINE[LL] := CH
               END;
            WRITELN;
            LL := LL + 1;
            LINE[LL] := ' ';
            READLN;
         END;
      CC := CC + 1;
      CH := LINE[CC];
   END (*GETCH*);

BEGIN (*GETSYM*)
   EOS := FALSE;
   WHILE CH = ' ' DO
      GETCH;
   K := 0;
   REPEAT
      IF K < AL  THEN
         BEGIN
            K := K + 1;
            A[K] := CH;
         END;
```

Fig. 4 (Continued)

```
      GETCH;
   UNTIL CH = ' ';
   IF (A[K] = '.') AND (K > 1)
   THEN
      BEGIN
         A[K] := ' ';
         K := K - 1;
         EOS := TRUE;
      END
   ELSE
      IF (A[K] = '.') AND (K = 1)  THEN
         EOS := TRUE;
   IF K >= KK
   THEN
      KK := K
   ELSE
      REPEAT
         A[KK] := ' ';
         KK := KK - 1;
      UNTIL KK = K;
   ID := A;
   I := 1;
   J := NORW;
   REPEAT
      K := (I + J) DIV 2;
      IF ID <= WORD[K]  THEN
         J := K - 1;
      IF ID >= WORD[K]  THEN
         I := K + 1;
   UNTIL I > J;
   IF (I - 1) > J
   THEN
      SYM := WSYM[K]
   ELSE
      SYM := IDENT;
END (*GETSYM*);
```

Fig. 4 (Continued)

The basic symbols are sequences of characters, such as SCHEMA,RECORD,etc. As in Fig. 4 a scanner is used to take care of the representational or lexical aspects of the input sequence of symbols. The scanner is conceived as a procedure GETSYM whose task is to get the next symbol. The scanner serves the following purposes:

a. It skips separators(blanks).
b. It recognizes reserved words,such as AREA, PICTURE, etc.
c. It recognizes non-reserved words as indentifiers.The actual identifier is assigned to a global variable called ID.

In order to scan the input sequence of characters, procedure GETSYM uses a local procedure GETCH (see Fig. 4) whose task is to get the next character. Apart from this main purpose,procedure GETCH also

a. Recognizes and suppresses end-of-line information.
b. Copies the input (Schema DDL) into output file, thus generating a program listing.

The Scanner constitutes the necessary one-symbol lookahead. Moreover, the auxiliary procedure GETCH represents an additional lookahead of one symbol plus one character.

2.3 Error Recognition and Diagnostics

The DDL parser was written on the basis of the syntax of the Schema DDL. It can detect 38 errors. Fig. 5 lists a set of possible diagnostic messages. During the parsing, as soon as an error is discovered,the error message will be printed out and the excution is stopped. A user can use the system editor,EDIT,to make the necessary correction before a rerun.

(1). "SCHEMA" expected
(2). invalid schema-name
(3). "." expected at previous statement
(4). "AREA" expected
(5). invalid area-name
(6). duplicate identifier
(7). unexpected identifier
(8). "TEMPOARY" expected
(9). invalid area-name or unexpected "." at previous statement
(10). privacy-lock (a identifier) expected
(11). "TEMPOARY" expected or "." expected at previous statement

Fig. 5: Error messages for schema parser.

(12). unexpected "." at previous statement
(13). record-name expected
(14). "LOCATION" expected
(15). "DIRECT" or "CALC" or "VIA" expected
(16). "USING" expected
(17). "WITHIN" expected
(18). "02" expected
(19). a data-name expected
(20). "PICTURE" or "PIC" or "SIZE" or "TYPE" expected
(21). picture-string expected
(22). invalid picture-string
(23). a integer expected
(24). "WORDS" or "USAGE" or expected
(25). "USAGE" or "OCCURS" EXPECTED
(26). "DISPALY" or "DISPALY-6" or "DISPALY-7" or "DISPLAY-9" expected
(27). "OCCURS" expected
(28). "MODE" or "ORDER" or "OWNER" expected
(29). "CHAIN" expected
(30). "ORDER" or "OWNER" expected
(31). "OWNER" expected
(32). "SYSTEM" or a record-name expected
(33). record-name not defined in Record Entry
(34). "MEMBER" or "SORTED" expected
(35). "MEMBER" expected
(36). "MODE" or "OWNER" expected
(37). "ORDER" or "MODE" expected
(38). a record-name expected

Fig. 5 (Continued)

As soon as a symbol is read, it performs the binary search on the array of reserved words(see Fig. 4). IF a symbol is not in the array it is considered as an identifier.

Identifiers are stored in a binary tree structure(see Fig. 6). If an identifier name is already in use(found in the tree) then a syntactical test is performed to determine whether duplication is allowed.

```
PROCEDURE CHECKDULPIDENT(VAR TREE: LINK;
VAR DULP: BOOLEAN; NEWINDENT: SYMB);

 BEGIN
    DULP := FALSE;
    IF TREE = NIL
    THEN
       BEGIN
          NEW(TREE);
          WITH TREE^ DO
             BEGIN
                LEFT := NIL;
                RIGHT := NIL;
                DATA := NEWIDENT;
             END;
       END
    ELSE
       WITH TREE^ DO
          IF NEWIDENT < DATA
          THEN
             CHECKDULPIDENT(LEFT, DULP, NEWIDENT)
          ELSE
             IF NEWIDENT > DATA
             THEN
                CHECKDULPIDENT(RIGHT, DULP, NEWIDENT)
             ELSE
                DULP := TRUE;
 END (*CHECKDULPIDENT*);
```

Fig. 6: Binary tree used to store identifiers.

The DDL parser can detect only one error, during one runtime; that is, no error-recovery is performed since the rest of the DDL is affected by the error.

2.4 Parsing Result

Besides syntax recognition, the parser can also recognize and store the following information:

a. AREA names.
b. RECORD names.
c. LOCATION MODE of a record type,i.e. how a record occurrence is physically stored in the data base.
d. Area where occurrences of the record will be stored.
e. Data-item and data-aggregate sizes based on the calculation.
f. Identification of relationships between record types, i.e. which is the owner and which is/are the member(s).
g. The overhead for every record type.

There will be a more detail discussion on c,e,f and g in next section. All the information is stored in two different linked lists.

2.5 Record Occurrence Size

A record occurrence contains (1)data (2)record overhead to hold all the linkage information. Separate discussions on how to calculate the storage for them are presented here. And these two calculation are done by the DDL parser automatically.

A. Data Size:

There are three formats which one may use to name a data-item or data-aggregate (see Fig. 2). In the DBMS Administrstor's Procedure Manual (Ref. 3, 4-8:4-10) there is a complete explaination of how a data-item or a data-aggregate is calculated.

In a record-type, there are several possible kinds of data entry formats. Where different formats use different units for the storage, the solution to this situation is to translate all the different units into the the lowest unit of measure - the bit. After summing up the number of bits for all the data-items in a record type, the corresponding number of words (1 word = 36 bits) is calculated.

B. Record Overhead

For each record type defined for the data base, the following criterion may be used to decide the amount of overhead in words for each record occurrence:

a. location mode is CALC	1 word
b. owner of set types	1 word per set
c. member of set types	1 word per set
d. LINKED TO OWNER in the set types	1 word per set
e. LINKED TO PRIOR in the set types	1 word per set

The overhead is the result of the calculation from the above plus 1 word for the line header.

3. USER'S MANUAL

3.1 Introduction

DBSAP is a PASCAL program aimed at aiding the Data Base Adminstrator in the development of his Device Media Control Language (DMCL), e.g., to enable the user to select individual areas, assign them to files, and allocate storage to them. There are two type of DMCL entries that thc user can use to:

a. Specify parameters that apply to the SCHEMA as a whole, e.g., the number of records on a page and the name of the journal (Environment Entry).

b. Assign area to files and specify the physical characteristics of these file (Area Entry).

DBSAP consists of two parts. The first part is a parser that reads the user's Schema DDL as input and examines the syntactical structure of the Schema DDL statements. Besides syntax recognization, the parser can also recognize and

store all the information needed to be used to develop user's DMCL statements. Refer to the discussion in the previous chapter. The second part is an iteractive program that helps the user to develop his DMCL statements by asking him to provide more information about the data base and its usage in the future.

3.2 Running the DBSAP

a. Use the system editor, EDIT, to create the Schema DDL file. Instructions on how to use the system editor is available from Decsystem-20 User GUIDE(Ref. 4).

b. Once the user has created the file containing the description of the schema, he can run the DBSAP program to develop his DMCL statements. To do so, the user must run DBSAP and give a command string as follows:

```
@EXECUTE (FROM) DBSAP.PAS
 LINK:   Loading
 [LNKXCT DBSAP Execution]
 INPUT     : file-name.1
 OUTPUT    : TTY:
 DMCL      : file-name.2
```

File-name.1 is the name of the file that has been assigned to the Schema DDL. The User must type this in when "INPUT :" appears. Type in "TTY:" after "OUTPUT :" appears. This will print out on the terminal you are on your input file the error messages(if any), the parsing result and interative questions. When the "DMCL :" appears, type in a variable name file-name.2 that will be assigned to the file that will store the output DMCL file.

c. During the parsing, as soon as an error is discovered, the error message will be printed out and the execution will be terminated. A user can use the system editor, EDIT, to make the necessary corrections before a rerun.

d. When the parsing is completed, and no error is detected, the following information will be listed:

(1). Number of areas
(2). Area names
(3). Number of records
(4). Record names
(5). Record data size
(6). Record overhead
(7). Total record size
(8). Details of overhead

e. After the above information has been printed out. The user is requested to assign the occurrences of each record type and its expected growth percentage. It is important to emphasize, that the above information should be as precise as possible, and it it should be obtained after conducting careful system analysis studies.

f. DBSAP will use the information obtained from the parser and from step e to provide the user with a table of the total occurrences of each record type.

g. DBSAP will help the user develop his DMCL statements by giving him all the alternative statements according to DMCL syntax (Ref. 3), and asking to choose the ones which are more suitable to his needs. If the input for the user's choice is valid, a DMCL statement followed by an "[OK]" will be printed out, to indicate that the statement has been accepted and has been stored in the output file-DMCL. If the input is invalid, one of two possible situations may occur. First, if the error is detected by DBSAP then it will give the user the warning "wrong input! try again". For example,

if the choice is "<1 or 2>" and it reads "3" as input, then the same question will be asked. The other situation is when an error is detected by the PASCAL compiler. In this case, the excution will stop, and an "? invalid format". message will be printed out.

h. When DBSAP gives the message that all the DMCL statements have been completed, all these statements are stored in the file which the user specified for the output file-DMCL. The user has to append the Schema DDL file Subsequently file. Subsequently, the user can use EDIT to key in the Sub-Schema statements.

3.3 The formulas used in storage calculations

The following is the list of the formulas used in storage calculations:

a. ph = 2
b. po = ph + nocalc
c. as(i) = lps(i) - po(i)
d. grpp(i) = as(i) / srec(i)
e. lrpp[i,j] = as(i) / rs(i)
f. to[i,j] = poc[i,j] * (1 + perct[i,j] / 100)
g. pages[i,j] = to[i,j] / lrpp[i,j]

i. ar(i) = pages[i,j]
j. lp(i) = fp(i) + ar(i) - 1

where i: i = 1,....,n i is the number of the areas in this data base.

[i,j]: The j'th record type in i'th area.

ph: Page Header.

po: Page Overhead.

nocalc: Number of CALC chains.

as: Available Size in a certain area that can be used to store Record occurrences.

lps: Logical Page Size is the maximum number specified by the user that can be used to stored record occurrences.

grpp: RECORD-PER-PAGE is the maximum number of records that can be stored on a page for an individual area.

srec: The smallest record size among an individual area.

lrpp: The number of record occurrences which can be stored in one page for an individual record type.

rs: Record Size.

to: Total Occurrences of an individual record type in an area.

poc: The occurrences of an individual record type.

perct: Growth Percentage of an individual record type.

pages: The number of pages required to store all the occurrences of an individual record type.

ar: The number of pages for an individual area.

lp: Last Page of an individual area.

fp: First Page of an individual area.

4. CONCLUSION

This thesis has presented a package designed to help the data base designer in the selection of the appropriate storage allocation for his data base. Using this package helps to avoid arbitrary storage assignment which may most likely lead to waste in storage or to insufficient storage as may be required by future expansion.

Even though, DBSAP is designed to help the data base designer to develop his DMCL statements, it does not provide an optimization module for the storage allocation. The last aspect is a direction for further investigation and development.

REFERENCES

1. Aho, Alfread V. and Ullman, Jeffery D. The theory of Parsing Transltion and Compiling, Volume I: Parsing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

2. Conference on Data Systems Languages (CODASYL). Data Base Task Group Report April 1971, Association for Computing Machinery, New York, 1971.

3. Digital Equipment Corp. DATA BASE MANAGEMENT SYSTEM Admistrator's Procedure Manual, Maynard, Massachusetts,1977.

4. Digital Equipment Corp. DECSYSTEM-20 User's Guide, Maynard, Massachusetts, 1978.

5. Knuth, D.E., "Top-down syntax Analysis",Acta Information, 1, No.2 (1971), pp. 79-110.

6. Lewis, P.M. and Stearns, R.E., "Syntax-directed Transduction", J. ACM, 15, No.3 (1968), pp. 465-88.

7. Martin, James. Computer Data-Base Organization (2nd ed), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

8. Olle, T William. The codasyl Approach to Data Base Management, John Wiley & Sons, New York, 1978.

9. Wirth, Nicklaus. Algorithms + Data Structure = Programs, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

VITA

Tung-Sheng Lai was born on Octeber 28, 1954 in Ping-Tung, Taiwan, Republic of China. It was there that he received his pre-collegiate education. From 1972 to 1976, he attended the College of Chinese Culture in Taipei, Taiwan. In 1978, he decided to continue his education with graduate study in the United States. In the fall of the same year he begin his studies in the Division of Computing and Information Science at Lehigh University.