To be presented at the Association for Computing Machinery Pacific 75 Conference, San Francisco, Ca., April 17-18, 1975.

LBL-3601
c. 1

A SYNTAX-ANALYSIS APPROACH TO DATA RELIABILITY

Gerry Litton, C. M. Lederer and Leo Vardas

November 22, 1974

Prepared for the U. S. Atomic Energy Commission under Contract W-7405-ENG-48

LBL-3601
c. 1

LBL-3601

A SYNTAX-ANALYSIS APPROACH TO DATA RELIABILITY

by

Gerry Litton, C. M. Lederer, and Leo Vardas

ABSTRACT

A unique approach has been developed to check data for syntax errors - a process that could well be likened to proofreading.

The key to the process is the use of:

(1) A syntax analyzer that was developed primarily for use as a compiler parser.

(2) The associated language used to specify syntax.

The data checking proceeds in two steps. First, the syntax rules to which the data must conform are written in a syntax-oriented language. They are then fed to a processor which produces a set of tables which are the encoded syntax rules. Second, the data to be check is fed to a syntax analyzer, which checks it using the encoded syntax rules as a template.

The specific data for which this process has been developed is that which constitutes the <u>TABLE OF ISOTOPES</u>, published periodically by the Lawrence Berkeley Laboratory.

INTRODUCTION

The TABLE OF ISOTOPES is a compilation of information on the properties and radiations of radioactive isotopes, which is published periodically by the Lawrence Berkeley Laboratory. With each edition, the magnitude of contained data has grown; the last edition contained several hundred-thousand pieces of numeric data on approximately 1500 isotopes - this amounts to several million characters of print. Although this amount of data presents no serious problem from the standpoint of computer storage or handling, it does represent major problems in abstracting, copying, input, and proofreading. The computerization of the data collection system has been discussed elsewhere;(1,2) it is the purpose of this paper primarily to discuss that aspect of the computer system which deals with the proofreading problem.

The checking of data for syntax accuracy consists of two basic parts. First, the syntax rules to which the data must conform are written in a syntax-oriented language. Then, the data is fed to a program which checks it against these syntax rules.

It is expected that this system will eliminate the need for manual proofreading of data for the TABLE OF ISOTOPES project.

I. TABLE OF ISOTOPES DATA

The amassing of data for inclusion in the TABLE OF ISOTOPES begins with the abstracting from suitable references - such as journals and books - or pertinent information. This data is massaged to conform to a standard format and then entered on a special keyboard which is linked directly to the Computer Center at the Lawrence Berkeley Laboratory. This keyboard, which is described elsewhere,(1) has a greatly expanded character set (12 bits), including the Greek and Roman alphabets, upper and lower cases, and a variety of special symbols. Figure 1 shows the input console in use, and Figure 2 shows a schematic drawing of the keyboard layout. During the data-entering process, the operator has a visual display and may edit the data to correct errors and make any other necessary changes.

Finally, the data is sent to one or more of the mass-store devices attached to the Computer Center. As it is stored, the data consists of many items, each of which describes a particular piece of data for a particular isotope. For example, a given item might contain a measurement of the half-life of a particular isotope.

II. THE PRE-PROCESSOR

The general approach to the use of a table-based syntax analyzer is as follows. First, the syntax rules for the text to be analyzed are written in the language appropriate to the syntax compiler being used (the name of the compiler used in this work is -Meta-). These rules are then fed to the compiler, and a set of syntax tables is produced. Finally, the desired text, or character stream, is fed to the associated parser along with the syntax table.

The parser scans the input stream and performs a comparison between the stream and the rules embodied in the syntax table. A -true- or -false- value is returned, depending on whether or not the character stream conforms to the syntax rules. This complete process is shown schematically in Figure 3.

The most direct approach to analyzing TABLE OF ISOTOPES data would be to write down all of the possible syntaxes allowable for all of the different types of data and then to generate one syntax table. Each item of data could then be fed to the parser and tested for a valid match with any of the "rules" encompassed in the syntax tables.

Although this method is the most direct, it is unfortunately also extremely time-consuming; inasmuch as the TABLE OF ISOTOPES encompasses many different types of data, the corresponding syntax table would be huge, and on the average, something like one-half of these tables would be scanned for each data item checked.

Therefore, a more complex, but much less time-consuming, method of using the syntax analyzer has been developed; and the following section describes this method.

Included with each TABLE OF ISOTOPES data item are several flags, which supply key pieces of related information, such as the reference number and the method of measurement. One flag of particular importance is that which defines the type of data contained in that item; and it is this flag which allows a computer program to identify a particular data item and to choose the appropriate syntax table from a set of such tables.

Therefore, instead of constructing one huge syntax table encompassing all possible data types, a set of syntax tables is constructed - one table for each type of data contained in the data bank. Furthermore, since each table describes only a single type of data, it is comparatively small.

To facilitate the handling of many different syntax tables, a syntax pre-processor is utilized. Input to the pre-processor is a number of sets of syntax rules, each set describing a particular type of data. For each set of syntax rules input, the pre-processor calls the Meta compiler, which processes the rules and returns a syntax table. This table is then stored in a special random-access array containing all of the syntax tables, and the address is stored in a syntax table index. Later, when a particular piece of data is being checked for correct syntax, this index will be used to find the appropriate syntax table.

A secondary function of the pre-processor is the processing of lists, for use with the syntax analyzer (see Section III for a discussion of the use of lists in syntax analysis). For each list input, the pre-processor packs it and stores it in a special random-access array reserved for lists, and an entry is made in a list index. When a particular list is to be used during syntax analysis, this index will serve to locate that list array.

When the pre-processor is finished processing syntaxes and lists, the random-access arrays, along with their indexes, are stored on one or more of the mass storage devices at the LBL Computer Center for later use.

All input to the pre-processor -both syntax rules and lists- is done via the input-editing system described in the previous section.

III. THE SYNTAX LANGUAGE

In order to write down the syntax rules to which a given character string must conform, the -Meta- language is utilized. This language was designed specifically to treat syntax descriptions in a natural manner, and as such it is vastly superior to the more scientifically oriented languages such as FORTRAN.

The Meta compiler accepts as input a syntax description written in the -Meta- language; as output, the compiler produces a set of tables which may be used at some later time by a parser to scan a piece of data (or character string) for correctness of syntax.

This section describes the highlights of the Meta language; it is treated elsewhere(4) in more detail.

Basic Elements of the Language

The Meta language description of a particular syntax is written as a sequence of rules. Each rule consists of one or more of the following basic elements.

(1) Terminals

A terminal is a basic unit of the input stream; a set of terminals consitutes the language in which the input stream is written. In this application, the set of terminals are the set of 12-bit characters which are used to write TABLE OF ISOTOPES data.

The term -literal- is defined to be a sequence of one or more terminals. Literals are written in the Meta language simply as the characters themselves; that is, with no special delimiters.

All language elements other than terminals must eventually boil down to terminals, since they are the units of information which directly describe the input stream.

(2) Special Characters

The set of special characters is quite small; they are used to indicate specific language elements, such as repetition, etc. They are discussed further on in this section.

(3) Separators

Individual elements of the language are separated by one or more blanks or by any of the special characters.

(4) Non-Terminals

In order to avoid unnecessary complexities, it is often useful to write a portion of a syntax description in terms other than the primary terminals. These might be groups or sets of terminals in some particular order or they might be complex prescriptions. Any such elements are denoted as "non-terminals". As mentioned above, all non-terminals must eventually boil down to terminals, although the distance between the former and latter might be long and complex.

(5) Rule Name

One example of a non-terminal is the reference, within a rule, to another rule. In this case, the latter is enclosed in quotation marks.

A simple examples will help to show the usage of the Meta language. Consider the following syntax rule to which a character stream, or portion thereof, must conform:

RULE = A ↑ BC

This states that the character string must contain either the literal "A" or the literal "BC"..... the special character "↑" denotes alternation.

It is worthwhile noting that for the purposes of this paper, literals are denoted by capital letters, either singly or in groups. In actual practice, the syntax rules are written using the TABLE OF ISOTOPES input keyboard (see Fig. 2); the set of literals used is a subset of the 12-bit character-set comprising the keyboard.

Repetition is indicated by the notation $\leq$ a,b FACTOR $\geq$, where "a" and "b" are integers. This notation indicates that FACTOR must appear in the character string at least "a" times, but not more than "b" times. Three special notations are used for the following special cases:

(1) $\langle$FACTOR$\rangle$ is equivalent to $\leq$ 1,1 FACTOR $\geq$, and indicates that FACTOR must appear exactly once.

(2) [FACTOR] is equivalent to $\leq$ 0,1 FACTOR $\geq$, and indicates that FACTOR may appear once.

(3) $\leq$FACTOR$\geq$ is equivalent to $\leq 0,\infty$ FACTOR$\geq$.

An example of one rule referencing another is given as follows :

RULE = A ↑ "Rule2"

Rule2 = B [CD]

This states that the character string must contain either the literal "A" or the literal "B", the latter possibly followed by the literal "CD" (note that the brackets imply that the existence of "CD" is optional).

Special Non-Terminals

So far, we have restricted the descriptive elements to literals - that is, sets of one or more characters and to relatively simple non-terminals. Were these the only ways to specify syntax, the language would be extremely restrictive. For example, there may be functional relationships among a certain group of characters in the input stream that might be more conveniently expressed in FORTRAN. One case in point might be if the characters were to represent a floating-point number within a specified range. Although this functionality might be extremely difficult - if not impossible - to describe in the Meta language, it is obviously simple to handle in FORTRAN.

Therefore, the Meta language incorprates a feature which allows it to specify the calling of any number of FOTRAN subroutines. It should be noted that although the specification for calling a subroutine is made via the Meta language, the actual call will be made by the parser at the time when the data is being analyzed.

Consider the following example:

RULE = **NUMS**

This specifies that, at the current position in the input stream, the characters must conform in syntax to whatever functionality is specified by the subroutine NUMS. As indicated here, a subroutine is specified by writing its name in boldface (on the 12-bit-character keyboard, -boldface- is one of the standard fonts).

Prior to execution, all such subroutines which are to be utilized are assembled and attached to a driver program which will call the syntax analyzer to check the desired data (see Section IV).

One special subroutine is treated in a slightly different manner because of its particular requirements. Suppose that at some point in the syntax description, one wishes to specify that the character string must at that point be one of a large list of literals. One could write the syntax specificiations as follows.....

Rule = Lit1 ↑ Lit2 ↑ Lit3 .... where Lit1,Lit2,Lit3, etc. are the items of the list.

If the list is lengthy, and in particular, if the list occurs in several different syntaxes, it is convenient to remove it from the syntax description itself and instead to call a FORTRAN routine which will perform a table-lookup within the list, the list having been previously defined.

Consider the following example:

RULE = *list*

This specifies that at the current position in the character string, the characters must contain one item from the list -*list*-. The Meta compiler treats a string of italicized literals as a list name and sets up a call to a special FORTRAN subroutine. This routine will find the list from a previously-compiler group of lists and then perform a comparison between the input stream at the current position and the list. A -true- or -false- value will be returned, depending on whether or not a match is found.

Again, it should be emphasized that the actual subroutine call is made by the parser when the data-checking is performed.

All lists to be utilized by the parser are compiled prior to the data-checking phase. They are entered via the input-editing system and processed and stored by the Pre-processor (see Section II and Fig. 4).

IV. SYNTAX ANALYSIS

Data checking is performed by utilizing a driver program which brings together and coordinates all of the miscellaneous parts necessary for syntax analysis.

When execution begins, this driver first reads in, from one of the local mass-storage devices, the syntax tables and array of lists, both of which were generated at some prior time by the pre-processor. Also read in at this time are the indexes associated with the lists and the syntax tables. The program next assembles, again from mass storage, whatever group of data items for which syntax analysis is to be performed. Following these setup operations, the driver then proceeds to process the data items one-by-one.

For each data item read in, the driver first identifies the exact category of data contained in the item. With this piece of information, the driver performs a table lookup and locates the appropriate syntax table from the appropriate array. Next, the character pointer is initialized to the first character of the data array. Finally, a call is made to the parser. In this call is supplied the location of the appropriate syntax table to be used.

The parser then proceeds to check the data against the rules embodied in the syntax table. It returns a value of "true" or "false", depending on whether or not the data was found to conform to the table.

The driver makes the appropriate output comments concerning the data item and then proceeds to the next item.

Other Subroutines

In the description of the Meta language (see Section III), reference was made to two types of FORTRAN routines utilized in writing syntax description, and which are callable by the parser. One of these routines is designed to check the input data against a specified list. Other routines are built to check the input data for various functionalities. For example, one routine simply examines the pointer to see if the end-of-data has been reached. Another routine checks the data for a valid decimal number - still another routine checks the data for a valid isotope symbol.

All of these routines are called by the parser during syntax analysis, and they are therefore included as part of the driver package. Figure 4 illustrates the schematic flow of this process.

Character Pointer

At any time during the syntax analysis of a given character string, a special pointer is used to point to the next character to be examined. This pointer is contained within a special common block which is accessible both to the parser and to the FORTRAN subroutines. Whenever any of the latter are called, they begin their examination at the character specified by this pointer. When they finish, they must increment the pointer by the appropriate number.

Thus, suppose that the subroutine "ISOTOPE" is called at some point in the analysis. The purpose of this particular routine is to look at the data beginning at the current setting of the pointer and to ascertain whether a valid isotope symbol exists there. Of course, the number of characters to be examined is at the start indeterminate... that is, it may range from one character to several. If the routine finds a valid isotope symbol in the next three characters, for example, it increments the pointer by that number and returns a "TRUE" value. Otherwise, a "FALSE" value is returned, and the pointer remains unchanged.

CONCLUSION

Using the techniques presented in this paper, it is now possible to produce machine-readable data with a high degree of accuracy in syntax.

As opposed to other languages which are more scientifically-oriented, the use of the -Meta- language not only makes the description of a syntax quite simple and straightforward, but also it greatly simplifies the processes of debugging and syntax modification.

REFERENCES

1. C. Michael Lederer, Computerized Production of a Data Compilation - Table of Isotopes, LBL-2319, October 1973.

2. C. M. Lederer, J. M. Hollander, and L. P. Meissner, UCRL-18530, October 1968.

3. T. E. Cheatham and K. Sattley, Syntax Directed Compiling, Proceedings of the Eastern Joint Computer Conference, AFIPS, Vol. 25, 1964.

4. Leo Vardas, A Parser for Syntax-Directed Translation, LBL-3600 (Unpublished).

FIGURE CAPTIONS

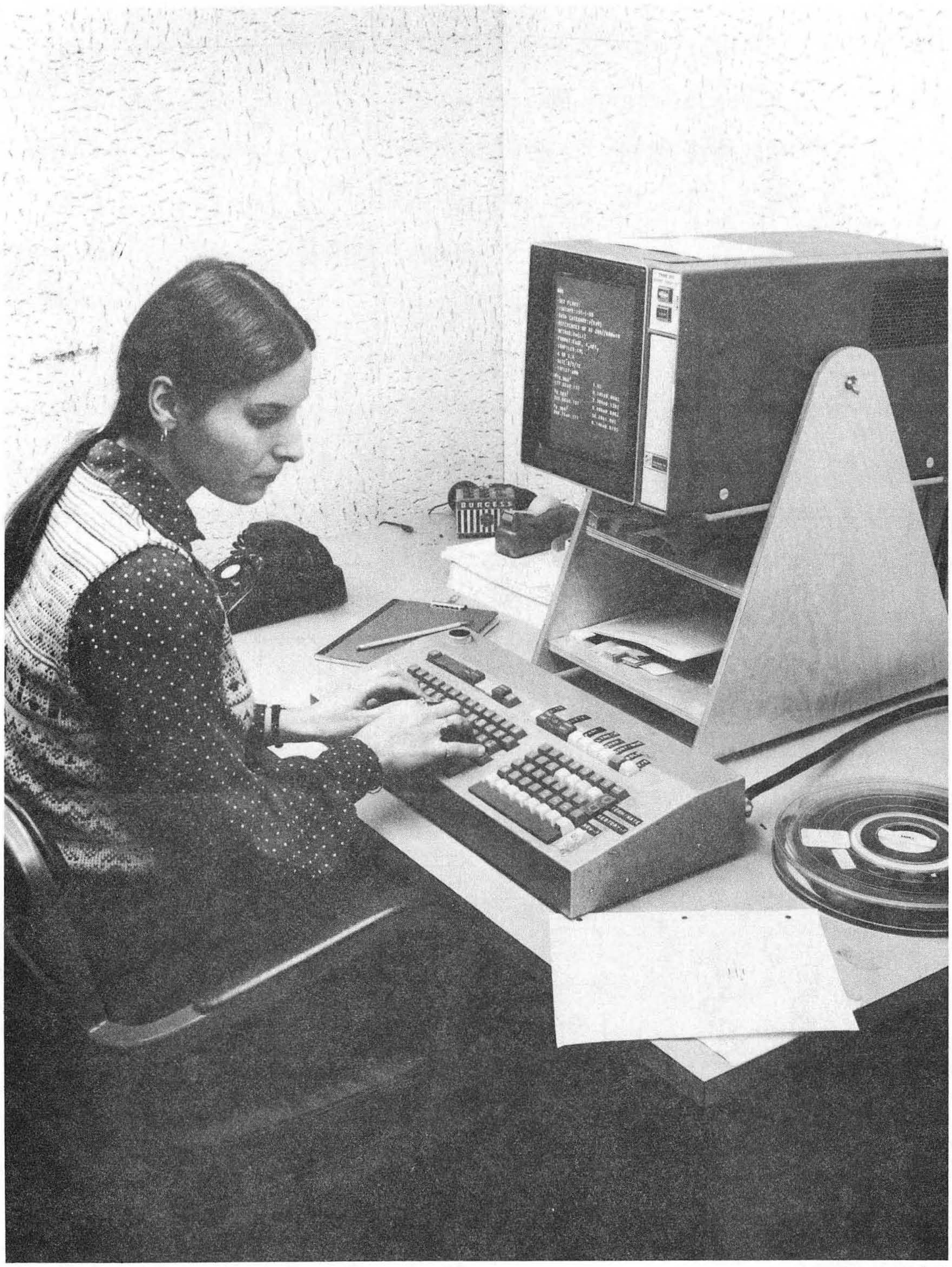
Figure 1. The console of the input-editing system.

Figure 2. Schematic layout of the keyboard.

Figure 3. Simplified schematic of a syntax-analysis system.

Figure 4. A more complete schematic of the syntax-analysis system.

0 0 0 4 3 0 0 0 5 4



XBB 7211-5767

Fig. 1

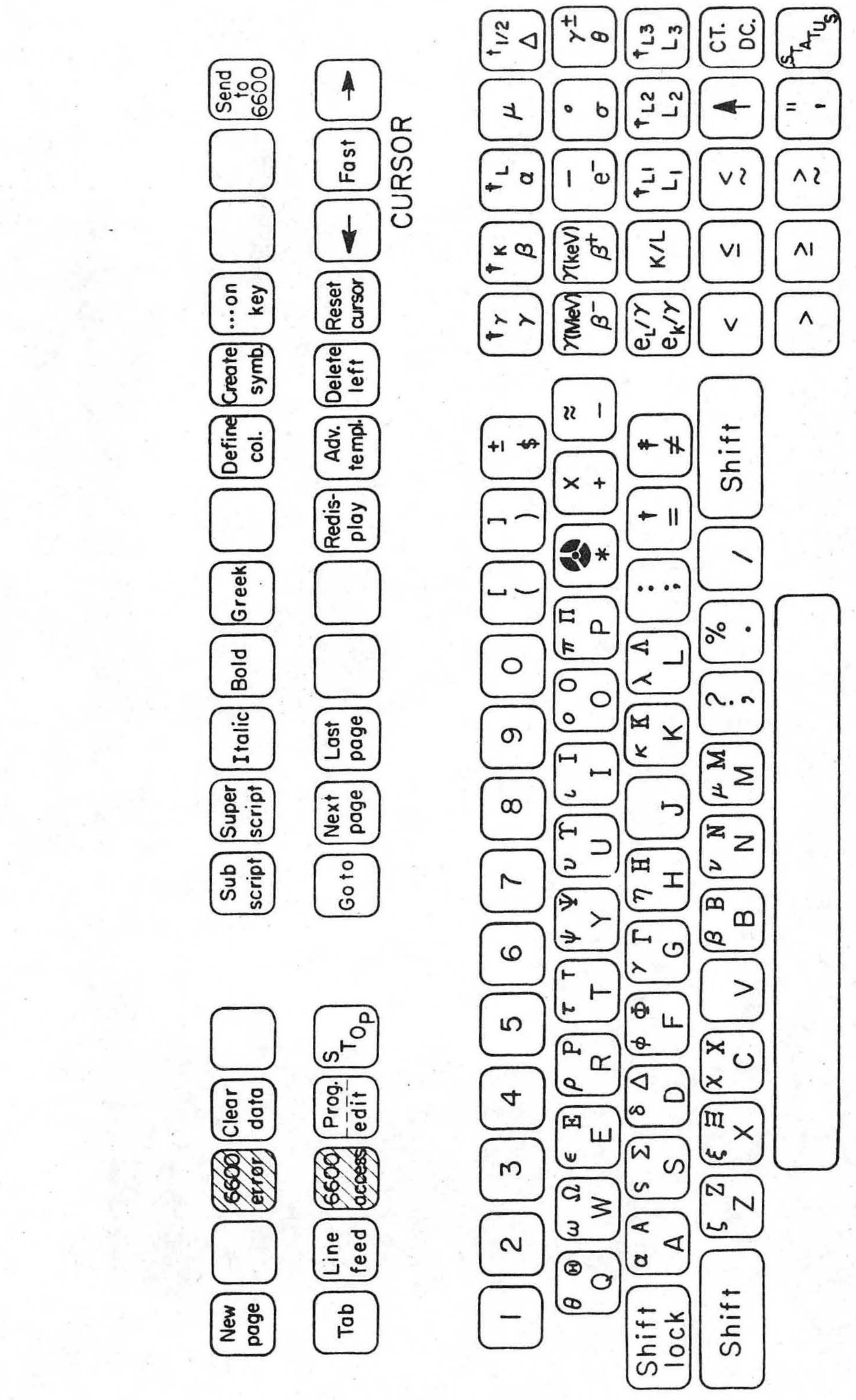
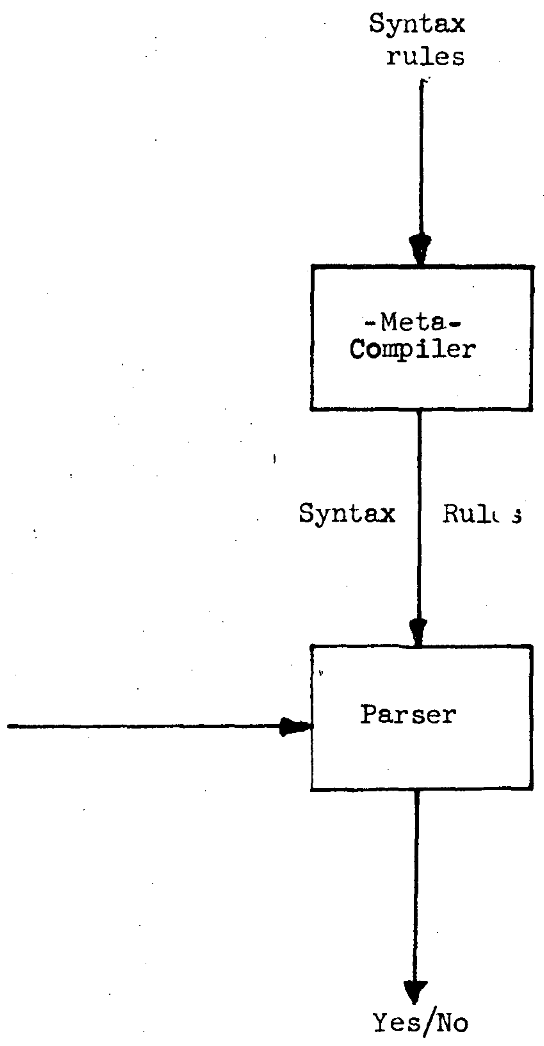
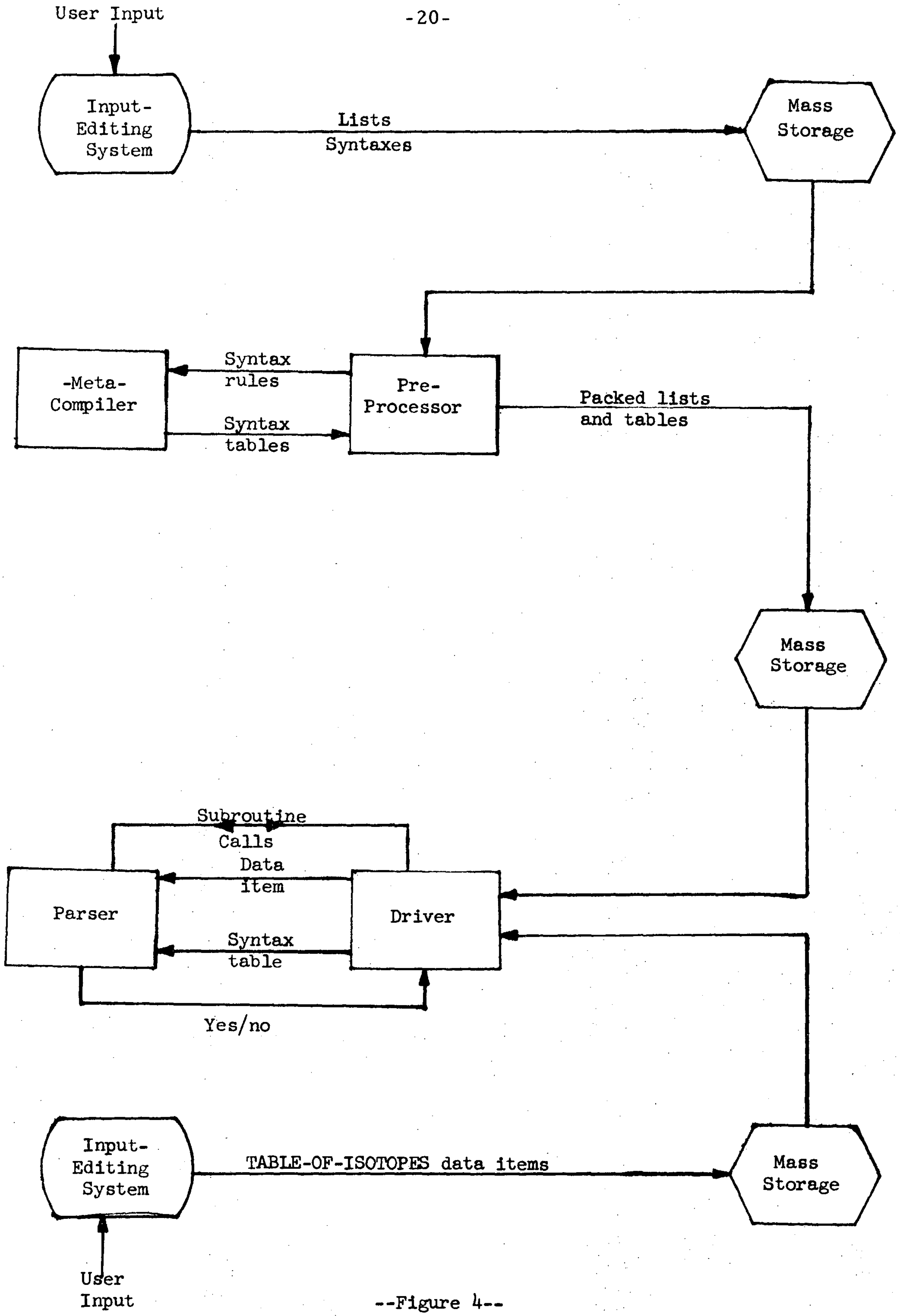

New page
6600 error
Clear data
Sub script
Super script
Italic
Bold
Greek
Define col.
Create symb.
...on key
Send to 6600
Tab
Line feed
6600 access
Prog. edit
STOP
Go to
Next page
Last page
Redis-play
Adv. templ.
Delete left
Reset cursor
Fast
CURSOR
Shift lock
Shift
Shift
K/L
CT. DC.
STATUS

Fig. 2



-- Figure 3 --


User Input
Input-Editing System
Lists
Syntaxes
Mass Storage
-Meta-Compiler
Syntax rules
Syntax tables
Pre-Processor
Packed lists and tables
Mass Storage
Subroutine Calls
Data item
Parser
Driver
Syntax table
Yes/no
Input-Editing System
TABLE-OF-ISOTOPES data items
Mass Storage
User Input

--Figure 4--