1.0 INTRODUCTION

Automatic text generation is the generation of natural language texts by computer. It has applications in automatic documentation systems, automatic letter writing, automatic report generation and HELP subsystems for time-sharing systems. This section introduces the subject and describes the contents of this paper.

The second section describes a basic approach to automatic text generation which makes it possible to generate a relevant text in response to a compact text specification. The structure of the data base (over-lapping tree structures, roughly) from which the text is constructed makes it possible to do this in a non-ad-hoc way because the relationships between the elements of the data base imply sentences and relationships between sentences and groups of sentences.

In the third section, a number of extensions to the basic approach are discussed. These include among others: (a) paragraphing, (b) natural language queries, (c) "flattening" of structures, and (d) elimination of redundancy.

The fourth section of this paper discusses the application of this approach to HELP subsystems for time-sharing systems and on-line documentation systems.

2.0 A BASIC APPROACH

This section describes a basic approach to the generation of natural language texts. Three subjects are discussed:

- text specifications, i.e., a means for specifying the structure of a text that is to be generated,
- the structure of the data base, i.e., the way in which the data base is organized to facilitate the generation of natural language texts, and
- the text generation algorithm which generates a text given a text specification and a data base.

This approach represents the basic insight presented in this paper. It is simple and generatively very powerful, but a number of improvements are possible. Some of these possible improvements are discussed in Section 3.

2.1 Text Specification

A text specification is a compact description of the outline of a text. The form of a text specification is as follows:

<text specification>::=<subspecification>{;<subspecification>}$^{\infty}_{0}$

<subspecification>::=<object name>(<R>{, <R>}$^{\infty}_{0}$)

<R>::=<relation name>{(<R>{, <R>}$^{\infty}_{0}$)}$^{1}_{0}$

<object name>::=one or more contiguous characters

<relation name>::=one to three contiguous alphabetic characters.

The metalinguistic symbols have the following significance:

::=	'may be rewritten as'
<>	encloses a class name
$\{\ldots\}_a^b$	... occurs between a & b times

Object names are key words or phrases. They represent objects of interest within a data base, for example, names of commands in a programming language, people on a project or pieces of equipment in a system configuration.

A relation is a connection or association between one object and a fragment of text (i.e., a part of a sentence or one or more closely related sentences) and zero or more other objects. The following are typical relation names: NT (narrower term); PT (part); FUN (function); SYN (syntax); EG (example).

The significance of text specifications and of object names and relation names in text specifications is developed further in the examples that follow. The significance of objects and relations in the data base is described in section 2.2. The algorithm for generating a text, given a data base and a text specification, is described in section 2.3.

First Example

Consider the following request:

> Please create a text that explains the function and the syntax of the narrower terms of command. Examples of each command should be included. Each command should be discussed separately -- function, first, then syntax, and last an example.

This request can be stated briefly by the following text specification:

COMMAND (NT(FUN, SYN, EG))

The corresponding text that would be generated would have an outline in the following form:

First Command
    Function of First Command
    Syntax of First Command
    Example of First Command

Second Command
    Function of Second Command
    Syntax of Second Command
    Example of Second Command

etc.

The output for one command in the outline might be:

The function of the Set Command is to set a specified control parameter to a specified integer value. The format of the Set Command is:

<name> = <integer>

An example of the Set Command is:

SL:OU = 100

In the example, the maximum number of on-line users (SL:OU) is set to 100.

Second Example

Suppose that in addition to the text of the first example, an intoduction is desired in which a list of all the commands is given. The appropriate text specification would be:

COMMAND (NT); COMMAND (NT(FUN, SYN, EG))

Third Example

Suppose that instead of grouping information by command, it is desired that all the functions should be grouped together, etc. Then, the appropriate text specification would be:

COMMAND (NT(FUN), NT(SYN), NT(EG))

2.2 Data Base

A data base for a particular subject consists of two parts:

(a) a thesaurus that relates objects to each other and to fragments of text, and

(b) fragments of text.

2.2.1 Thesaurus

A thesaurus contains an entry for each object.

An entry for a single object consists of any number of relationships. Each relationship relates the object to a fragment of text and, in some cases, to one or more other objects in addition.

An object that is being focused on (i.e., as the object in a text specification or as the object that an entry is for) is referred to as a subject.

An object should be included under a relation in a particular entry if it occurs in the fragment for that relation and its meaning is not self-evident in that context.

The following is an example of an entry in a thesaurus for SET COMMAND:

SET COMMAND
FUN: 10
CONTROL PARAMETER
SYN: 11
NAME
'='
INTEGER
EG: 12

FUN, SYN and EG are relations. The function of the SET COMMAND is stated in fragment 10, the syntax in fragment 11 and an example of the SET COMMAND is given in fragment 12. The object CONTROL PARAMETER occurs in fragment 10 and its significance is not self-evident in that context.

2.2.2 Text Fragments

The data base includes a text fragment for each relationship in each entry. These fragments can be arranged (in the data base) into one or more unified texts, perhaps with some fragments left over.

Fragments 10, 11 and 12 referred to above might read as follows:

10: The function of the Set Command is to set a specified control parameter to a specified integer value.

11: The format of the Set Command is:

<name> = <integer>

12: An example of the Set Command is:

SL:OU = 100

In the example, the maximum number of on-line users (SL:OU) is set to 100.

2.3 Text Generation Algorithm

A flow chart of the text generation algorithm for handling subspecifications is given in Figure 2.3a. The algorithm as described does not include any error checking.

3.0 EXTENSIONS

This section suggests some of the ways that the basic approach could be modified to advantage. Like the previous section, it is divided into three subsections which discuss text specifications, the structure of the data base and the text generation algorithm. But instead of discussing one basic approach, it discusses many possible extensions of that basic approach. The extensions discussed often effect more than one of these three areas. They will be discussed wherever they can be presented in the most illuminating way.

Figure 2.3a

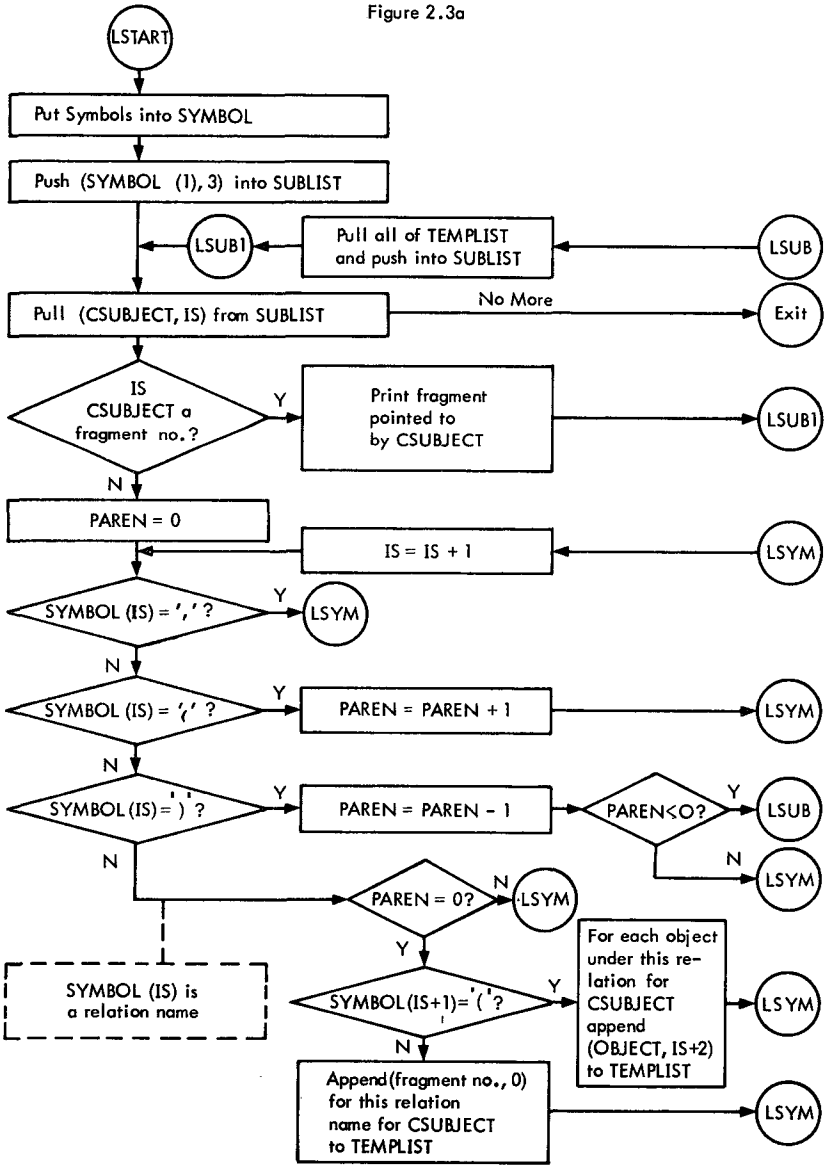

LSTART
Put Symbols into SYMBOL
Push (SYMBOL (1), 3) into SUBLIST
LSUB1
Pull all of TEMPLIST and push into SUBLIST
LSUB
Pull (CSUBJECT, IS) from SUBLIST
No More
Exit
IS CSUBJECT a fragment no.?
Y
Print fragment pointed to by CSUBJECT
LSUB1
N
PAREN = 0
IS = IS + 1
LSYM
SYMBOL (IS) = ',' ?
Y
LSYM
N
SYMBOL (IS) = '(' ?
Y
PAREN = PAREN + 1
LSYM
N
SYMBOL (IS) = ')' ?
Y
PAREN = PAREN - 1
PAREN<0?
Y
LSUB
N
LSYM
N
PAREN = 0?
N
LSYM
Y
SYMBOL (IS) is a relation name
SYMBOL (IS+1) = '(' ?
Y
For each object under this relation for CSUBJECT append (OBJECT, IS+2) to TEMPLIST
LSYM
N
Append (fragment no., 0) for this relation name for CSUBJECT to TEMPLIST
LSYM

FIGURE 2.3b

Notes for Flow Chart

SYMBOL	a list containing the symbols of the text specification in the order of their occurrence.
SUBLIST	a list of items each of which is either (a) a fragment number or (b) a pair that consists of a subject and an index to the first symbol in SYMBOL to be processed for that subject.
CSUBJECT	the current subject being processed.
IS	an index to the symbol in SYMBOL that is being processed.
PAREN	a counter for keeping track of parentheses.
TEMPLIST	temporary list for collecting items to be pushed into SUBLIST for one CSUBJECT.

3.1 Text Specification

3.1.1 Paragraph Boundaries

With texts of any size paragraphing is desirable. A new symbol (PARG) might be allowed which would indicate a paragraph boundary. Consider the following example of how it might be used:

```
COMMAND(NT(PARG, FUN, SYN, EG))
```

Paragraph boundaries would occur before the discussion of each of the narrower terms of COMMAND.

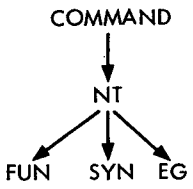
Section numbers and headings could be handled in a similar manner.

3.1.2 Higher Level Fragments in the Output

Consider the text specification:

```
COMMAND(NT(FUN, SYN, EG))
```

This specification can be viewed as a structure:



In the basic approach only the fragments of the lowest level relations are printed. The fragment that lists the NTs of COMMAND is not printed.

It would probably be more reasonable to interpret text specifications as implying that the fragments at all levels should be printed. If it was desired that fragments at a level be suppressed, they could be marked by a special symbol, say a minus sign.

3.1.3 Hierarchies of a Relation

There are many relations that naturally form hierarchies, e.g., narrower term of, part of, works for and subroutine of. Consider a data base with a hierarchy of NTs. The underlined letters represent subjects of entries.

3.1.4 Satisfying Dependencies

A fragment associated with a particular relation for a particular subject is dependent on explanations of the objects of that relation. An explanation of objects consists of the fragments for the essential relations of the object. But what is essential?

A priori it seems reasonable that functions (FUN) of objects would be essential and examples (EG). Not. But to insist on a particular classification of relations as essential and non-essential seems unnecessarily rigid. It might be better to have a default set of essential relations including (or limited to) FUN and also provide a capability to override this default set by a statement such as:

ESSENTIAL = FUN, NT

In addition or instead, relations that are considered essential for a particular subject could be marked as such in the data base.

Assuming that higher level fragments are included, that FUN is the only essential relation and that brackets indicate that dependencies are to be satisfied, consider the following text specifications:

```
COMMAND (NT(SYN))
```

What text specification would cause all four fragments to be printed? A(NT(NT(NT))) would, but suppose the depth of the hierarchy is not known.

A convenient way to handle this problem would be to use a special symbol (perhaps '!') after a relation name to indicate that indefinite nesting of the relation is desired.

3.1.4 Satisfying Dependencies

A fragment associated with a particular relation for a particular subject is dependent on explanations of the objects of that relation. An explanation of objects consists of the fragments for the essential relations of the object. But what is essential?

A priori it seems reasonable that functions (FUN) of objects would be essential and examples (EG) not. But to insist on a particular classification of relations as essential and non-essential seems unnecessarily rigid. It might be better to have a default set of essential relations including (or limited to) FUN and also provide a capability to override this default set by a statement such as:

ESSENTIAL = FUN, NT

In addition or instead, relations that are considered essential for a particular subject could be marked as such in the data base.

Assuming that higher level fragments are included, that FUN is the only essential relation and that brackets indicate that dependencies are to be satisfied, consider the following text specification:

[COMMAND (NT(SYN))]

The fragments giving the narrower terms of COMMAND and those giving the syntax of all the narrower terms of COMMAND would be included in the resulting text. In addition, the functions of COMMAND, of the narrower terms of COMMAND, and of any objects of the SYN relations would be included. Further, the functions of any objects of these FUN relations would be included, etc.

3.1.5 Natural Language Queries

The approach in dealing with natural language queries is to convert them into text specifications. In order to make the conversion, the following types of words would have to be isolated in the query:

relations (e.g., function, syntax, example),
objects (e.g., ADD COMMAND) and
connectors of objects with relations or relations with relations (e.g., of).

In the following example, the words that would need to be isolated are underlined:

Please create a text that explains the function and the syntax of all of the commands in the data base.

Next it would have to be determined which objects and relations were connected in the query and how. What we have is FUN and SYN of NT of COMMAND.

This must finally be translated into:

COMMAND (NT(FUN, SYN))

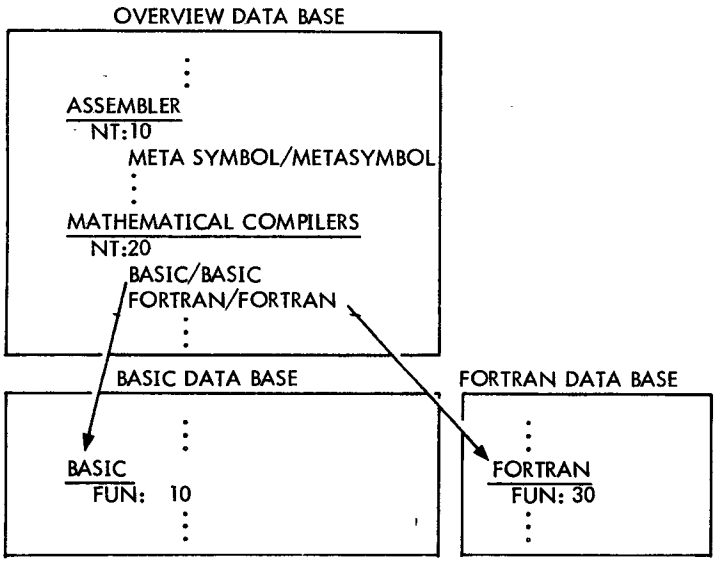
3.2 Data Base

3.2.1 References across Subject Areas

The approach described here depends on a text specification being processed for a particular data base. The data base should be highly controlled and relatively free from ambiguities.

Although each specification must be directed at a particular data base, not all (or even any) of the fragments in the resulting text would necessarily be from that data base.

Consider the following data bases:
A slash indicates that the name of the data base for that object follows:



The text specification MATHEMATICAL COMPILERS (NT(FUN)) would result in a text consisting of fragment 10 from the BASIC data base and fragment 30 of the FORTRAN data base.

3.2.2 Higher-Level Connectives

A higher-level connective is a connective that connects a sentence or a group of sentences with a sentence or a group of sentences. This is in contrast to the relations discussed so far which relate a subject to an object or to something else.

The following are examples of such connectives:

similarly (SIM),
in contrast,
thus,
otherwise and
for example (EG).

These connectives can be incorporated into the data base by expanding the reference to fragments. Consider the following reference to a fragment:

FUN: 10, EG:20, SIM: X(FUN)

This reference says that the function of the subject of the entry in which it occurs is stated in fragment 10. It says further that an example of the function is given in fragment 20 and that the subject X has a similar function.

3.3 Text Generation Algorithm

3.3.1 Generation of Fragments

The implementation of the text generation algorithm is simpler if fragment numbers and corresponding fragments are included in the data bases for all relationships; but for some relations (e.g., PT and NT) the fragment can be generated from the thesaurus entry itself. In other cases (e.g., FUN) part of the fragment can be generated.

For example, a relationship with the following format:

<subject>
PT:
<Object>
<Object>
⋮

implies a fragment of the form:

<subject> has the following parts: <object>, <object>...

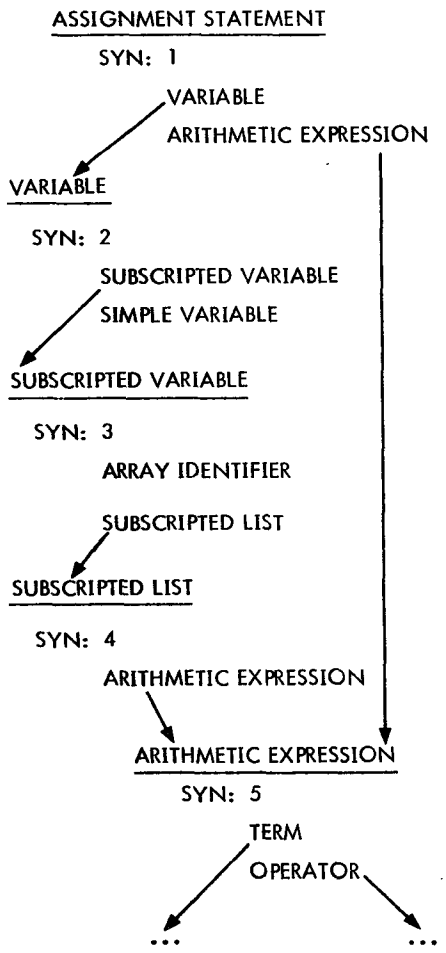
Lexical information for the subjects and objects would be necessary to include the correct articles and endings.

3.3.2 Eliminating Redundancies

In the context of a reference manual for a programming language, syntax would probably be considered an essential relation. The relations between syntactic objects is (roughly) hierarchical, but the overall structure of syntactic objects is generally not quite a

tree structure because of the fact that more than one object is often dependent on a single object.

Consider a data base that contains the following objects and relations:



The text specification:

ASSIGNMENT STATEMENT(SYN!)

would result in the syntax of ARITHMETIC EXPRESSION being described twice in exactly the same words. Probably this is not desirable, and it would be even less desirable if TERM and/or OPERATOR required further objects to explain them.

How can such redundancy be identified and how is it to be handled?

One way would be to construct a graph for the relationships to be included in a text plus references to all the occurrences of each object in the graph. For any object that occurred more than once a check would be made to determine if the subgraph going down from it in one occurrence was the same as in some other occurrence. All but one such subgraph would be deleted. The one that was to generate text earliest would be retained.

3.3.3 Flattening of Structures

Structures that go beyond a certain depth are often confusing if they are not broken up or flattened. Thus, in describing a programming language the basic symbols (such as arithmetic expressions) are often discussed before the discussion of individual commands. This means that in discussing an individual command (such as an assignment statement), it is not necessary to explain arithmetic expressions (or variables) in all their complexity.

Given the capacity for eliminating redundancies, it is possible to flatten structures. As an example, consider the following text specification.

ARITHMETIC EXPRESSION(SYN!); ASSIGNMENT STATEMENT(SYN!)

First, the fragment for the syntax of arithmetic expressions would be printed along with the fragments for the objects it depends on, etc. Second, the fragment for the syntax of the assignment statement would be printed along with the fragments for the objects it depends on, etc., except that the fragments for arithmetic expressions and the fragments for objects it depends on, etc. would not be printed because they are redundant.

4.0 APPLICATIONS

The approaches to text generation described in the previous two sections have applications in many areas including the following:

HELP subsystems for time-sharing systems,
Automatic Documentation Systems,
Vocabulary Control,
Automatic Letter Writing and
Automatic Report Generation.

The first two of these applications are discussed below.

4.1 HELP

A HELP subsystem is a part of a time-sharing system that helps the user to understand the system and the various parts of the system. Two main types of help may be provided:

--ability to answer questions about the system (without reference to the current state of the system) and

--ability to answer questions about the current

state of the user's job.

The techniques described in this paper are oriented more towards the former capability.

The user of a HELP subsystem is typically in the middle of a task when he needs help. He wants what he has done so far to be intact when he returns to his original task. Moreover, he does not want to have to make a special effort to achieve this because (a) he might forget and (b) it takes time and he is in a hurry. Often these goals can be achieved by incorporating the HELP capability into the terminal executive of the system.

It is especially important that a HELP subsystem help the user to understand how it should be used. For example, if the user types in an illegal text specification, the HELP subsystem might offer to display some material concerning the proper format of a text specification.

Ability to handle simple natural language queries is very desirable, at least the first time a user uses a HELP capability. It means that he can use it without having already learned to use text specifications.

Example

A user of a time-sharing system is entering the statements of a BASIC program on-line when a syntax error occurs on an assignment statement. But the error message does not make clear to him how he can correct the error. So, he exits to the terminal executive. Then he asks a series of questions about the syntax of an assignment statement in BASIC, finds the information that makes clear how to correct the error, and returns control to the on-line BASIC compiler with the same environment

(including the partly entered program) as when he left. Then, he can reenter the statement that was in error and continue as if nothing had happened.

4.2 On-Line Documentation Systems

An automated documentation system consists of capabilities for maintaining a data base and for producing formal documents such as reports or documents for individual use. The approach discussed in this paper has primarily to do with the generative capabilities. The documents generated would be natural language texts.

Some of the advantages of an on-line system are: (a) the system can be accessed when the information is needed; (b) the information that is received may suggest further queries and (c) syntactic or semantic errors in the text specification can be corrected at once.

A system using the approach described has the advantage of suggesting gaps in the documentation. For example, one might discover that a particular object never occurs as a subject or that a relevant relation is missing from an entry.

As a fall-out of the approach, one has a thesaurus (or glossary) for vocabulary control.

A couple of examples of the use of such a system may be helpful in communicating its significance.

First Example

A new person has been assigned to an implementation project. He would like up-to-date documentation of parts of the system relevant to

the work he will be doing. In different areas he wants different types of information. The structure of the texts generated for him can be tailored to his needs by use of appropriate text specifications. If he needs more information in some areas, he can use the system interactively.

Second Example

The information in a particular area changes frequently and a number of people need to receive up to date information periodically. A text specification can be created to generate the appropriate information, and (assuming the structure of the data base doesn't change significantly) the same specification can be used to generate a text with the same structure (but different information) as often as is desired.

BIBLIOGRAPHY

1. Lauriault (Loriot), James

   Shipibo Paragraph Structure
   unpublished paper, Summer Institute of Linguistics, August, 1957.

2. Jacobson, S.N.

   "A Modifiable Routine for Connecting Related Sentences of English Text."
   in Computation in Linguistics (edited by Paul L. Garvin and Bernard Spolsky), Indiana University Press, Bloomington, Indiana, 1966.

3. Woolley, George H.

   Syntax Analysis beyond the Sentence
   (presented at the Fourth Annual Meeting of the Association for Machine Translation and Computational Linguistics), Computer Associates Inc. document no. CA-6607-2121, July, 1966.