REPRESENTING TEXT STRUCTURE FOR
AUTOMATIC PROCESSING

by

Lynne Ann Price

Computer Sciences Technical Report #324

May 1978
COMPUTER SCIENCES DEPARTMENT
University of Wisconsin - Madison

REPRESENTING TEXT STRUCTURE FOR AUTOMATIC PROCESSING

by

Lynne Ann Price

Computer Sciences Technical Report #324
May 1978
Representing Text Structure for Automatic Processing

BY

LYNNE ANN PRICE

A thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN - MADISON

1978
Information can easily be organized, cross-referenced, sections, and other related on index and the body of a manual. An anchor passage is used to locate material without altering a separate search to locate material. This program allows users to locate material without altering the material, a system for Interaction between manual searching and computer searching. In particular, a system for Interaction between manual and computer searching is suggested for automatic searching. Although no algorithm is suggested, for automatic searching, data is organized in the form of a text, the text is organized to indicate the interaction among such concepts and to associate the concepts appearing throughout a text. The steps allow the user to organize, search, and retrieve information in a programming environment. In a programming environment, the second type of retrieval is intense, where the second type of retrieval is intense, paragraphs of a text and their functions are associated by the hierarchy formed by the chapters, sections, and the text groups, which particular other sections, descriptions of the classes of structural relations are emphasized.
every evening at school. will be requested next year when it's your turn to spend
Finally, a direct note to my husband: the kids and I
drew the figures.

composed, and crepe sorel is a contributor. its homogeneous
the text, sally handy-karnosky, paul lagergren, nancy
many hours discussing my ideas and suggesting corrections to
several of my friends, especially kathleen kernel, spent
schedule.

term for this time and willingness to rearrange this
rushed frantic for his help with the writing, and another
committee members, who did their frequent encouragement
after the first session. i am also grateful to the other
very helpful, i appreciate both karen's continued support
research, both my advisors, dick vonchak and larry treas.
I would like to thank everyone who assisted this

acknowledgments
8.7 Canonical Forms and Measures of Complexity

8.8 Automatic Generation of Text

Appendix A

Appendix B

Appendix C

Appendix D
Introduction

1. General Description

2. Reference.
There are several approaches to understanding the complex relationships between different versions of a text, such as those involving structural, semantic, and stylistic features. These approaches often require a comprehensive understanding of the text's historical and cultural context.

Some approaches to understanding texts involve the use of computational tools, such as natural language processing (NLP) techniques. These tools can help identify patterns and relationships within texts that are not immediately apparent to human readers. For example, NLP tools can be used to identify key words and phrases, as well as to extract thematic and structural information from the text.

One approach to understanding texts is to use a combination of computational and qualitative methods. This involves analyzing the text's structure and content using a variety of tools and techniques, including machine learning algorithms, statistical models, and linguistic rules. By combining these approaches, researchers can gain a more comprehensive understanding of the text's meaning and significance.

Another approach is to use text mining techniques to extract meaningful patterns and relationships from large collections of texts. This can help identify trends and patterns that are not apparent when looking at individual texts separately. For example, text mining can be used to identify common themes and topics across a corpus of texts, or to identify key concepts and ideas that are repeated across different sources.

Overall, understanding texts requires a multidisciplinary approach that draws on a variety of tools and techniques. By combining qualitative and quantitative methods, researchers can gain a more comprehensive understanding of the text's meaning and significance.
In this paper, Chapter 2 surveys structural features of text. In this section, the organization of the remainder of this text is described. Chapter 3, previous representations of text, sets the stage. Chapter 4 proposes a new notation.

1. The structure of this text

The techniques discussed are applicable to interpretation. Although most attention is given to applications within computer science, the techniques described in detail, although most applicable in part, are also applicable to other areas of science and engineering. For a complete manual and tutorial documentation of a computer-assisted writing system, see McNeil's book, "The structure of a scientific notation," and "The structure of a scientific notation with application to literature."
are ideas of language samples and not coherent texts
distortments are composed of natural language elements, they
distortments are composed of natural language elements, they
language units. While literature has developed and
language units. While literature has developed and are never resulting in no coherence to form jargon
are never resulting in no coherence to form jargon
they execute sequences of natural language expressions that
they execute sequences of natural language expressions that
continuous texts consist of coherent grammatical sequences!
continuous texts consist of coherent grammatical sequences!
isn’t, and essays, our execute isolated paragraphs.
isn’t, and essays, our execute isolated paragraphs.
including short stories, novels, newspapers, or magazines.
including short stories, novels, newspapers, or magazines.
that is complete, continuous, and linear. Complete texts
that is complete, continuous, and linear. Complete texts
This thesis is primarily concerned with written prose
This thesis is primarily concerned with written prose

I. Texts to be Considered

Describing the structure is introduced.

Exposure of natural language texts become a model for
Exposure of natural language texts become a model for
their articulation. It is important to study the articulation of
their articulation. It is important to study the articulation of
different aspects of text structure than do others. Because
different aspects of text structure than do others. Because
some work mentions in the context of literary criticism. Some
some work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some
work mentions in the context of literary criticism. Some

not provide a complete description of the prototype of a

10
2 Some structural relationships expressed there, and identification of cohesive elements, determine sections, and the effect of presentation of the content. The structural relationships are not only meaningful in themselves, but also help in understanding the overall content. The remainder of this chapter emphasizes properties that are relevant to encode the structural relationships that are prominent in the text.

Consequently, the notation proposed in Chapter 4 allows an accurate (3, 4) dealing with the order in which facts are presented. (3, 4) Presentation to a theme of fact by very abstract (3, 5) presenting to a theme of fact in different contexts. Structural relationships may be common to non-determinate passages (3, 6). A location where text (4, 5) chapters, sections, and the recurrent theme, success in chapters, emphasize the structural relationships of a document. Further utilize sections containing several paragraphs. Further, the following pages concentrate on presentation among structural relationships. Identification of structural relationships and sentence structure. Presentation to the understanding of a text, non-redundant by means of structural features, condensed only in the sentence. Therefore, to have only a marginal effect on the text, it is necessary to refer to the next paragraph. The remainder of this chapter emphasizes properties that are relevant to encode the structural relationships that are prominent in the text.
actual number of problems within the block.

foo following by a book of worked examples, and to improve the actual number of problems within the block.

But this is only a partial view of the field. The number of problems or example problems may vary among different sources. Mathematical textbooks, for example, careful presentation or a set of problems may be expressed at several levels of detail. Structure can be expressed at several levels of detail; the amount of abstraction may not be pertinent for some studies.

The second one is a conflicting follow-up study.

to the second one as a conflicting follow-up study.

work may thoroughly disagree the earlier experiment and further cooperate disassembling the second. Another report of the same type was published in a preliminary to be held.

A second form of emphasis is the establishment of an idea.

The portions are narrated by different characters (e.g., the portion of information from the second paragraph). The portions are treated as if they were separate but can be related in various ways.
The number of stories connects largely to the size of the audience. All three novels featured in the discussion, each by a different author, have parallel structures: they are about a boy and a girl [15] who live in the same neighborhood. The Forest, written by a single author, explores the theme of the forest, the last great wilderness. The Fire, written by two authors, deals with different aspects of fire, contrasted with different aspects of water. In this way, the structure of the novel is reflected in the structure of its parts.
In discussions of semantic networks, the range of possible meanings for words, terms, and concepts often expands beyond the literal sense. These expansions include metaphorical, connotative, and connotative meanings. The structure of a concept's semantic space may include various associations, some of which are more direct than others. The discussion of the gorilla and the baby gorilla, for example, highlights the importance of understanding the context in which these terms are used.

Paraphrasing...

The relationship between any number of concepts is complex and can be difficult to describe succinctly. Expressions such as "the concept of a concept" illustrate this complexity. The various sections of this text discuss the interconnections and differences between the gorilla and the baby gorilla, and the broader context in which these terms are used.
3.2 Introduction

3.3 Representing Text Structure

Research, features common to several previous studies are performed by discussion of some of the goals of this paratextuality, need to describe textual organization and represent text structure. Examples of notation and unprocessed text passages. This chapter focuses on methods for learning compressed, and the relationships suggested by representational relationships that are retained for a text vary in the amount of detail contained, the type of special application. The different representations for one representation may seem those that are retained for a chapter 2, the annotation who encodes a particular text of the various structural features that are used for a single unprocessed, the approach developed in chapter 4 allows that should be encoded to encode representations in which he is the same time no researchers. A new representation of text structure should preserve

bacterial

special case of the more general goal of understanding human action, the problem of understanding the relations that discourse shares properties with other the interaction of "nonverbal behavior" to this fact

without end.

nonverbal behavior, movie scripts, editorials, conversations, research reports, novels, law's, conversations, we're trying to balance issues, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversies, controversi...
function alone. In the traditional point diagramming
comprehensive, another factor approaches, when one
exceeds and satisfies the criteria of function, cross-reference,
representation, out they are simplifications that do not
outlines and tables of contents are common parts.
There is no obvious method for ending text structure.

2.2. The Hierarchic Approach

two iterations can be used to measure the accuracy.

The notation introduced below is intended to encode the
units used.

General-purpose notations are perfectly determined by the
individual factors. The possible applications of a
different notation are well as to refer to pieces of
different factors, that are used to combine the same from
187 factors and units are used to combine them from
choice of units for the structural study of tocolysis, under
the text. Motifs ogeographic the plot of a story. According to
functions, especially the aprotase activity by a position of a
molecule (6.6. annual) 6).

previous research the relationship (6.6. introduction)
progress in more detailed below, structural units used in
description is dependent on units of some sort. As

function: the function needed by each
ordering: the order in which ideas are presented
language: 2.2. to refer to.

representation: the proposed method. It should be
applicable to different classes of tasks.

generality: the notation should not depend on

practical applications: should be economically usable in several

utilization: the resulting description

following criteria can be used to measure the accuracy.

just a few examples from Chapter 2, extracted from the narrative of a tale. An example is

"The Deep of the Parents' Exposition," an account of the children of the Gaerell family, narrated from the perspective of a young girl who takes the form of a character's mother.

In the Deep, the parents, characters, and their actions are described in a way that highlights the structure of the narrative. The narrative's deeper meanings are explored through a combination of exposition and interaction, creating a rich tapestry of events and emotions.

The Deep of the Parents' Exposition is a narrative that explores the deeper meanings and interactions of the characters in the story. It is a rich and complex exploration of the relationships and events that drive the narrative forward.

The Deep of the Parents' Exposition is a narrative that explores the deeper meanings and interactions of the characters in the story. It is a rich and complex exploration of the relationships and events that drive the narrative forward.
Martine was holding tightly to the string.

(1) The martine hit a branch and (2) burst.
(3) Suddenly a gust of wind caught it.
(4) The branch fell into a tree.
(5) The branch fell into a tree.

The structure of a story is represented by a parse tree, the nodes of which are connected by arcs of the parse tree.

---

**Example:**

```
(episode -> event + reaction)
```

These syntactic rules are accompanied by semantic rules, for

```
episode -> event + reaction
```

and

```
story -> setting + episode
```

defining the syntax of stories with rules such as:

```
Although Popp's notation satisfies the criteria of
```

...
different categorizations of textual elements. The breaking
detail included in the tree. Other categorizations reflect
these alterations reflected a difference in the level of
branching, or three distinct extents (as mentioned here chosen).
Level (2) may be said to consist of a single event, a par
more accurately make of a text. The content of nodes (3) through
unquestionably make this the group permuted a

27
3.3 Non-Hierarchical Methods
By a representation of a representation, in the sense, we may be given description to represent the process of perception to the reader. The process of perceiving text, when the details are interrelated, can be controlled to the next of not clearly dealing with other ideas, text can be omitted. Transitory passages are not needed. There are other applications where some portions of a text can be omitted.

(1) The main point of the text.
(2) The main point of the story.
(3) The family sergeant.
(4) The purpose or reward.

Examples:

Stories to three structural patterns. One of these patterns is the main point of a text. Transitory passages are not needed. Transitory passages are not needed. Transitory passages are not needed. Transitory passages are not needed. Transitory passages are not needed. Transitory passages are not needed. Transitory passages are not needed.

A general notion of a general notion should be able to understand portions of a text. This notion should understand portions of a text.

What is the importance of a general notion? A general notion can be understood through portions of a text. This notion should be able to understand portions of a text.

Chapter 2 argues that order of presentation is unimportant. Chapter 3 suggests that order of presentation is unimportant. Chapter 4 suggests that order of presentation is unimportant. Chapter 5 suggests that order of presentation is unimportant.
Chapter 9: A Hierarchical Composite

3.5 A Hierarchical Composite

Parentheses should be introduced and separately delimited. Transitory parenthesis between ideas. For this purpose, transitory parentheses may be considered as part of the main paragraph. Of course, a detailed treatment requires a new section for the demonstration. Of course, a detailed treatment requires a new section for the demonstration. Of course, a detailed treatment requires a new section for the demonstration. Of course, a detailed treatment requires a new section for the demonstration.
To prepare a text for the on-line documentation system, the exact appearance of a particular passage tree is essential. By the appearance of a passage tree, we mean the meaning of a continuous section of text. When this meaning is used, the word "passage" has two meanings: Firstly, the root of the passage tree corresponds to the entire text. The text segments represented by their descendants, the portions of the document where the text can be incorporated into the documentation of their descendants, are called passages. Leaves of the passage tree are of a text is called the passage tree; each node in the tree represents the natural subdivisions of each component of a step is elaborated below.

4. Strips

If you don't' find it in the index, look very carefully through the entire book...
adjacent, non-overlapping text segments. Remember of this paper, the paragraphs are formed over several general functions served by passages in many texts. Important functions (e.g., monitoring, reward, „docking”, „response”) are functions targeted rather than named. Many of these functions are used in these schemes (e.g., „monitoring function”, „reward function”). The instrument has been adapted from [7], and the data from [11]. The data from [11] can be used to describe text-to-sentences such as those of the passag e appropriat ely identified. Paragraphs with the same appropriately identified, paragraph can be used to show the function of „reward function”. In this paper, the function of „reward function” and serves the function of „monitor function”. Important functions are named here. In chapter XII and XIII, the function names are the names of chapter XI. In chapter XI, the function names are the names of chapter X. An example of a paragraph with both section and option headings. An example of a paragraph with an option and section heading. It is necessary that every passage in a strip be connected. In addition, one or more functions served by the passage may be assigned to the corresponding node in the passage tree. To identify portions of a text, one or more names may prevail. Examples, that might be displayed together, are some hierarchical representations of text structure are a good example.
The relevant chapter has been divided into passages whose hypertextual manual for an alphabetic programming language is an example of the idea. The concept graph is illustrated below. A possible concept graph is as follows:

4.3.1 Concept Graphs

4.3 The Passage-Dependent Network
Figure 4.2: A sample concept graph.

Figure 4.3: Shows a concept graph for constant-based expressions.

Figure 4.4: Leaves of a hypothetical parse tree.

42
Figure 4.4 A less-detailed segmentation

Figure 4.5 An example of the use of concepts in arithmetic expressions

The form of the concept graph in Figure 4.5 can be retained...

Figure 4.3 A small variation

And the graph shown in Figures 4.4 and 4.5 preserve this information...

More substantively, the second example might be longer...

Another method of distinguishing examples

Simple

Complex
The diagram illustrates a concept graph with several references to a single passage. This figure is almost identical to Figure 4.2. It contains a node labeled "expression" and a node labeled "common." The graph shows the relationships between these nodes and other nodes in the graph. The passage number is 4.4, and the page number is 46.
Co-occurrence nodes are also part of the dependency graph from the root node. However, if a node is removed from the dependency graph, the co-occurrence node is retained. A co-occurrence node cannot be removed from the dependency graph by pruning a copy of the dependency graph from the root node. If a co-occurrence node is removed from the dependency graph, the co-occurrence node is retained. A co-occurrence node cannot be removed from the dependency graph by pruning a copy of the dependency graph from the root node. If a node is removed from the dependency graph, the co-occurrence node is retained. A co-occurrence node cannot be removed from the dependency graph by pruning a copy of the dependency graph from the root node.
For the reader, redundant identifiers in the hope that they clarify examples.

Redundant identifiers in the hope that they clarify examples. The remainder of this paper uses the parent node. The remainder of this paper uses the parent node. The remainder of this paper uses the parent node.

Figure 4.12: Redundant identifiers in a concept graph

Figure 4.11: Example of a co-occurrence node

Figure 4.7: Recognition leads to transformation yields. Figure 4.11 from section 4.3.3. Names and functions in concept graphs.

Note: The nodes marked respectively "range" and "example" only other relevant ancestors of the references to C and to co-occurrence nodes. Thus, starting from "integer", the descendants of a selected node are ignored unless ancestors of co-occurrence nodes are included, as shown in Figure 4.9. As the program executes the this new graph can be used to provide the sample.
formed by the category node representing the concept. All
the graph in the shape of a concept network by connecting the supergraph
and the component graph can be connected with
emanating from the corresponding nodes in this supergraph.
The entire network would probably contain other areas.

**Figure 4.13** Portion of a passageway-like network

![Diagram](image)

Examples

```
\text{e.g.,}
```

```
\text{\texttt{\#1,\#2,\ldots,\#n}}
```

**Definition**

```
\text{\texttt{\#1,\#2,\ldots,\#n}}
```

**Range**

```
\text{\texttt{\#1,\#2,\ldots,\#n}}
```

```
\text{\texttt{\#1,\#2,\ldots,\#n}}
```

```
\text{\texttt{\#1,\#2,\ldots,\#n}}
```

```
\text{\texttt{\#1,\#2,\ldots,\#n}}
```

The subgraph networks may appear as follows:

- **Mathematical expressions** are used to express to

  - **Contrast**

  - **Real**

  - **Integer**

  - **Constant**

- **Type**

The network diagrams are used to express the concept of a

- **Concept**

- **Example**

- **Range**

- **Definition**

In the case of the concept networks, the entire network would probably contain other areas.

**Graph for multiplication**

The graph for multiplication and the graph for division are similar. On the other hand, the two types of concepts can be expressed as two graphs of the graph for multiplication and division, respectively. The graph for multiplication and division are used to express the concept of a

- **Contrast**

- **Real**

- **Integer**

- **Constant**

- **Type**

The network diagrams are used to express the concept of a

- **Concept**

- **Example**

- **Range**

- **Definition**

In the case of the concept networks, the entire network would probably contain other areas.

**III** Concepts pertaining to a given application could be

4.3.4 Combining The Graphs Of Several Concepts
Figure 4.1.5 Concept Graph for "Real Constant"

expression

expression:  The concept graph for "real constant" is the graph for Figure 4.1.5, except that it does not include the reference to "real constant" as a node, and the node "example" is not connected to the node "expression." The concept graph for "expressions" is a supergraph of the concept graph for "real constant." The concept graph for "expression" includes additional nodes and edges not present in the concept graph for "real constant."
This concept graph includes the co-occurrence node, but does not contain the "integer constant" ancestor of that node.

There is another advantage to encoding all cross-references in a single network. Concepts such as "constant" and "real" are different from concepts such as "definition" and "example". An analyst concerned with locating information views the latter ideas as subcategories of concepts of the former type. However, a researcher interested in the use of definitions within a class of documents treats the first type of concept as instances of the second. The same passage-dependent network can be used by both individuals. Nodes representing ideas in each class are modifiers in the concept graphs for concepts in the other class.

4.4 The Passage-Independent Network

Although semantic information is represented in the passage-dependent network, this graph is not a "semantic network" in the usual sense of that expression. The traditional semantic net represents meaning; the passage-dependent network uses meaning, supplied by the analyst, to group passages of a text. Thus, Figure 4.13 shows that "integer constants" can be discussed in relation to "definition", "range", or "examples". It does not define "integer constant" as an entity with the properties "definition", "range", and "examples". The third component of a sterep is a semantic network used to represent any structurally significant portion of a text's content. Unlike the passage-dependent network, this network is a semantic net in the usual sense of the phrase.

This network is called the passage-independent network because the information encoded within it is not explicitly bound to specific passages of the document. In fiction, family relationships among the characters typically have this property. Propp [56], for one, considers some family relationships important to the structure of folktales. This information could be encoded only awkwardly in the passage-dependent network. In a sterep for the familiar fairy tale "Hansel and Gretel", a reference to the sentence "Hansel took his sister's hand" could be used to indicate the children are brother and sister. However, the reference could not have the same significance if the sentence were replaced by "Hansel took Gretel's hand". In applications that require this information, the passage-independent
network allows it to be specified in a manner that is independent of simple variations in the text.

Other relationships among recurring elements can be encoded in the passage-independent network. For programming manuals, the passage-independent network can indicate parameters used by different routines, options used on various programs, different language constructs that can employ a given sub-construct (e.g., statements that can contain arithmetic expressions), or the range of values a data structure may acquire. Streps for various types of instructions (e.g., recipes, assembly directions for a bicycle, knitting patterns) can indicate tools used in different tasks. For history books, a president might be associated with the time period he was in office. Geographic data (e.g., distances between cities, identity of state capitals) might also be encoded.

The information in the passage-independent network need not all pertain to concepts identified in the passage-dependent network. Relations among passages in the passage tree can be indicated; possibilities include implication, cause and effect, and simultaneous occurrence. To represent the flow of time in an event-oriented text, all three components of the strep are needed. For each point in time mentioned in the text, a concept is entered into the passage-dependent network. This concept refers to the passages describing events that occur during the time period. The passage-independent network encodes the ordering of the different time periods.

Some applications for streps do not require a passage-independent network. Chapter 7 describes a system designed to utilize the information in the passage tree and passage-dependent network only. In certain instances, complex relationships can be indicated by functional labels in the passage-dependent network. In a strep for a fairy tale, there must be a concept filling the function of "hero" and another filling that of "villain". The conflict between the corresponding characters need not be encoded in the passage-independent network; it is implied by these functions. In a study of character roles in folktales, an analyst would presumably begin by creating a complete strep for each tale in some corpus. He would indicate the relationships among the characters in the passage-independent networks. Once he had examined enough data to form generalized role identifications, he could use labels for the identified functions to test his hypotheses on additional stories without encoding their passage-independent networks.
When it exists, the passage-independent network is part of the structural framework established by the passage tree and passage-dependent network. Some of its nodes correspond to concepts in the passage-independent network or passages in the passage tree. In fact, in the examples given above, the passage-independent network shows additional relations between concepts of the passage-dependent network. Because of this interrelationship, the entire strep may be viewed as a partitioned network [65] whose three main planes are the passage tree, passage-dependent network, and passage-independent network. Individual concept graphs may be viewed as subplanes of the passage-dependent network.

4.5 Adequacy of Streps

Chapter 3 identifies criteria for an adequate representation of text structure. This chapter has introduced such a notation and shown how the criteria of order, function, cross-reference, and flexibility are met by the strep. The ability of the strep to satisfy other criteria is demonstrated in succeeding chapters. It is appropriate to review the criteria in turn and describe how each is satisfied by the strep.

practicality: This property can be demonstrated by implementation of an economical system using streps. Although no such implementation has yet been undertaken, Chapter 7 thoroughly describes one such program and Chapter 8 mentions several other possibilities.

generality: Of the four complete examples given in the next two chapters, none uses a text written in expectation of analysis by strep. The examples represent four very different classes of text: a Russian fairy tale, a science book for young readers, internal documentation for a set of computer programs, and a user's manual. Although three of the texts were written in English, the fourth is a translation from the original Russian.

order: As illustrated in this chapter, the passage tree encodes the order in which ideas are presented in the text.

function: Functional labels attached to the nodes in the passage tree identify the function served by portions of the text.

cross-reference: The passage-dependent network associates appropriate portions of the document with recurring entities.

flexibility: Many different streps are possible for one text. The manner of connecting the passage tree to the text, the number of divisions shown in the passage tree, the specific concepts shown in the passage-dependent network, the names and functions identified, and the relationships encoded in the passage-independent network may all vary. This flexibility allows the analyst to encode only relevant structural features.
completeness: The applicability of streps to entire texts is demonstrated by the examples in Chapters 5 and 6. The partial examples in the current chapter are intended to show that each of the structural features described in Chapter 2 can be encoded.

incrementality: For successful use with programming manuals, it must be possible to update a strep with minimal effort each time a text is revised. Chapter 7 describes a method of combining a strep-generator with a word-processing system in order to simplify concurrent updating of both strep and text.

The next chapter demonstrates the generality and completeness of the strep notation by applying it to two complete texts.

5. Two Detailed Examples

5.1 Introduction

This chapter describes possible streps for two different texts: a Russian folk tale and a science book for young readers. The passage trees and passage-dependent networks are discussed in detail. Because the passage-independent network is highly dependent on an individual application, a possible passage-independent network is described here only for the first text. The bodies of the selected texts and the completed passage trees and networks are too bulky to be included in entirety here; this material is presented in Appendices A and B.
5.2 Frolka Stay-at-Home

5.2.1 The Story

The first example shows that Propp's analysis [56] of a Russian fairy tale can be embedded in the strep notation and that this notation easily permits elaboration of the analysis. On page 128, Propp presents a summary of a sample story with a kidnapping theme. The tale he selects is numbered 131 in the fifth and sixth editions of Aleksandr Afanasev's Narodnye Russki Skazki, the definitive collection of Russian folktales. In English, the story is called "Frolka Stay-at-Home". 1 A translation of the story is presented, segmented into passages, in Appendix A. Propp's summary, with explanations of his symbolism is shown below:

1. Norbert Guterman has translated many of the Afanasev tales into English. Although the original enumeration of the stories was not preserved in the translation, it seems clear from the summary that No. 131 is the story Guterman titles "Frolka Stay-at-Home". Prof. James O. Bailey of the Department of Slavic Languages at the University of Wisconsin at Madison has kindly taken the time to confirm this identification.

(1) A tsar, three daughters (α; Setting).

(2) The daughters go walking (φ; Absentation of Younger Generation).

(3) overstay in the garden (α₁; Violation of Implied Interdiction).

(4) A dragon kidnaps them (α₂; Villainy in the Form of Abduction).

(5) A call for aid (B₁; Mediation in the Form of a Call for Help).

(6) Quest of three heroes (C; Counteraction and T; Departure of heroes).

(7) Three battles with the dragon (H₁; Struggle in the Form of Battle in Open Field—I; Defeat in the Form of Villain Beaten in Open Combat).

(8) rescue of the maidens (K₄; Object of Quest Obtained as Direct Result of Preceding Action).

(9) Return (O; Return of Hero).

(10) reward (w₀; Monetary Reward).

Figure 5.1 Propp's Summary of "Frolka Stay-at-Home"

Concatentation of the symbols identified with each element of the summary (except the setting, which does not correspond to action in the story), yields Propp's description of the structure of the text:

φ³ α₁ α₂ B₁ C T H₁ I K₄ O w₀

Figure 5.2 Propp's Notation
5.2.2 The Passage Tree

This section is not intended as a criticism of Propp's work. Propp achieved his goal by exhaustively categorizing the events in Russian folklore and defining order restrictions on the resulting functions. It is only for other applications, such as generation of full-length natural-sounding stories, that a more complete analysis is required.

The passage tree corresponds to Propp's linear string. Propp's analysis and terminology can be preserved in the string notation by using his symbols, or their descriptive titles, as functional labels on the nodes of the passage tree. With a direct translation, the tree would have eleven nodes: the ten leaves would be direct descendants of the root and each leaf would correspond to an item of the summary shown in Figure 5.1.

Although a tree formed in this manner would accurately duplicate Propp's description, its passages could not be associated with text segments in the manner chosen in Chapter 4. Propp's string is effective as a summary of the story, and it does indicate the order in which events occur. However, the text cannot be rigorously segmented so that consecutive passages correspond to consecutive symbols in his string. The problems are minor and result from Propp's use of a linear notation to represent a tree structure. Very simple extensions to the original analysis permit a segmentation of the story to be associated with a passage tree in a manner that preserves the functions identified by Propp.

The most serious discrepancy between the order of events in the story and in the summary involves the order of the battles with the dragons and the rescue of the princesses. The summary indicates that all the fighting occurs before the rescue is accomplished. In the folktale, however, each battle results in the release of one of the maidens.

The three-fold repetition of story elements is very common in folklore. Propp mentions trebling but does not dwell on a topic that "has already been sufficiently elucidated in scholarly literature" (p.74). According to a strict interpretation of Propp's rules, "liquidation of a misfortune" (rescue of a princess) cannot be followed by a struggle between the hero and a villain. Recognizing this story is well-formed even though a battle follows a rescue, Propp summarizes these events with the phrase "three battles with the dragon" followed by "rescue of the maidens" and
does not mention that these events are actually intercalated.

When structure is represented by a tree instead of a linear string, the order restrictions can be stated with respect to subtrees. The rule mentioned above can be rephrased as "a node denoting liquidation of misfortune cannot be followed by a sibling that denotes a struggle between a hero and a villain". This rule allows structures such as the following:

![Figure 5.3 Accurate Representation of the Three Battles]

This figure represents the events in the story more accurately than does Figure 5.2.

Other discrepancies between the order of events in the story and in the summary are less significant. For example, Propp indicates that the "absentation of younger people" precedes the "violation of implied interdiction". Both functions are indicated in the translated tale by the single sentence, "One night the king's daughters tarried in the garden..."; neither function precedes the other. When the text is segmented to form the passage tree, this situation can be indicated by assigning both functional labels to the segment containing this sentence.

Propp himself uses multiple identifiers on a single passage. In the summary, for instance, the quest of three heroes is marked both "Consent to Counteraction" and "Departure of Heroes", and the battles are labelled both "Struggle in the Form of Battle in Open Field" and "Villain Beaten in Open Combat." Nevertheless, the tree shown in Appendix A does not use the multiple labels that Propp does. The "Consent to Counteraction" and "Departure of the Heroes" are indicated by separate clauses in the text:

Three men agreed to undertake this task...

and

and they set out to look for the princesses.

In the passage tree, each clause forms a distinct passage and is individually labelled.

In the other case, the advantages of a tree structure are used to distinguish between a battle and a villain's defeat. It would be inaccurate to claim that the defeat follows the battle; the defeat is part of the battle. Propp
has solved this problem by combining the two functions. In the tree, however, the passage corresponding to each battle has two descendants: one for the beginning of the battle, one for its conclusion.

The tree structure can also indicate other function-subfunction relationships. Propp obviously felt that the hero's counteraction to the act of villainy was important; therefore, he identifies the "Consent to Counteraction". In a linear notation, he cannot indicate the functions that combine to form the counteraction. However, in the passage tree, the events from the "Consent to Counteraction" through the final rescue are grouped together as descendants of a node labelled "Counteraction".

Although Propp's work is a classic in the field of folklore, it is by no means accepted as definitive. Bremond [9] points out that Propp's functions do not account for many types of similarities among fairy tales. Propp's analysis, for instance, does not classify "Frolka Stay-at-Home" as a story about dragons. In addition, a purely functional analysis overlooks similar episodes that serve different purposes (e.g., depriving someone of food may be an act of villainy in one story and the villain's punishment in another.)

The Aarne-Thompson Tale-Type Index [1] is another well-known classification of fairy tales. This index is divided into Animal Tales, Ordinary Folktales, and Jokes and Anecdotes. These categories are subdivided; the Ordinary Folktales, for example, include Tales of Magic, Religious Tales, Romantic Tales, and Tales of the Stupid Ogre. Dundes [18] has two major objections to the Aarne-Thompson approach: 1) this scheme does not uniquely classify stories that involve, say, both an ogre and a magic object and 2) this scheme does not show similarities between essentially identical stories in which, for example, a fox tricks a bear or a man tricks an ogre.

The strengths and weaknesses of Propp's method are complementary to those of Aarne and Thompson. The strep notation allows both views to be represented. "Frolka Stay-at-Home", for example, can be classified as Tale-Type No. 300, "The Dragon-Slayer", or as Tale-Type No. 301, "The Three Stolen Princesses". These additional functions can be assigned to appropriate nodes of the passage tree as shown in Appendix A. The resulting strep then encompasses the Aarne-Thompson classification as well as an augmented version of Propp's analysis.
5.2.3 The Passage-Dependent Network

Although the passage tree formed in the preceding sections includes all the information shown in Propp's descriptive string, it is not complete. Propp mentions the roles played by the characters in a story and the events in which they participate. His notation, however, does not encode these data. The passage-dependent network shown in Appendix A includes this information.

Just as Propp's summary neglects the repetition of the battles, it does not indicate that more than one dragon is involved in the story. Although the actual kidnapper is identified simply as a "dragon from the Black Sea," the oldest princess is rescued from a five-headed dragon while her sisters are held by monsters with seven and twelve heads. With nodes for each dragon (all descended from a single villain node), the passage-dependent network can easily describe the role of each dragon.

To illustrate other possible uses of the passage-dependent network in the analysis of folktales, Appendix A also identifies passages in which characters speak about each other and passages describing events that occur in different locations. Systematic inspection of these concepts for a large number of folktales might reveal structural rules for this class of texts in addition to those already identified by Propp.

5.2.4 The Passage-Independent Network

The passage-independent network shown in Appendix A encodes the relationships that define the functions served by various characters. For example, Frolik is a hero because he fights the dragons and rescues the princesses. The network also shows relationships among the various trebled elements: relative age of the princesses, number of dragons' heads, distance between each dragon's home and the tsar's kingdom. With the associations between characters and text segments provided in the passage-dependent network, and with the assumption that textual and temporal orders coincide in this story, these relationships in the passage-independent network show that the princesses are rescued in order of decreasing age, the dragons are encountered in order of increasing number of heads, and so on. Again, similar networks for other stories might reveal consistencies in these structural features across the genre.
5.3 When an Animal Grows

5.3.1 The Text

The second sample strep describes a nonfiction book for beginning readers. A children's book was selected as a short text that is nonetheless long enough to present a nontrivial example. To emphasize the generality of the streg notation, nonfiction was chosen in contrast with the fairy tale discussed above. Despite the fact that stregs are normally designed for a definite purpose, no specific application is discussed here. Stregs for children’s books could be used for many types of research. Classification of the structures of existing children's books (analogous to Propp's or Aarne-Thompson's classification of the structure of fairy tales) is one possibility. Psychological experiments dealing with children's recollection of stories is another; the structures of the recalled versions can be compared with that of the original. Structural factors could be incorporated into a measure of readability: can a child of specified reading ability comfortably read a particular story? Of course, more detailed texts intended for older audiences can have organizations similar to that of the text discussed here.

When an Animal Grows contrasts the growth rate of different animals. It begins with a description first of a newborn gorilla and then of a newborn lamb. Next it discusses the gorilla at three months and then returns to the lamb at a few weeks of age. The text continues to alternate descriptions of the two animals throughout their growth. A similarly-organized discussion of two species of birds follows.

A typical section about an individual animal begins with its age (e.g., "The baby gorilla keeps growing. He is six months old now.") and continues with a description (e.g., "He is still getting milk from his mother, but he is eating more and more plants"). Sometimes there are several text segments dealing with one growth period. The section about the newborn mammals describes the eating and sleeping habits of first the gorilla and then the lamb. It continues with a discussion of the position of the mother and baby gorilla within the band of gorillas followed by a description of the relationship of the lamb and her mother to the flock of sheep. Mention of the animals' age is not repeated in succeeding segments about each stage of growth.

Variations in the pattern of alternation occur. The gorilla matures more slowly than the lamb and some sections about the former have no counterpart describing the latter.
5.3.2 The Passage Tree

This text is easy to segment. In fact, the book is printed in two colors so that all passages relating to the gorilla (and the ducks) appear in brown ink and the sections describing the lamb (and the sparrows) are written in green. These color changes dictate some of the segmentation needed to form the passage tree. The hierarchy of passages shown in Appendix B is, however, more complex than a linear division by color alone.

Nodes near the root correspond to long sections that include both colors. The text following the brief introduction, for example, is divided into a section on mammals followed by a section on birds. The first subsection of the former deals with the newborn animals. This portion in turn is divided into a segment on eating and sleeping followed by a segment on relationships with other animals of the species. Each of these divisions is broken into a terminal segment in brown about the gorilla and one in green about the lamb.

The leaves of this part of the passage tree correspond to entire segments of a single color. However, in other subtrees, several terminal passages may be formed from one section printed in a single color. Typically, a single-color passage may be divided into a statement giving the animal's age followed by a description of its abilities at that age. Sometimes these descriptions are further segmented. In the discussion of birds, for example, the finer divisions of the passage tree are used to encode the fact that the same features are discussed in the same order for both species. The following section of the text is segmented into the named passages shown in Figure 5.5:

- a) The little sparrows are in a nest hidden in the grass. They are tiny and helpless.
- b) They have no feathers.
- c) They can't see.
- d) They can't walk.
- e) How different the ducklings are.
- f) They have soft little feathers.
- g) They can see.
- h) the mother duck leaves the nest. The little ducks follow.

Figure 5.4 Some Leafs in the Passage Tree
Figure 5.5 The Corresponding Subtree

Even with as rigorously organized a text as this one, it is not always clear how a particular passage should be segmented. Consider the following passages:

a) The baby gorilla gets less and less milk from his mother. By the time he is a year and a half old, he eats only stems, roots, and leaves...
b) The little lamb nurses only for a few months...
c) ...When she is a year old, she can have a little lamb of her own.
d) But the baby gorilla is still growing...

Figure 5.6 Weaning and Becoming Adult

Since b and c both deal with the lamb, the following subtree indicates a possible hierarchy over these passages:

This segmentation is supported by the author's use of colored ink: passage a forms an entire brown segment, b and c together form a green segment, and the following brown section begins with d.

Nevertheless, an alternate segmentation seems slightly preferable. Passages a and b both deal with weaning; c and d are concerned with reaching maturity. This section consists of the concatenation of segments on these topics, with the usual order of gorilla and then lamb reversed in the second subdivision. The hierarchy shown in Figure 5.8 reflects this grouping.
The structure of this section of the text cannot be completely described by a division into nonoverlapping segments. However, within a step, either tree can be used successfully, because whatever information is omitted from the passage tree can be supplied in the passage-dependent network. If Figure 5.7 is selected, the concept of "weaning" should relate passages a and b; if Figure 5.8 is used in the passage tree, the concept "lamb" should refer to both b and c.

5.3.3 The Passage-Dependent Network

Concepts in the passage-dependent network presented in Appendix B relate discontinuous passages with a common theme. Each species of animal has a node. The concept of "age" is represented by a node that points to every passage where the age of one of the animals is mentioned. The concept of "diet" links the passages that describe how the diet of each animal changes as it matures and also connects passages dealing with the diets of different species. Some more abstract concepts have been identified. For example, "moving" has descendants marked "standing", "walking", "crawling", "climbing", "swimming", and "flying". These specific types of movement eventually refer to passages describing the physical abilities of the different animals.

Although the network contains more than a hundred nodes, few of these concepts co-occur. In fact, most nodes in the passage-dependent network have only two parents. This situation results from the fact that most leaf passages describe only one feature of a single species. A concept referring to such a passage has one parent indicating the species and one parent indicating the feature. For example, when the concepts "sparrows" and "feathers" refer to a particular passage, none of the concepts, "ducks", "lamb", "seeing", and "being fed" refer to the passage as well.
5.3.4 Final Remarks

Although the names given to concepts in the passage-dependent network have strong connotations, the passage-dependent network and passage tree record the organization of the book rather than its content. Essential aspects of the meaning of the text are not encoded by these data structures. The theme of the book is explicitly stated in the first few pages:

Some animals grow slowly, like the gorilla that is helpless at first. And some grow fast, like the lamb. Birds grow in different ways, too.

The rest of the book presents evidence that animals grow at different rates. The cumulative effect of the statement and proof of an idea is not encoded in the stroph shown so far. It may be possible to represent this effect by a complicated construct within a traditional semantic network such as the passage-independent network.

However, no passage-independent network is shown in Appendix B. The passage tree and passage-dependent network adequately reflect the general form of the text. In a sense, the information shown is independent of the meaning of the text. A text with completely different subject matter could be organized in a similar fashion. With the names appropriately changed, nodes near the root of the passage tree and most of the passage-dependent network could still apply. For instance, a children's book describing the seasons of the year might begin, "Now it is winter. The trees have no leaves. Now it is summer. Branches are covered with leaves and the grass is green. In winter, it is cold outside. In summer, it is warm..." (In fact, many other existing texts have an alternating pattern. William Faulkner, for example, preferred to have his two novels, The Wild Palms, and The Old Man, published with the chapters intercalated. A stroph for this book, like the stroph for When an Animal Grows, should reflect this pattern).
6. The Structure of Programming Manuals

6.1 Introduction

This chapter investigates the structure of programming manuals. Constructs that frequently appear in texts of this class are discussed. Methods for representing these organizational patterns in the strep notation are presented along with a list of functions that identify these forms in passage trees and passage-dependent networks. Some of the functions mentioned here also apply to broader classes of documents. Cross-referencing, for example, which is discussed in detail in Section 6.4, appears in virtually every type of technical writing and in many fictional texts as well. The chapter concludes with the analysis of two short manuals. Although the selected documents are about the same length, their structures are very different. This difference is reflected in their streps, whose passage trees and passage-dependent networks are given in entirety in Appendices C and D.

6.2 Features of User Documentation for Computer Programs

It would not be possible to define a list of functions for reference manuals, analogous to Propp's work on folklore, that exclusively characterizes all material within a document and prescribes the order in which information can be conveyed. Nevertheless, many similarities can be observed in samples from a class of documents that includes tutorials on statistical packages intended for naive users as well as terse commentaries on formal definitions of structured languages. Figure 6.1 lists some types of material that frequently occur in technical documents. Figure 6.2 enumerates typical elements of programming documentation.
chapter/section/subsection
title/section heading
table of contents
index
appendix
glossary
figure
table
footnote
revision
erratum
introduction
conclusion
body
definition/description
example
acknowledgement
limitation
exception
bibliography
reference to another document
reference to another section (including "see above")
suggestion
intra-document conventions (schemes for numbering sections, notation, summary of organization, etc.)

Figure 6.1
Some Structural Features that Frequently Appear in Technical Documents

programs
files
data structures
control structures
identifiers
internal representation
input/output
operations/processes
options
syntax
semantics
subroutines
parameters
default values
calling sequence
order of commands
input conventions (e.g., columns 73-80 ignored; statements terminated by by ";", etc.)
conventions and style (e.g., indentation, conventions for choosing identifier names)
control cards
deck set-up	imesharing/batch
implementation
system change
hardware
cost

efficiency
error handling
error messages
debugging
maintenance
mechanics of running (e.g., where to submit a deck)
syntax summary
reserved words
character set
collating sequence

Figure 6.2
Typical Features of Programming Manuals
The entries in Figure 6.2 are very general. More specific features can be listed for restricted classes of documentation. Manuals on algebraic languages, for instance, contain sections on the assignment statement and the IF statement. Some features are specific to the documentation of a single manufacturer. Manuals for programs run on a UNIVAC 1100 series computer, for example, use "qualifier" and "element" in a manner that would be meaningless in another context. Some features of documentation are specific to an individual installation: many of the manuals produced at the Madison Academic Computing Center refer to a local concept, the "saved file". In all these cases, the concepts listed appear in many texts. Because of their frequency, it is important to recognize these features in an application that involves description of existing texts, generation of new texts, or location of material within texts.

6.3 Functional Labels in the Passage Tree

Most of the features mentioned in Figures 6.1 and 6.2 can be identified in a strep with functional labels on the nodes of the passage tree. Care must be taken with non-linear features such as footnotes, tables, and figures. As mentioned in Chapter 2, footnotes may be assumed to be embedded in the text at the point where they are first referenced. The same convention may be followed for tables and figures.

Passages often serve more than one function (e.g., a node may be labelled both "appendix" and "character set"); conversely, some functions are filled by several passages. There may be restrictions on the acceptable arrangements of passages with a given function. For example, only the left-most sibling of any subtree may be labelled "introduction". While this rule is easily described as a restriction on tree structure, rules for "appendix" are based on the linear nature of texts. A document may have any number of appendices. However, they must be grouped together; one follows the other after the main body of the text, but before any references or index. Restrictions on other features are less severe; definitions may appear almost anywhere except within bibliographic entries; footnotes may appear anywhere except within entries in a bibliography or within other footnotes.

Knowledge of these and similar restrictions is essential in generating texts; it can also be used to check for errors in proposed streps for existing texts. Similar
properties can be used in evaluating texts. For example, a text in which no term is used before it is defined may be more readable than one where this is not the case.

The features listed in Figures 6.1 and 6.2 cannot be adequately described with identifiers in the passage tree alone. Functional labels are used in the passage-dependent network to group and cross-list instances of these functions. For example, the concept graph for a particular command is likely to contain nodes marked "syntax", "semantics", and "examples".

Furthermore, a function which is used in the passage tree in some streeps may be used in the passage-dependent network for others streeps. In a particular text, all acknowledgements (or all examples) might be grouped in a separate section; in this case, the appropriate function should be indicated in the passage tree. However, these features may be scattered throughout a document, embedded in text sections that contain other material as well. In the latter situation, these contracts are best identified through the passage-dependent network.

6.4 Cross-Referencing

A "cross-reference" is a statement within a document about another part of the same text or about another text. This device is used by an author to suggest that relevant material exists elsewhere; some comprehension of the text is often required to locate the referenced passage. Cross-references occur very frequently in programming manuals. In the streep notation, a text is divided into passages, and it is natural to express cross-references as references to the passage tree.

Many cross-references, such as "explained in Chapter 6", "see Section 3.2", appear to identify the relevant node in the tree. On the other hand, "see page 37" does not fit a representation in which page boundaries are not encoded. In addition, this reference does not mean that all of page 37 is relevant to the current point; rather, it indicates that some logical portion, a passage, of page 37 is relevant. Similarly, "as will be shown below" could refer to any portion of the succeeding text. An author who uses one of these phrases assumes that the reader is capable of recognizing the intended passage with the help of a general pointer. Even references that seem to be quite explicit may, in fact, be vague. "Volume 13" may name an individual
passage in a tree; nevertheless, the sentence "This example is one of many similar cases treated in Volume 13" does not precisely pinpoint the location of the referenced discussion.

Several types of cross-references can be distinguished. In the first place, there are internal and external references. The former (e.g., "as will be shown below") indicate passages within the same document; the latter mention portions of some other text.

Secondly, there are essential and gratuitous references. An essential reference points to another section containing material that is relevant to the current passage but is not expressed within it; a gratuitous reference repeats information that is stated elsewhere. Thus, the sentence "The type of these parameters will be described in the next section" involves an essential reference; "as explained in the following section, the parameters to this routine are all integers" is a gratuitous reference. The writer of program documentation frequently must decide whether to repeat information included elsewhere or simply refer to it. Concise texts such as reference manuals characteristically avoid gratuitous references. Programming textbooks and other tutorial guides are more likely to duplicate important details.

A final dichotomy among cross-references is that of explicit versus implicit references. A semantic connection between two nonadjacent passages may exist in a text even if neither passage explicitly refers to the other. This situation is called an implicit cross-reference. Suppose a user reads that a program will prompt for a job priority. Two versions of the accompanying manual may be identical except that the second omits the sentence, "The options are explained in Section 4" that appears in the first. In the first case, the reference to Section 4 is explicit; in the second, it is implicit. The reader of either document is likely to wonder about appropriate responses to the program's prompt. This minor variation of wording should not affect the user.

An implicit reference may be assumed to exist between any two passages with a semantic connection. Another concept in the passage-dependent network often links the passages so that it is not necessary to explicitly encode the implicit cross-reference in a strep.

Cross-references inherently involve two passages: the referenced passage and the passage containing the reference. To represent references in a strep, the roles of both passages must be indicated. This relationship between passages can be encoded in the passage-independent network.
However, if an application does not require the passage-independent network for other reasons, cross-referencing alone does not warrant its inclusion. Furthermore, the passage-dependent network provides a convenient mechanism for representing the pointer that is the essence of a cross-reference.

To encode this information in the passage-dependent network, a node corresponding to each cross-reference is included in the network. This node has two children: one, with a modifier labelled "referenced passage" itself has a child that terminates at the referenced section; the other child, modified by a node labelled "referencing passage" is used to identify the referencing segment. For example, the concept graph in Figure 6.3 can be used to show that Section 1.0 refers to Section 2.0.

![Figure 6.3](image)

*Figure 6.3 Encoding Cross-References in the Passage-Dependent Network*

A more complex example, shown in Figure 6.4, presents an encoding for a manual where Sections 2.4 and 2.8 both refer to Section 3.0 while Section 3.0 in turn contains cross-references to Sections 1.7 and 4.2. An additional node "Cross-Reference" has been added in order to group all four references together.
6.5 Topic, Co-topic, and Mention

An idea can appear in a text as the primary focus of a passage, as one of several points treated in a passage, or as an issue mentioned in the context of some other subject. It is sometimes helpful to distinguish these possibilities in a step with the functional labels, "topic", "co-topic", and "mention". In steps used for generating texts, this qualification provides a mechanism for identifying the
material to be presented in each passage. It is also helpful in retrieving selected passages from existing texts.

The size of the divisions made in the passage tree influences whether a concept will be a topic or co-topic in a given passage. When a text is broken into small portions, it is more probable that the leaves of the passage tree are concerned with single topics. The concepts discussed in introductions, summaries, and so on, are usually co-topics.

When a concept is mentioned in one passage, but is the topic of another, a cross-reference links the first passage with the second. In the earlier example about prompting for job priorities, the cited passage mentions "job priority" and also cross-references a section concerning this topic.

6.6 Two More Examples

Appendices C and D give the passage trees and passage-dependent networks for two short programming manuals along with the bodies of the texts. The items listed in Figures 6.1 and 6.2, the categories of cross-reference, and "topic", "co-topic", and "mention" identify the function of nodes in the trees.

The first manual is the ASCII FORTRAN Supplement, available at the Madison Academic Computing Center, as a local addition to the vendor-supplied programmer reference manual. The primary purpose of the ASCII FORTRAN Supplement is to provide information needed by users who are more familiar with another compiler, FORTRAN V, than with ASCII FORTRAN. The document discusses features provided by ASCII FORTRAN that have no counterpart in FORTRAN V and steps for interfacing between the two compilers.

The second example is based on the LEXICO Maintenance Guide. LEXICO is a system for maintaining collections of texts, concording individual texts and classifying the words that appear in the concorded texts. Several features of the system simplify further development and debugging; this document describes these capabilities. The Maintenance Guide is an internal document not intended for general users.

The two documents are about the same length, roughly fifteen pages. Each assumes its readers are experienced programmers with a thorough knowledge of material not presented in the document. Despite these similarities, the structures of the two manuals are very different. Some of the differences are immediately obvious in the passage tree.
Only the ASCII FORTRAN Supplement has a table of contents and uses examples.

The major difference is in the interaction among the concepts contained in each text. The ASCII FORTRAN Supplement is a list of independent topics. The LEXICO Maintenance Guide has some sections dealing with various data files and others describing the use of several supporting programs. Explanations of how a particular file is affected by various programs occur throughout several passages. Similarly, the one-letter codes used to identify described options have different meanings to different programs. Since a reader may be interested in any combination of option-letter, program, and file, names for these various entities are used as modifiers in the passage-dependent network. The result of this difference between the two texts is that, while the passage-dependent network for the ASCII FORTRAN Supplement contains 98 nodes, there are 269 nodes in the network for the LEXICO Maintenance Guide.

These two streps are used to provide examples of interaction with the on-line documentation system proposed in the next chapter. A few comments on some of the decisions made while encoding these structures are included in Appendices C and D.

7. THUMB

"All the documentation in the world can’t replace somebody who knows what’s going on."

--overheard in the halls of a computing center
June 21, 1977

7.1 Introduction

This chapter describes the design of an on-line documentation system based on streps. The resulting set of programs, given the rather strained title, "Text Heuristics for Using Manuals Better," allows a user to "thumb" through a manual interactively. THUMB is designed to operate even on manuals that are not written specifically for on-line use. The system does not exist as of this writing. Implementation of a similar system, using aircraft maintenance manuals instead of programming manuals, is scheduled to begin in June, 1978.

1. The expansion of this acronym was kindly suggested by Prof. Raphael Finkel.
The system has two major components. The first subsystem, ENCODE, is used by an expert to build or modify a detailed strep for a document. No passage-independent network is needed in this application. The passage tree and passage-dependent network are supplemented by an extensive lexicon, relating keywords, key phrases (and their synonyms and plural and singular forms) to names and functions in the strep. The output from ENCODE is a file containing the lexicon, the passage-dependent network, the passage tree, the text segments associated with the leaves of the passage tree, and the necessary pointers for linking these elements. All arcs are represented by two-way pointers so that any part of the strep may be traversed in either direction. The following diagram indicates the flow of information through ENCODE.

![Diagram](image)

**Figure 7.1 Information Flow Through ENCODE**

The data structure output by ENCODE is used by PERUSE, the second component of THUMB. The user of this program, called the peruser, requests information interactively. He need know nothing about text structure, streps, or the decisions made by the expert who encoded a particular document. The peruser requests information using the vocabulary items stored in the lexicon. The lexicon provides access to the passage-dependent network through which pertinent portions of the passage tree are found. A successful search of the passage tree results in display of text segments associated with leaf nodes of selected subtrees. The operation of PERUSE is indicated in Figure 7.2 below.
On-line documentation systems not only provide a convenient means of obtaining an entire document, but can also help locate elusive information within a particular text. Moreover, descriptions of system changes can be readily incorporated into on-line documents; updating all copies of printed documentation is more difficult and expensive.

As more people with varying degrees of computing experience use interactive programs, there is a growing need for on-line documentation. On-line documentation is not, however, intended only for novices. The experienced programmer does not use a manual at every terminal session, but can occasionally forget the details of a particular process or be momentarily confused by similarities between different systems. All users can benefit from interactive assistance. In particular, as Truitt and Emery [71] note, the increasing use of computer networks emphasizes the need for adequate user services, including on-line documentation. Remote users may not have ready access to printed documents and cannot turn to consultants as conveniently as can users of local systems.

Even when written documentation is readily available, the process of finding the answer to a specific question can require considerable effort. An interactive program that performs some of the record keeping required while flipping
between an index and the corresponding text benefits even
the fortunate programmer with a terminal next to a full
bookcase of up-to-date manuals.

Some interactive systems use analogues of conventional
tools (the table of contents and index) for finding material
within a document. The SPEAKEASY language [12, 13]
developed at Argonne National Laboratory supplies on-line
assistance in the form of "Help Documents". Although
included as a bulky appendix to the user's manual, these
paragraphs are written specifically for interactive use and
are organized in a hierarchy whose tree structure is similar
to that of a passage tree. A user who enters "HELP" is
presented with a list of the major categories within the
SPEAKEASY vocabulary. "HELP" followed by the name of a
SPEAKEASY concept causes a display either of a list of the
subconcepts at the next level in the tree or of an
explanation of that term.

ZOG [49, 60], developed at Carnegie-Mellon University,
displays sections of written material in an order determined
by the individual user. This program is a generalization of
the PROMIS system implemented at the University of Vermont
for maintaining medical records. Robertson, Newell, and
Ramakrishna [60] report that PROMIS has been used
successfully by naive users. Each section displayed by ZOG
includes a list of relevant material that a user might wish
to view next. Since more than one display may suggest a
particular section as a possible next choice, the data are
organized as a network rather than as a tree. ZOG maintains
a stack to record the nodes that have been displayed during
an interactive session. At any point, the user may
backtrack through this stack. In addition, the user may
mark certain nodes and return to the identified displays at
a later time. These two features provide the user with a
simple method of following alternate chains of thought from
a common starting point.

Another program with some of THUMB's proposed
capabilities is SEARCHER, which is used at the University of
California at San Diego to dispense documentation on locally
produced software [30]. SEARCHER locates sections of
program write-ups to be displayed interactively or printed
as a batch process. The stored material is organized in a
tree structure. Each text section is given a multi-level
identifier (e.g., 7.2.8.16) that includes a numeric field
for each of its ancestors in the tree. In addition, a set
of keywords is associated with each section. The user can
ask for a list of the sections associated with any boolean
combination of keywords. Any section of a document can be
requested by its identifier. Any subsection of the current
focus (going down one level in the tree structure) is available without specification of its complete identifier. The user may also select material any number of levels above the current point. Finally, he may inspect portions of an alphabetized list of the keywords associated with a document.

The documentation for the algebraic language called FOSOL [24] is also organized in a tree structure, though its hierarchy is not as deep as in SPEAKEASY or ZOG. The FOSOL manual is divided into chapters that are identified by names without the conventional numbers. Each chapter is divided into sections called topics. The first topic in every chapter identifies the remaining topics. The command "HELP:" produces a list of chapter names. The user may also request HELP on an individual chapter, in which case the system prompts for topics. Copies of this interactively-oriented manual may be produced on a batch printer, but FOSOL has no other off-line documentation.

FOSOL has another interesting feature. Its interactive users may inspect any portion of the documentation at any time during a session. Furthermore, when certain types of errors occur, sections from the manual are automatically displayed.

Ideally, all interactive programs should have the capability of displaying on-line documentation. A user should not be required to end a session with a program in order to obtain information about it. FOSOL is not the only system to provide assistance of this type. Many other programs have a HELP command that explains error messages or briefly describes available commands. The LEXICO system [73, 74, 75, 76] provides several user aids. Whenever LEXICO issues an error message or prompts for data, the user may ask for one or more explanations of the message. This feature makes the system convenient for the novice without annoying the experienced user who requires less detail.

LEXICO's designers realized that the multiple explanations might not provide sufficient help for all users. The final level of each message therefore contains a reference to a specific section of the user's manual. (The LEXICO documentation was originally intended to be accessible interactively.) The inaccessibility of potentially useful material is not the only aspect of LEXICO's on-line documentation that can be improved. On rare occasions, the successively more detailed explanations can add to the user's confusion instead of alleviating it. During the development of LEXICO, a user once accidentally transmitted a line after typing only one character. This
character happened to be interpreted as a request for capabilities that the user did not know existed. The program responded with several questions that were incomprehensible to her. Unfortunately, the more detailed explanations she obtained dealt with the process she had inadvertently begun, and used vocabulary with which she was unfamiliar. She was therefore unable to return to the task she had been performing. PERUSE provides a user in similar circumstances with all necessary definitions. Given an initial message, it can provide more specific information (e.g., to a user trying to fix a syntax error) as well as more general information (e.g., to a user who is lost). The data encoded in a strip is used to locate the text sections containing these facts.

On-line documentation is supported by many other systems with commands similar to those of the varied programs described above. THUMB combines and expands these capabilities. PERUSE provides interactive users with all the information available in off-line documents. As will be shown in the next section, it responds to different types of requests in a consistent fashion and is easy to use. The tree-searching capabilities of several of the other programs are included in THUMB. As with SEARCHER [30], keywords can also be used to access material. PERUSE allows the reverse process as well; it can display a list of the concepts associated with a particular passage. The browsing capabilities provided by ZOG's stack are extended in PERUSE. While ZOG users search through a prepared network, the choices provided by PERUSE are dynamic, determined by the keywords in the user's request. In addition, PERUSE allows readers to follow cross-references and to relate on-line displays to hard-copy manuals by page number. The advantages of each of these features are enhanced by their combination.

7.3 PERUSE

7.3.1 Overview

This section describes the PERUSE program used to display passages of text. There are two ways to access PERUSE. The first possibility, in which PERUSE is a separately executable program, allows perusers to initiate a terminal session in order to obtain documentation. Figure 7.3 illustrates this implementation. Solid lines indicate the flow of information as a peruser inquires about some
program, System X. Dotted lines indicate possible flow for inquiries about other programs.

![Diagram of PERUSE as a Separate Program]

Figure 7.3 PERUSE As a Separate Program

The second approach, represented in Figure 7.4, allows any program in the system to invoke the capabilities of PERUSE. In this form, users can access PERUSE without interrupting a dialogue with another interactive system.

![Diagram of PERUSE Accessible from Other Programs]

Figure 7.4 PERUSE Accessible from Other Programs

The current proposal concentrates on the capabilities of THUMB and leaves the task of integrating this system into others for future work.

The usability of THUMB depends in large part on the syntax of PERUSE commands. The peruser is primarily interested in the target system; everything else he has to learn about THUMB is distracting. A restricted natural language format is desirable. PERUSE should be able to respond to questions like:

Where are parser options defined?
What options does the parser program have?
Define the possible options on the parser.

There is no need for PERUSE to accurately decode every input. When an ambiguous request is made, the peruser can be prompted for clarifying information. Keyword recognition schemes (where words in a request are ignored if they do not
appear in a special purpose dictionary) have been successful in similar situations. This approach was used, for example, in the GUIDE program for selecting PLATO IV computer science lessons [54] and in Project Genie [59].

Using similar techniques, PERUSE could translate natural language input into a more formal command language. As convenient as this would be for the user who is not familiar with the implementation of PERUSE, the sophisticated user (for example, an expert familiar with ENCODE) who sometimes finds natural language ambiguous and unwieldy might prefer to express requests more concisely in the command language. The peruser should be able to enter requests in either form, as indicated in the diagram below.

![Diagram](https://via.placeholder.com/150)

**Figure 7.5 Alternate Forms of Input**

In the following material, the capabilities of PERUSE are described in a tentative command language. The emphasis is on the effects of commands and not their syntax. Error handling, batch output, and options for long or short display formats, although important in a practical implementation, are not described here. As with the integration of PERUSE into other programs, the designs of the natural language interface and an extended command language are left for a later project.

Three general categories of commands exist in the command language: those that use the relationships in the passage-dependent network to define a set of related passages, those that refer to specific passages, and those that solicit information in the passage-dependent network pertinent to a particular passage. These commands can be explained without using the phrase "passage-dependent network". The documentation for PERUSE will simply state that a user may ask the program for a list of passages pertinent to selected concepts.

PERUSE sometimes identifies several passages or concepts that the user might wish to investigate further. For example, if a user inquires about a concept that is pertinent to non-adjacent passages, or asks for cross-references from a passage that refers to two or more other sections, the program produces a numbered list of the relevant passages. Similarly, if the peruser asks to see a passage that has several subpassages, the program lists these subpassages. When such a choice point is encountered,
the user may select any of the listed possibilities. The result may be another choice point, or a single passage. If it is a single passage, PERUSE indicates its length and asks the user if he wishes to have it displayed. Some commands -- FIRST CHOICE, NEXT CHOICE, etc. -- allow the user to inspect several alternatives from a given choice point. The RETURN command enables him to resume at a previous choice point. In addition, other commands -- FIRST SECTION, NEXT SECTION, etc. -- allow the peruser to scan adjacent sections of text in their written order.

The alternatives listed at a choice point may be concepts as well as passages. When inspecting a concept graph in the passage-dependent network, PERUSE lists modifiers of the concepts selected by the user. If the user's choice is one of these modifiers, the response is a list of children of the modifier that fall within the original concept graph. Again, these actions can be explained to the peruser without reference to a strep: PERUSE lists concepts that can be used to distinguish among the identified passages.

Although the statements in the command language are described later, a sample session is given immediately below to illustrate some of the system's salient features. The reader of this paper may wish to glance through the example now and return to it after reading the ensuing material. The dialogue is based on the strep for the LEXICO Maintenance Guide shown in Appendix D. User inputs are underlined and preceded by the prompt character, '>'. Displayed passages are surrounded by asterisks.

>ABOUT options

Related Concepts:
1. Description
2. Setting Parser Options
3. Parser Options that are Reserved Words
4. Subconcepts of "Individual Programs"
5. Subconcepts of "Individual Options"

Passages:
6. P9 Section 2.2 Options (13 lines)
7. P17 Section 3. Options (43 lines)

>2

Choice 2. P9 Setting Parser Options
Section 2.2 Options (13 lines)
Page 2 Line 19 through Page 3 Line 4
Display?

>YES

=================================================================
* Parser options (which are listed in Section 3.1) may
  be set on the control statement or with either of the
  commands
  *
  ROUTE 1005;
  OPTION olist;

Figure 7.6 Sample PERUSE Session (Page 1)
Both commands may be entered in any block. The former causes a display of all available options followed by prompts for the options to be changed. In the latter command, olist is a list of letters and numbers indicating options whose on/off flags are to be reversed. Reserved words appearing within olist must be enclosed in quotes. d and e are reserved throughout the system and t is reserved in EDIT blocks.

Choice 4. Subconcepts of "Individual Programs"

Choice 1. Parser Options

Related Concepts:
1. Setting Parser Options
2. Subconcepts of "Individual Options"
3. Subconcepts of "Files"

Passages:
4. P30 Section 2.2 Options (13 lines)
5. P20 Section 3.1 Parser Options (25 lines)

INTERACTING WITH log file

Passages:
1. P29 L Option (1 line) in Section 3.1 Parser Options
2. P31 N Option (1 line) in Section 3.1 Parser Options

LAST CHOICE

Choice 2. P31 N Option (1 line)
In Section 3.1 Parser Options
Page 6 Line 13
Display?

YES

CROSS-REFERENCE

P39 Section 4.2 The Log File (12 lines)
Page 9 Line 11 through Page 9 Line 22
Display?

YES
A log file, containing most system messages and all user input read by UGIN (or UGINA6), will be catalogued and saved for every run of LEXICO initiated without the N option whenever a master log file exists. A master log will be created and initialized, but not saved, whenever a run is initiated with the L option on and the N option off, unless there is an existing master log. The master log file, called 5603*LOG, contains its length in words in the first location. All following locations contain names of run logs in two-word packets. If no log has been catalogued, a temporary log may be created on unit 12 by setting option N equal to 0 after initialization.

*****************************************************************************

>CONCEPTS

Log File
Writing Files
Naming Files
Creating and Saving Files
Mention of L Option on the Parser
Mention of N Option on the Parser

>NEXT SECTION

P60 Section 4.3 The Statistics File (3 lines)
Page 10 Line 2 through Page 10 Line 4
Display?

>NO

Figure 7.6 Sample PERUSE Session (Page 4)
7.3.2 Display Conventions

7.3.2.1 Identifying Passages

Because the document to be inspected is also accessible off-line, every displayed passage must be identified in a way that simplifies transition between the interactive and off-line media. Page and line numbers provide some of the necessary identification. Section and chapter numbers and titles also help. Before displaying a passage, PERUSE identifies it with whatever names it has in the passage tree. Passages that are not explicitly marked in the original document are identified as a part of the smallest separately identified segment containing them. A final clue for locating a displayed passage within the printed version is provided by the passage numbers themselves. Because the nodes in the passage tree are numbered in symmetric order, the user can predict the relative position of two passages. P12, for instance, must either precede or include P29. On the third page of Figure 7.6, since P59 consists of exactly Section 4.2, it is listed as

P59 Section 4.2 The Log File (12 lines)
Page 9 Line 11 through Page 9 Line 22

In contrast, P31 is not a separate segment in the text. Therefore, it is described as

P31 N Option (1 line)
In Section 3.1 Parser Options
Page 6 Line 13

Since different conventions are used to label segments of written documentation, some care is necessary in providing a general implementation. In order to allow the peruser to access material by the names used in a particular text, the expert must inform ENCODE of the naming and numbering conventions used within it. "Chapter" and "section" are not the only words used to identify text segments. Segments from LEXICO's user documentation, for example, need to be identified by guide number as well. Neither titles nor section numbers can be assumed to be unique. Several chapters in one document may have a "Section 2"; the first display in Figure 7.6 shows that Section 2.2 and Section 3 of the LEXICO Maintenance Guide are both entitled "Options".

In some of the commands described below, the peruser may specify a particular passage. In such cases, it is assumed that the peruser may enter any relevant page, section, or passage number. The following are all valid possibilities:
7.3.2.2 Preventing Lengthy Displays

Without forcing a user to repeatedly respond to an "are-you-sure-you-want-to-see-this" prompt, PERUSE must protect the user from being inundated with large amounts of output. The definition of "large" is elusive in this context: its meaning is dependent on screen width, transmission speed, the user's mood, and the total document length. A sophisticated algorithm could employ terminal characteristics, user profiles, and suggestions from the expert who encoded the document. To minimize the number of lines shown, whenever a list contains three or more descendants of a single concept, the ancestral concept is displayed instead. Thus, at the beginning of Figure 7.6, the entry is

4. Subconcepts of "Individual Programs"

instead of a list of all the programs whose options are described in the text.

Redundancy is eliminated in lists of passages. If the passage-dependent network indicates that both a passage and one of its subpassages are pertinent to some concept, there is no need to list the subpassages. Similarly, instead of listing all the subpassages of a passage, only the parent passage is identified.

As further protection from accidental requests for lengthy and time-consuming displays, PERUSE indicates the amount of output to be expected. The identification of each passage includes a line count. The displayed number reflects the space required to list the passage at the terminal, rather than its volume in the original document.

7.3.2.3 Connecting Phrases and Section Headings

Some passages may consist solely of connecting phrases such as "on the other hand," or "This situation also exists in other contexts." There is no point in displaying these passages unless both the preceding and succeeding segments are listed as well. The passage numbers of connecting phrases need never be shown. The following dialogue would be slightly ridiculous:
Instead PERUSE would respond as follows:

If the user first viewed "First Point" and then "Second Point", the connecting phrase would automatically be displayed at the appropriate time.

In general, section headings can be treated as connecting phrases. Since the section heading is shown when PERUSE asks if the segment should be displayed, it is not necessary to repeat the heading when the passage is shown. If the peruser continues to read succeeding material, any additional titles encountered are displayed.

7.3.2.4 Other Conventions

In an actual implementation, conventions must be established for entering reserved words as data items, for entering a long command over several lines, and for entering several commands on a single line. Upper/lower case distinctions are usually ignored in PERUSE commands. Occasionally, the meaning of a keyword depends on such distinctions. In the LEXICO Maintenance Guide, for example, "The Parser" refers to a program; "PARSER" is a file containing the source code for that program. The expert must declare similar cases to ENCODE when establishing the lexicon. In ambiguous cases, or when receiving input from an upper-case only terminal, PERUSE prompts the user for the intended meaning.

PERUSE should provide a comment facility so the peruser can annotate a session. Finally, a mechanism should exist for posting messages to the expert who maintains a stref or to the writer who updates a document.

7.3.3 PERUSE Commands

7.3.3.1 Command Summary

The PERUSE commands are listed below. Brackets enclose optional phrases; braces delimit alternatives. Keywords are in capital letters; peruser-specified values in lower-case.

ABOUT expression
OCCURRING WITH expression
INTERACTING WITH expression
EXCEPT expression
OR expression
7.3.3.2 ABOUT and Its Variations

The ABOUT command described in this subsection requires the most explanation. The command's syntax is

```
ABOUT expression
```

where `expression` is formed from lexicon entries and the connectives, OCCURRING WITH, INTERACTING WITH, OR, and EXCEPT. As explained below, OCCURRING WITH and INTERACTING WITH may be interpreted as two forms of "AND". The keywords in the expression refer to names or functions in either the passage-dependent network or the passage tree. OCCURRING WITH, INTERACTING WITH, and EXCEPT have precedence over OR, but parentheses may be used to depart from this convention.

The ABOUT command causes PERUSE to display a list of all passages pertinent to the indicated concepts along with any additional concepts that can be used to distinguish among these passages. As a rule, the identified passages have the selected concepts as the topic or a co-topic. However, if the peruser wishes to see sections where the selected concepts are mentioned in some other context, he may include "mention" as one of the concepts in the boolean expression.

When the command is entered, PERUSE searches the lexicon for the concepts used in the expression. Assume momentarily that the lexicon entries all point to nodes in
the passage-dependent network. PERUSE follows the paths emanating from these nodes and builds a set of passages referenced at the ends of these paths according to the specified connectives. When two subexpressions are joined by INTERACTING WITH, the intersection of the paths determined by the subexpressions is used; paths joined by a co-occurrence node are treated as non-intersecting. When two subexpressions are joined by OCCURRING WITH, the intersection of the sets of references determined by the subexpressions is used. When two subexpressions are joined by OR, the union of the sets they determine is formed. Finally, all references determined by the second of two subexpressions joined by EXCEPT are discarded from those determined by the first.

The result of the ABOUT command is a two-part display. First, PERUSE lists identifiers of nodes encountered along the examined paths, except 1) those nodes used by the peruser in the expression and 2) those nodes discarded by use of EXCEPT. If an encountered node has no identifiers, the identifiers of its closest identified ancestors are displayed instead. The second part of the ABOUT output is a list of the passages in the final set.

For example, suppose a peruser wishes to ask the question

Which programs other than the Parser have an X Option?

He can enter the request as

ABOUT program INTERACTING WITH x option EXCEPT the parser

The appropriate portion of the passage-dependent network for the LEXICO Maintenance Guide is shown below.

![Diagram of the passage-dependent network]

Figure 7.8
The Relevant Portion of the Passage-Dependent Network

The dotted circles indicate nodes on the intersection of paths from "Programs" and "X Option". The pointer to P40 is discarded because it is descended from "The Parser". The nodes named "The Concorde" and "The Adder" are encountered on the traversed paths, so their identifiers are displayed.
The output from this command is shown in Figure 7.9. Only one passage is found, so PERUSE asks the user if the passage should be displayed.

Related Concepts:
1. The Concordor
2. The Adder

Passages:
3. P41 3.2 Adder and Concordor Options (5 lines)
   Page 6 Line 2 through Page 6 Line 33

Display?

Figure 7.9 Result of the ABOUT Command

Because this portion of the passage-dependent network does not contain multiple references to a single passage, the same results would be produced by the command ABOUT program OCCURRING WITH x option EXCEPT the parser. If, however, there is more than one reference to a single passage, the two connectives can produce different results. Suppose passage P5 in a revision of the LEXICO Maintenance Guide were to read as follows:

This document describes supporting programs (e.g., The Resolver and The Message Processor) and auxiliary files.

and that the passage-dependent network contains this subgraph.

![Figure 7.10 Subgraph with Two References to P5](image)

If the peruser enters

ABOUT example INTERACTING WITH auxiliary files
the null set results; paths descended from "example" and "auxiliary file" do not cross. However,

ABOUT example OCCURRING WITH auxiliary files
yields P5, since P5 is referenced on paths descended from each of these concepts.

The distinction between these connectives is related to that between category and co-occurrence nodes. INTERACTING WITH yields passages pertinent to a combination of the specified ideas (in this case, "example of auxiliary files"); while OCCURRING WITH lists passages where both concepts happen to appear. As long as valid lexicon entries are requested, PERUSE can never give a null response to an OCCURRING WITH expression. However, the only passage where two concepts co-occur may be the entire text; the peruser will be more interested in the smallest passages where the
concepts co-occur. When responding to the ABOUT command, PERUSE lists only the parent passage if the results of an INTERACTING WITH expression include all its children. Because of the information loss that would result, this simplification is not performed for an OCCURRING WITH expression. Perusers are likely to use INTERACTING WITH more often. OCCURRING WITH can be helpful in locating passages containing concepts interactions not encoded by the expert.

Entries in the lexicon may point to the passage tree as well as the passage-dependent network. This possibility is useful in two situations. First, it permits the peruser to use the ABOUT command to locate a passage by title alone. Second, it saves the expert the work of adding concepts to the passage-dependent network for each topic discussed in only a single portion of the text. PERUSE processes expressions that refer directly to the passage tree as though an extra category node existed in the passage-dependent network for each passage in the passage tree. An arc connects the assumed node for a passage with every terminal node in the passage-dependent network referring to that passage or to one of its subpassages. If no terminal node refers to a passage, one is generated for this purpose. The assumed node has no identifiers. The lexicon entry for any passage identifier points to the assumed node for the passage.

For example, the ASCII Fortran Supplement contains a section entitled "Cost Warnings to FTN Users". One of the subsections of this passage is called "Direct Access I/O". Although the concept "cost" is not encoded in the passage-dependent network, PERUSE would respond to the request

ABOUT cost AND direct access I/O with

P69 Direct Access I/O (7 lines)
Page 13 Line 6 through Page 13 Line 7
Display?

Lexicon entries may refer either to names or to functions. In some circumstances, some ambiguity may result. If a programming language has a DEFINE statement, the request

ABOUT define INTERACTING WITH parameter cannot be uniquely interpreted. Does the peruser want a definition of the word "parameter" or is he asking about the parameters of the DEFINE statement? An actual implementation must provide a simple means of disambiguating such requests.

The commands
INTERACTING WITH expression
OCCURRING WITH expression
EXCEPT expression
ON expression

qualify the result of a preceding ABOUT command. Expression
is an expression of keywords similar to that in an ABOUT
command. The result of the sequence
ABOUT expression1
INTERACTING WITH expression2
EXCEPT expression3

is equivalent to the single command
ABOUT expression1 INTERACTING WITH
expression2 EXCEPT expression3

However, each ABOUT command begins a new sequence that is
independent of all preceding commands.

Selection of a "related concepts" choice after an ABOUT
command is equivalent to the appropriate RESTRICT command.
In the first exchange from Figure 7.6, the entries
2
and

RESTRICT setting parser options

would produce identical results.

7.3.3.3 The CHOICE, SECTION, LEAF, and PASSAGE Commands

At any choice point, the peruser may enter the integer
number of a selected option. Alternatively, a CHOICE
command may be entered to select the first or last option,
or to make the decision relative to a previous choice from
the list. Figure 7.11 uses the ASCII Fortran Supplement to
illustrate some of the possibilities. (Note that since only
one passage is located by the original ABOUT command, PERUSE
immediately identifies its subpassages.)
ABOUT checkout INTERACTING WITH commands

Related Passages:
P44 in P40 CHECKOUT and FTNR (47 lines)
  1. P45 Call subprogram([arg1,...,argn])
     (7 lines)
  2. P46 DUMP[,opt] [var] [/unit] (8 lines)
  3. P47 Go [nL] (5 lines)
  4. P48 BREAK, CLEAR, SET, SETBP (7 lines)
  5. P49 SETBP[,opt] var[/unit] (15 lines)

LAST CHOICE

P49 SETBP[,opt] var[/unit] (15 lines)
  Page 10 Line 2 through Page 10 Line 16
  Display?

NO

CHOICE-

P48 BREAK, CLEAR, SET, SETBP (7 lines)
  Page 9 Line 31 through Page 9 Line 37
  Display?

NO

L

P47 Go [nL] (5 lines)
  Page 9 Line 26 through Page 9 Line 29
  Display?

Figure 7.11 Sample Dialogue Using CHOICE Commands

The SECTION commands allow the user to select various subpassages of a current passage. When any of the PERUSE commands so far discussed select a non-leaf passage, the result is a choice point listing the subpassages of the selected passage. This convention protects the peruser who enters

CHAPTER 2

from receiving a twenty-page display. Nevertheless, a peruser who really wishes to inspect a long passage may find it tedious to step through the tree structure. The LEAF commands display text segments associated with nodes along the portion of the frontier of the passage tree that descends from the specified passage. The syntax of the SECTION and LEAF commands parallel that of the CHOICE commands.

It is to be expected that the CHOICE, SECTION, and LEAF + and - commands will be used most often with the default value of 1 for the parameter n. There are two situations when it is useful to specify another value. One occurs when a segment begins with the warning that some readers may wish to skip a specified number of sections. The other situation occurs when the peruser who has read several consecutive passages wishes to reread one of the earlier ones (which may have disappeared from the screen) and then continue.

The peruser who knows the location of desired material need not bother with an ABOUT command and selection of one of the resulting choices. A segment may be requested by
passage number, section number or page number with the command

PASSAGE passage

If a page is named, PERUSE displays the leaf passage containing the first line on the specified page. With the
trep shown in Appendix D for the LEXICO Maintenance Guide,
the following commands all produce the same result:

PASSAGE P16
PASSAGE section 2.9
PASSAGE page 5

7.3.3.4 Commands for Browsing Through the Strep

The peruser may inspect portions of the passage-dependent network and passage tree. The SUB and
SUPER commands cause display of n generations of ancestors of specified nodes; n is either a positive integer or the
word ALL. The CONCEPTS command causes PERUSE to list the
modifiers (other than "Topic", "Co-topic", "Mention", or
descendants of "Cross-Reference") of all nodes in the
passage-dependent network that point to a current passage.

The CROSS-REFERENCE command transfers to a section
identified in an internal cross-reference within the current
passage. NOTE and FOOTNOTE are special cases of this

command. If only one cross-reference occurs, PERUSE asks
the user whether the referenced section should be displayed.
If the original passage contains more than one
cross-reference, PERUSE lists the referenced sections and
asks the peruser to choose among them. Of course, the
peruser is only able to access cross-references that have
been encoded in the strep by the expert. Implicit
cross-references may sometimes be available, however.

7.3.3.5 RETURN, CHECKPOINT, and STATUS

PERUSE maintains a stack describing the program's
status as each command is processed. The one-word RETURN
command is used to return PERUSE to a previous state. When
entered after any selection from a choice point, the RETURN
command causes PERUSE to resume at that choice point. When
entered after CROSS-REFERENCE, this command causes PERUSE to
return to the referencing passage. RETURN can also undo the
effect of a RESTRICT, REMOVE, or ADD command. After
SECTION+1, RETURN returns to the preceding section. The
results of RETURN are not always defined; RETURN makes no
sense after an ABOUT or PASSAGE command. The command

RETURN n
where \( n \) is an integer, causes the system to resume at the same point as \( n \) successive RETURN commands.

Of course, after a few levels, it becomes annoying to count the number of RETURN's needed. At any time, the peruser may enter

```
CHECKPOINT name
```

where `name` is an alphanumeric identifier containing at least one letter. Then, any number of commands later (including ABOUT or PASSAGE commands), the peruser may enter

```
RETURN name
```

to cause PERUSE to resume at the state where the CHECKPOINT command was entered. Because a checkpoint may be popped from the stack used by the RETURN \( n \) command, PERUSE maintains a separate list of checkpoints.

After a RETURN command, PERUSE displays a summary of the resulting state. For example, after a RETURN to a particular passage, PERUSE displays its passage number, section number, and title and the page and line numbers it occupies in the text; the text segment itself is not displayed. After a RETURN to an ABOUT command, PERUSE displays the expression used to define it; the resulting choices are not shown again.

The STATUS command is used to obtain more complete information after a RETURN -- a new prompt to display a portion of text or the list of choices at a choice point. The STATUS command may also be used after a sequence of OCCURRING WITH, INTERACTING WITH, EXCEPT, and OR commands to obtain a list of the complete expression defining the current state.

7.3.3.6 Table of Contents, Index, and Other Tables

Tables of contents, indices, command summaries, and lists of character sets, reserved words, or rates for different computing services constitute a special class of text components called tables. These structures often appear in appendices as well as within the body of a text. The LEXICO Maintenance Guide, for example, contains tables in the lists of options given in Section 3 and in the list of packet types in Section 5.2.

Although tables are not segments of running text, the division of a table into distinct passages is needed to select portions of the table by keyword. For example, the result of

```
ABOUT the parser INTERACTING WITH x option
```

is P40:

```
X (29)--display undefined packets
```
A peruser must be able to inspect any portion of a table. The necessary information can be encoded in a step if the table as a whole is delimited as a separate passage serving the function, "table". The subpassages of the table correspond to lines across it. In particular, the first subpassage in many tables serves the function, "column headings". Each line in the table is subdivided into its various fields. The fields are identified by line and column headings and numbers.

PERUSE has several commands for inspecting the first table in any specified passage or the current passage. The peruser may print any number of lines or columns and he may search for values in a specified field. Each field may be identified by line or column number or title.

The TABLE commands are also useful for searching through a manual's index. The peruser may easily inspect each of the pages listed in an index entry with a CROSS-REFERENCE command. An example from a hypothetical manual is shown in Figure 7.12 below.

> COLUMN 1 logical expression OF index

> logical expression.......63,75-78,112

>CROSS-REFERENCE

1. P38 Section 2.3 Logical Variables (20 lines)
2. P319 Section 6.4 Logical Expressions (67 lines)
3. P372 Section 7.1 IF Statements (25 lines)

>2

Choice 2. P319 Section 6.4
Page 75 Line 19 through Page 78 Line 8

Figure 7.12 Consulting the Index

7.3.4 Final Remarks

The commands listed in the previous section allow the peruser to locate any needed material easily. Although preceding chapters have dealt with the information to be encoded in a step, little has been said of the internal representation of the data structure. The design of the file formats and other conventions is a nontrivial project. However, several well-known techniques can be applied. The lexicon, for example, would presumably be stored as a random-access file from which pointers to the
passage-dependent network are retrieved using a hash-coding scheme. Some internal details are largely dependent on the operating system under which THUMB is implemented. Because of this dependence and because similar problems are solved in the development of every computer system using large semantic nets, this aspect of the implementation of PERUSE is not discussed here. Once the internal representation of a strep is designed, the implementation of PERUSE is a substantial but straightforward programming task.

PERUSE should record some information about its own use. The OLDS documentation system [53] used at New Mexico State University operates on the philosophy that writeups accessed most frequently are most important and therefore should be most frequently revised. That system records the number of times its users refer to each section. PERUSE can provide experts and writers with similar data. It can also record the number of times lexicon entries are used.

The program should keep track of requests it is unable to answer — keywords that do not appear in the lexicon and expressions in ABOUT commands that yield no passages. Such data reflect spelling errors on the part of the perusers; they also suggest revisions to the lexicon, passage-dependent network, and the text itself.

7.4 ENCODE

7.4.1 Overview

The availability of streps to PERUSE, and their effective design, depend on the ease with which they may be entered and maintained by the expert. Although the bulk of the work must be done by a human reader who understands the material, ENCODE is designed to remove as much drudgery from the task as possible. It is easier for an expert to encode a strep than to write a special on-line version of a manual. In the following pages, ENCODE is first described in relation to existing documents; its value as a writing tool is mentioned at the end of the chapter.

THUMB's second subsystem has several components. Once the expert establishes communication with ENCODE at an interactive terminal, he identifies the particular subprocess he wishes to perform. When the requested component involves tasks that can be processed quickly, or when it is convenient for ENCODE to prompt the expert for pertinent information, the process is performed interactively. When the requested task is costly because it requires scanning the entire text or printing large listings, ENCODE initiates a batch run that performs the
desired operation at a later time. Because there is one controlling interactive program, the expert can easily switch from one task to another. In addition, he need not memorize the formats of the various control cards needed to run the batch components of ENCODE -- these details are handled by the interactive component.

The feasibility of a mixed batch and interactive system accessed with a consistent means of communication is exemplified by LEXICO (73, 75). LEXICO uses another technique that is borrowed by ENCODE. A batch program produces a first approximation to an analysis that is not well-enough defined for complete computerization. Any resulting errors are corrected interactively. In ENCODE, this estimate consists of tentative processing of a document's index and table of contents as well as the attempt to locate cross-references, examples, and other selected concepts automatically.

The major components of ENCODE are

ENTERTEXT: a batch program that scans a machine-readable copy of a document, augmented by various commands that describe the format of the text and the formation of the passage tree. ENTERTEXT enters the passage tree and text segments into the strep. It makes tentative entries in the lexicon and passage-dependent network. It produces several printed listings.

NEWCONCEPTS: an interactive program that prompts the expert for instances of the concepts common to this kind of document and for passage references to be associated with the entered concepts. NEWCONCEPTS enters information into the lexicon and the passage-dependent network.

CHECK: an interactive process that cycles through the associations made by ENTERTEXT and DOCSSCAN, allowing the expert to verify or correct each one.

OFFLINELIST: a batch routine that produces an off-line listing of any component of the strep: the lexicon, passage-dependent network, passage tree, or text segments.

DOCSSCAN: a batch routine that is optionally initiated by NEWCONCEPTS. DOCSSCAN searches through the text segments for keywords entered into the lexicon by NEWCONCEPTS. As keywords are encountered, tentative passage references are added to the passage-dependent network. These references may later be verified through CHECK.

EDITSTREP: an interactive component that allows the expert to inspect, insert, delete, or modify any portion of the strep.

The operation of ENCODE is diagrammed below.
The ENCODE processes need not be executed in any given order. Although ENTERTEXT will usually be run only once, the other components can be used repeatedly during the process of encoding a single strep. Typically, ENTERTEXT and NEWCONCEPTS will be run before the others. If NEWCONCEPTS is run first, DOCSCAN and ENTERTEXT can be executed simultaneously. Off-line listings can be obtained whenever they are needed. CHECK can be used after each run of ENTERTEXT or NEWCONCEPTS. Some experts may not use NEWCONCEPTS or CHECK; they may prefer to enter the same data through EDITSTREP.

The design of the controlling interactive program makes it easy for the expert to alternate among CHECK, NEWCONCEPTS, and EDITSTREP. In addition, he may at any time test any portion of the strep by observing PERUSE's performance on the data.

The individual ENCODE processes are described below. Because ENCODE has so many more commands than PERUSE, and because the functions to be performed are more easily understood, the ENCODE subsystem is discussed here in less detail than was given to PERUSE in Section 7.3.

7.4.2 ENTERTEXT--Inputting the Text

Initial generation of the passage tree is done in batch mode using a program called ENTERTEXT. The expert supplies an input file that contains the text of the document, a description of its format, and a description of its segmentation. There are two types of records in this file: lines of text and ENTERTEXT commands. Some of the latter pertain to the appearance of the text. The PAGE command
indicates the start of a new page in the printed text. Since page numbers throughout a text are not always continuous, the page number, page, must be specified. Thus, page numbers of prefatory material (i, ii, iii, etc.), of inserted pages (2a, 2b, etc.), and of different chapters (2-1, 2-2, 3-1, 3-2, etc.) can be entered. Of course, conventions for automating regular sequences of page numbers could be established.

The SINGLE and DOUBLE SPACE commands and the SKIP command refer to the appearance of blank lines in the text. Each paragraph must be preceded by either a FIXED or a VARIABLE command. FIXED indicates that the succeeding material must remain in its original position on a line of text. Tables and sample programs, for example, have this property. VARIABLE indicates that horizontal spacing is not significant in the succeeding paragraph. When a paragraph is preceded by the VARIABLE command, PERUSE displays the passage starting at the left margin of the screen. It fills a line on the terminal with as many words from the printed text as room allows, and in this manner continues to print successive lines of the paragraph. Of course, a displayed passage may contain more than one paragraph.

Since the purpose of this chapter is to discuss a practical application of streps, every input problem has not been covered. The omitted issues include subscripts, superscripts, italicization, underlining, special characters, and footnotes. When portions from two lines in a printed manual are concatenated, PERUSE must know whether to retain or discard a line-final hyphen. A more substantive area is the inclusion of nonverbal figures. If PERUSE is to be used from a graphics terminal, the input to INIISTREP can contain digitizations of illustrations and commands describing how this material should be displayed. On a terminal without plotting capabilities, figures must either be omitted or described as accurately as possible with the available character set.

Along with the text and its formatting conventions, the input to ENTERTEXT shows the formation of the passage tree and the names and functions assigned to its nodes. A portion of the LEXICO Maintenance Guide, as prepared for ENTERTEXT, is shown in Figure 7.14. Parentheses delimit passages. The two parentheses in this sample indicate that Section 2 is a separate passage and that its first subpassage, Section 2.1, is a leaf. The SECTION and TITLE commands identify section numbers and section headings.
and underlined in the manual. The FUNCTION command identifies the purpose of the table.

```
TITLE TABLE OF CONTENTS
FUNCTION table of contents
FIELD 2-44,45-46
FTN Compiler Option Changes ........ 2
Collection of FTN Programs ........ 2
Interactive Postmortem Dump ....... 3
FTN Walkback Facility ............. 3

Figure 7.15 Entering a Table of Contents
```

7.4.3 Processing the Input File; CHECKing the Results

As ENTERTEXT scans the input file, it numbers the passages and creates the passage tree. The program updates the lexicon to include entries for the page numbers and section headings encountered. It counts the number of non-blank lines on a page and records beginning and ending page and line numbers for each passage. The printed output from ENTERTEXT has three parts. The first part shows the structure of the passage tree in outline form and the second
shows the text segment associated with each leaf passage. These listings are illustrated in Appendices A through D of this thesis. The third part of ENTERTEXT's output shows the lexicon entries made by the program.

ENTERTEXT looks for the expressions, "example", "for instance", and "e.g.", and marks passages that probably contain examples. It also searches for words that suggest explicit cross-references: mention of section or page numbers, "see", "following", "earlier", "above", and "below". When a phrase such as "described in Section 3.7" is found, ENTERTEXT creates a cross-reference from the current passage to Section 3.7. However, when a phrase such as "described below" is located, the program is unable to identify the referenced passage. In this case, it records the location of the cross-reference, so that the CHECK program can prompt the expert for the missing information.

When ENTERTEXT encounters a LABEL command, it processes the succeeding passage accordingly. While scanning a manual's table of contents, the program translates page numbers of named sections into passage references. ENTERTEXT enters each item in the index into the lexicon. It translates page numbers in index entries into passage numbers by assuming that the first passage on the specified page or range of pages is the one intended. These assumed cross-references are tentatively entered into the passage-dependent network.

As ENTERTEXT prints the text segments associated with each leaf passage, it notes in the margin any references it has created in the passage-dependent network pointing to that passage. The expert may inspect these marginal notes off-line and make any necessary corrections at a later time. If he chooses to enter the corrections with the CHECK routine, that program steps through a specified segment of text, displaying each generated item in the passage-dependent network that refers to the passage. As each item is shown, the expert may leave it as is, delete it, or modify it. A typical change is to sharpen a reference from a large separately-titled passage (e.g., Chapter 2) to a specific subpassage. If a manual has an index, the expert can correct any erroneous page references generated by ENTERTEXT. CHECK also prompts the expert for the information needed to complete vague cross-references (e.g., "see below").
7.4.4 NEWCONCEPTS

The NEWCONCEPTS program prompts the expert for names to be classified under various functions. Using the functions listed in Figures 6.1 and 6.2 -- "programs", "parameters", "options", "commands", "syntax", and so on, it asks the expert to enter names of concepts in selected categories. It also prompts for passages pertinent to the entered concepts. The program uses knowledge about the relationships among the entities usually discussed in programming manuals. If the expert enters the name of a subroutine, the program prompts for a list of its parameters. It asks the expert to identify the passages describing the operation of the routine, its calling sequence, and each parameter. Similarly, if the expert enters a command name, NEWCONCEPTS prompts for the location of descriptions of its syntax and of its semantics.

As the expert supplies each name, NEWCONCEPTS enters it into the lexicon. The program creates a node for the concept in the passage-dependent network, making the new node a descendant of the function concept. For example, if the expert declares a subroutine named "SORT", NEWCONCEPTS adds this node as a descendant of "Subroutine". In addition, as each passage reference is supplied, NEWCONCEPTS enters appropriate pointers in the passage-dependent network.

Figure 7.16 shows a dialogue between NEWCONCEPTS and the expert encoding the ASCII Fortran Supplement. The exchange consists of a series of questions asked by the program and the responses entered by the expert. Any time the expert is unsure of the response expected, he may enter "HELP" to request an explanation. Figure 7.17 shows the resulting portion of the passage-dependent network.
Function?
HELP
Enter one of the following:
programs
parameters
options
commands
data structures
(For other possible functions, enter HELP again)
COMMANDS
Command?
walkback
Command: Walkback
Syntax described?
P20
Semantics described?
P23
Examples?
P24
Other references?
YES
concept?
occurrences
where?
P17, P19
Other references?
NO
Figure 7.16 A Session with NEWCONCEPTS

NEWCONCEPTS is one of the few components of THUMB that is specifically oriented to programming manuals. A
possibility for future research is the design of a "meta-NEWCONCEPTS" program. Such a system would prompt for
the functions to be used by ENTERTEXT and NEWCONCEPTS. The user would also supply semantic rules for interpreting the
functions and the expected relationships among concepts in the entered categories. This program would make THUMB
available to readers of different types of documents.
7.4.5 DOCSCAN

Figure 7.17 does not indicate every passage in the ASCII Fortran Supplement that is pertinent to the concept, "Walkback". "Walkback" is mentioned in the table of contents and in the discussion of the compiler F option in P10. The expert does not have to encode the table of contents entry; this item is located by ENTERTEXT when that program processes the table of contents. The expert could have entered the reference to P10 through NEWCONCEPTS. However, in this case, he has chosen to let this reference be located by another program, DOCSCAN.

DOCSCAN searches through the text for occurrences of keywords stored in the lexicon by NEWCONCEPTS. In the example from the ASCII Fortran Supplement, once "walkback" is entered into the lexicon by NEWCONCEPTS, DOCSCAN can find the reference to this term in P10. Since references to P5 (the table of contents), P20, P23, P24, P17, and P19 were encoded by NEWCONCEPTS and ENTERTEXT, DOCSCAN ignores occurrences of the word in these passages.

The output from DOCSCAN is similar to the text-segment listing produced by ENTERTEXT. DOCSCAN also prints marginal notes showing the passage references it has generated.

After studying the DOCSCAN listing, the expert may use the CHECK routine to validate the data encoded by DOCSCAN.

7.4.6 EDITSTREP

The ENCODE programs described thus far are not sufficient to build an accurate strep. A manual may not have a table of contents or an index for ENTERTEXT to process. Even if it has both, the printed index is unlikely to show all the different groupings of concepts that can be useful in the passage-dependent network. The expert must have the opportunity to add keywords and concepts without relating them to the general functions known to NEWCONCEPTS. In addition, he must be able to modify all portions of the data structure.

The EDITSTREP program allows the expert to inspect, insert, delete, or modify any portion of a strep. Of all the programs that compose ENCODE, the expert will probably use EDITSTREP most often. To process revisions to a document as well as corrections to the original input file, EDITSTREP must have all the capabilities of a general-purpose text editor. In addition, it must be able to manipulate the arcs and nodes in the passage-dependent
network and passage tree as well as the pointers that link the pieces of a strep together.

The passage tree can be changed by combining sibling passages into a single node or by splitting a node into subpassages. When a passage is combined with a neighboring one, EDITSTREP changes references to either passage in the passage-dependent network so that they refer to the combined passage. When a passage is subdivided, EDITSTREP asks the expert whether references to the original passage should be left as is or changed to refer to one of the new subpassages. Names and functions in the passage tree can be changed, inserted or deleted; corresponding updates are made in the lexicon. When the expert revises a text segment, appropriate changes are automatically made to the page numbers and line counts stored with the passage tree. At the end of a session with EDITSTREP, consecutive passage numbers are assigned to the nodes in the passage tree; passage references in the passage-dependent network are updated to reflect the new numbers.

EDITSTREP has other commands that allow the expert to modify lexicon entries: he can specify new synonyms, change pointers to the passage-dependent network, and so on. In the passage-dependent network, the expert can inspect the ancestors or descendants of any concept. He can add or delete nodes or arcs; he can add, delete, or change node names or functions.

7.4.7 ENCODE as a Writing Tool

The use of THUMB presumes a machine-readable copy of the printed text. A significant portion of the expense of encoding a document is often due to preparation of the original input. However, as word-processing systems are used to produce an increasing amount of program documentation, the problem of generating machine-readable input is diminishing. In any case, it is hoped that the benefits provided by THUMB to the peruser far outweigh any development cost to the expert.

In a situation where production of an off-line text is partially automated, it is unnecessary to use a document-preparation system as a preprocessor for ENCODE. The former program uses some of the information that must be encoded into the input file for ENTERTEXT -- the word processing system must be able to underline, to use subscripts, to compute page numbers, to count the number of lines per page, to single or double space, and to recognize paragraphs to be printed in fixed or variable formats. The
expert should not be required to duplicate these specifications for different programs.

When the expert who encodes a document happens to be its author, time can be saved if ENCODE is implemented as an extension of a document-preparation system. By interspersing ENTERTEXT and formatting commands with textual material, the same input file can be used to produce both on-line and off-line documentation. If the format of the internal file on which EDITSTREP maintains the text segments is compatible with the text-preparation program, a revision entered through EDITSTREP can be cycled through the text-formating program to produce the off-line revision. In this way, neither the on-line nor the off-line document will ever be more up-to-date than the other version.

The merging of ENCODE with a word-processing system has other advantages. If the strep is completed before a final draft is printed, the passage-dependent network can be used to produce a thorough index automatically; the passage tree can generate a table of contents. Appendices listing all the commands or all the reserved words in a programming language can also be generated automatically from the passage-dependent network. The strep can be used to organize partially-written material. An author can use EDITSTREP to rearrange, cross-reference, and expand material.

Some printed manuals describe commands or routines in alphabetical order. ENCODE can sort text segments that are generated in a more natural order. Furthermore, it can retain the original order so that perusers can access the material by either arrangement. Freedom from linear ordering is one aspect of the "hypertexts" described by Nelson [49]. For documents restricted to interactive use, no fixed sequence of passages is needed; all access can be made through the passage-dependent network.
8. Other Applications

8.1 Introduction

The first criterion listed in Chapter 3 for measuring the success of the strep is that of practicality:

The resulting description should be economically usable in several practical applications.

In the previous chapter, an application to information retrieval was thoroughly described. Also, possibilities for assisting writers were mentioned. This chapter presents other uses for the strep.

8.2 Automatic Generation of Text

The writing tools mentioned in Chapter 7 presuppose that an author will provide both the content and the actual wording of each passage. Algorithms for generating stories have so far produced very condensed plot summaries rather than natural-sounding texts. Meehan [45] manually paraphrased the output of his text-generating program before reporting the results in his thesis. Prince [55] points out that the sequence of events produced by his transformational grammar can be embodied in media such as plays, films, or drawings as well as spoken or written language. Repetitious sentence structure and lack of detail cause the stories generated by Klein’s “meta-symbolic simulation system” [38, 39, 40] to seem like a list of sentences rather than a coherent text.

The output of these algorithms can be viewed as structural descriptions rather than complete texts. Even as structural representations, they are rather sparse. Only the events themselves are included. Point of view, emphasis, and the possibility of differences between temporal and textual ordering are ignored. Meehan acknowledges the possibility of automatically generating versions of a story that differ in order of presentation and amount of detail. Streps can be used to manipulate the alternatives. Klein’s system could also be enhanced by structural considerations. In fact, two of his simulations are based on structural analyses -- Propp’s characterization of Russian folktales [56] and Levi-Strauss’s comparison of the mythology of neighboring South American Indian tribes [43]. However, Klein’s data structure contains only events.
All considerations of text structure must be explicitly programmed for each simulation.

Even text-generators that do not produce polished natural language could account for structural features that are not considered by the systems mentioned above. The basic data structure for such a program would be a canonical form (with current techniques in artificial intelligence, expressed as a frame [46] or script [63]) for the streps to be used. Variables in this framework would include point of view, whether dialogue is to be quoted or summarized, whether conclusions are implied or explicitly stated, and so on. Concepts in the passage-dependent network for a story would include characters, time periods, and locations. A program could then select names to fill the specified functions.

If wording is to be automatically supplied, some variations in sentence structure could be selected according to structural considerations instead of on a purely random basis. For example, the active form of a statement might be generated if its subject (indicated by a pointer in the passage-dependent network) is the protagonist of a story, while the passive form would be produced otherwise.

8.3 Canonical Forms and Measures of Complexity

Canonical forms for streps have other uses. Propp's work consists of the identification of a canonical form for the structure of Russian folktale. Similar attempts can be made for other classes of texts, including programming manuals. Formalization of the "typical" computer manual, along with understanding of more common deviations, would reduce the effort needed to outline proposed documentation. In addition, such an analysis could be used in the development of documentation standards.

Another area for future research is the development of metrics for various aspects of structural complexity. Possible measurements include the degree of interconnection among the concepts in the passage-dependent network, the number of distinct concepts, the number of concepts per passage, word or syllable, and the frequency with which concepts occur in non-adjacent passages. To compare different streps for a single text, measurements of completeness and redundancy could be developed.

Measurements of structural complexity could be used to investigate readability; features could be identified that contribute to the time a reader needs to process a text, his difficulty in understanding it, and how quickly he forgets
its content. Metrics could also be used to discover the need for some types of revision. For example, corrections might be indicated if the value of some measurement is outside a pre-specified range or if it changes dramatically from passage to passage. Metrics could also be used in attempts to identify works of questionable authorship.

The varied tasks mentioned here are difficult. They do not all involve the same structural relationships. Nevertheless, the strep is intended to allow researchers in different areas to encode features relevant to them. Furthermore, the existence of a single notation that can emphasize different aspects of text structure should ease communication among investigators in different fields.

References

1. Antti Aarne and Stith Thompson, "The Types of the Folk Tale: A Classification and Bibliography" (Folklore Fellows Communications No. 74, 1928).


28. Norman Friedman, Form and Meaning in Fiction (University of Georgia, 1975).


44. David Locke and Alan Stewart, "Computer-Determined Readability Profiles", ACM SIGLASH Newsletter, 8(1975), No. 4, pp. 9-12.


56. Vladimir Propp, Morphology of the Folktale (University of Texas, 1968).


Glossary

acyclic network
A network in which no node is an ancestor of itself.

alphanumeric
Fertaining to characters, usually to a string of letters or digits.

ancestor
A node in a directed network that is a parent of a second node, or a parent of a parent of that node, etc.

category node
A node in a passage-dependent network that represents the interaction in meaning of its parent concepts.

child
A node in a directed network entered by an arc; the node from which the arc emanates is called the parent of the other node.

choice point
A situation during a dialogue with PERUSE in which the program displays several options and asks the user to select one.

completeness
The criterion that a representation of text structure should be applicable to entire texts and should be able to encode all needed structural relationships.

concept
A node in the passage-dependent network representing a semantic entity that recurs in a text.

concept graph
The portion of the passage-dependent network pertinent to a single concept; the concept's modifiers and subconcepts.

connected network
A network in which, for any two nodes, there is a list of nodes beginning with the first and ending with the second, such that every node on the list is either a parent or a child of the following node.

connecting phrase
A few words joining two passages; displayed by PERUSE only when both surrounding passages are also displayed.

continuous text
A sample of written natural language that forms a coherent whole.

coo-occurrence node
A node in a passage-dependent network that identifies segments of text relevant to all its parent concepts without indicating that the meanings of the parent concepts combine.

coo-topic
One of several subjects equally developed in a passage of text.

cross-reference
An expressed or implied suggestion within a passage of text that the reader might wish to consult another portion of the same text or a different text; also the ability to represent the recurrence of semantic entities throughout a text.

descendant
A node in a directed network that is the child of another node, or the child of a child of the second node, etc.

directed network
A set of nodes and arcs, each arc emanating from one node and entering another.

ENCODER
One subsystem of THUMB; used by an expert to prepare a document for PERUSE.

essential cross-reference
Reference to a passage explaining material that is relevant to a current passage but not presented within it.
expert
Person who uses the ENCODE subsystem of THUMB.

explicit cross-reference
Explicit identification within a portion of a text of another passage.

external cross-reference
A reference within one text to another document.

flexibility
The criterion that a notation for representing text structure should allow different descriptions of a single text, emphasizing different features and providing different details.

frontier
The set of leaves in a tree.

function
The purpose of a structural element in a text, expressed in terminology pertinent to a class of texts (e.g., hero, introduction, example); also, the criterion that representations of text structure encode the function of structural elements.

functional label
An identifier attached to a node in a strep to indicate its function.

generality
The criterion that representations of text structure be applicable to different classes of text.

gratuitous cross-reference
A cross-reference to a passage that repeats information contained in the current passage.

identifier
A label attached to a node in a strep; either a name or a functional label.

implicit cross-reference
An implied suggestion that a reader consult another passage; mention of material that is explicated elsewhere.

incrementality
The criterion that a representation of text structure should be modifiable in conjunction with revision to the represented document.

internal cross-reference
A reference to another passage within the same document.

leaf
A node in a tree that has no children.

leaf passage
A leaf node in a passage tree; a unit of text that is not further subdivided in a particular strep.

lexicon
A list of keywords and key phrases used by THUMB to identify names and functions in a strep.

linear text
A sample of written natural language in which each word except the first has a unique following word; opposed to nonlinear text such as footnotes or tables.

mention
A reference to a subject that is not one of the primary topics of a passage of text.

modifier
A node in the passage-dependent network used to classify subconcepts of other nodes.

name
A name given to a node in a strep to identify it with terminology used in the text (e.g., the name of a character or chapter).

order
The criterion that representations of text structure encode the order in which information is presented in a text.

parent
A node in a directed network from which an arc emanates; the node where the arc enters is called the child of the original node.
passage
A node in a passage tree or the associated segment of text.

passage-dependent network
One of the three parts of a streg: a directed acyclic network encoding the interaction of semantic entities within particular portions of a text.

passage-independent network
One of the three parts of a streg: a semantic network encoding relations pertinent to the entire text rather than isolated passages.

passage number
An identification number assigned by ENCODE to a node in a passage tree.

passage tree
One of the three parts of a streg: a tree showing the hierarchy formed by a text's chapters, sections, paragraphs, and so on. Each division or passage may be identified by one or more names or functional labels.

PERUSE
One subsystem of THUMB; used by perusers to inspect a document interactively.

peruser
A person who uses the PERUSE subsystem of THUMB.

practicality
The criterion that representations of text structure be usable in practical applications.

root
The unique node in a tree that has no parent.

stack
A list of items arranged so that the last item entered is the first to be retrieved.

streg
Structural representation; a notation in which the structure of a text is represented by a passage tree, passage-dependent network, and passage-independent network.

subconcept
A descendant of a node (or concept) in the passage-dependent network.

tagmemics
A linguistic theory in which every syntactic unit is identified by both purpose (slot) and form (filler), for example, as a subject (slot) has the form of a noun phrase (filler).

text structure
The relations among the semantic and syntactic entities within a natural language text.

THUMB
A two-part computer system for on-line documentation. With one component, an expert prepares a manual for interactive inspection by the second component.

topic
The unique subject of a passage of text that has such a primary focus.

tree
A connected, directed network in which no node has more than one parent.

Turing test
A test of modelling of human behavior by machine; successful imitation is achieved if, hidden from both a person and a computer, a human questioner cannot identify them.
Appendix A

This appendix gives a step for the Russian fairytale, "Frolka Stay-at-Home", along with a segmentation of the text. The passage tree is shown below. The passages have arbitrarily been assigned the identifiers P1, P2, etc. Functions are given after these identifiers. Parentheses have been used to delimit successive Proppian labels of a single passage; brackets delimit tale-types. The tree structure is shown in outline form: the immediate descendants of each passage are indented beneath it.

P1  Frolka Stay-at-Home
    P2  Title
    P3  Body
       P4  Setting
       P5  Absentee of Younger Generation and
            Violation of Implied Interdiction
       P6  Villainy in the Form of Abduction
           [Motif 301 The Three Stolen Princesses]
       P7  Mediation in the Form of a Call for Help
       P8  Counteraction
       P9  Consent to Counteraction
       P10 Departure of Heroes
       P11 Rescuing the Princesses
       P12 Rescue of the First Princess
       P13 Finding the Princess (Quest)
       P14 Struggle in the Form of Battle in Open Field
       P15 Beginning of Struggle
       P16 Villain Beaten in Open Combat [Motif 300 The Dragon-Slayer]
    P17 Object of Quest Obtained as Direct Result of Preceding Action
    P18 Rescue of the Second Princess (Quest)
    P19 Finding the Princess (Quest)
    P20 Struggle in the Form of Battle in Open Field
    P21 Beginning of Struggle
    P22 Villain Beaten in Open Combat [Motif 300 The Dragon-Slayer]
    P23 Object of Quest Obtained as Direct Result of Preceding Action
    P24 Rescue of the Third Princess (Quest)
    P25 Finding the Princess (Quest)
    P26 Struggle in the Form of Battle in Open Field
    P27 Beginning of Struggle
    P28 Villain Beaten in Open Combat [Motif 300 The Dragon-Slayer]
    P29 Object of Quest Obtained as Direct Result of Preceding Action
    P30 Return of Hero
    P31 Monetary Reward

The text segments associated with each terminal passage are shown below. Passages are identified with the numberings given to nodes in the passage tree as well as by page and line references to the Pantheon Books edition.
There was once a king who had three daughters, and such beauties they were as no tongue can tell of nor pen describe. Their garden was big and beautiful and they liked to walk there at night. A dragon from the Black Sea took to visiting this garden.

One night the king's daughters tarried in the garden, for they could not tear their eyes away from the flowers; suddenly the dragon appeared and carried them off on his fiery wings. The king waited and waited but his daughters did not come back. He sent his maidservants to look for them in the garden, but all in vain; the maidservants could not find the princesses.

The next morning the king proclaimed a state of emergency and a great multitude of people gathered. The king said: "Whoever finds my daughters, to him I shall give as much money as he wants."

Three men agreed to undertake this task—a soldier who was a drunkard, Frolka Stay-at-Home, and Erema;
and they set out to look for the princesses.

They walked and walked till they came to a deep forest. As soon as they entered it they were overwhelmed by drowsiness. Frolka Stay-at-Home drew a snuffbox out of his pocket, tapped on it, opened it, shoved a pinch of tobacco into his nose, and cried: "Eh, brothers, let us not sleep, let us not rest, let us keep going." So they went on; they walked and walked and finally came to an enormous house, and in that house was a five-headed dragon. For a long time they knocked at the gate, but no one answered. Then Frolka Stay-at-Home pushed the soldier and Erema away and said: "Let me try, brothers!" He snuffed up some tobacco and gave such a knock at the gate that he smashed it.

They entered the yard, sat in a circle, and were about to eat whatever they had. Then a maiden of great beauty came out of the house and said: "Little doves, why have you come here? A very wicked dragon lives here, who will devour you.

You are lucky that he happens to be away." Frolka answered her: "It is we who shall devour him."

He had no sooner said these words than the dragon came flying and roared: "Who has ruined my kingdom? Do I have enemies in the world? I have only one enemy, but his bones won't even be brought here by a raven." "A raven won't bring me," said Frolka, "but a good horse did." The dragon upon hearing this said: "Have you come for peaceful purposes or to fight?" "I have not come for peaceful purposes," said Frolka, "but to fight."

They moved apart, faced each other, and clashed, and

in one stroke Frolka cut off all the five heads of the dragon. Then he put them under a stone and buried the body in the ground.
P17 Object of Quest Obtained as Direct Result of Preceding Action
Page 300 Line 30

The maiden was overjoyed and said to the three brave men: "My little doves, take me with you" "But who are you?" they asked. She said that she was the king's eldest daughter; Frolka told her what task he had undertaken, and they were both glad. The princess invited them into the house, gave them meat and drink, and begged them to rescue her sisters. Frolka said: "We were sent for them too!" The princess told them where her sisters were. "My next sister is even worse off than I was," she said. "She is living with a seven-headed dragon." "Never mind," said Frolka, "we shall get the better of him too; it may be somewhat harder to deal with a twelve-headed dragon." They said farewell and went on.

P19 Finding the Princess (Quest)
Page 301 Line 4

Finally they came to the abode of the second sister. The house where she was locked up was enormous and all around it there was a high iron fence. They approached it and looked for the gate; finally they found it. Frolka banged upon the gate with all his strength and it opened; they entered the yard and, as they had done before, sat down to eat.

P21 Beginning of Struggle
Page 301 Line 10

Suddenly the seven-headed dragon came flying. "I smell Russian breath here," he said. "Bah, it is you, Frolka, who have come here! What for?" "I know what for," answered Frolka. He began to fight with the dragon and

P22 Villain Beaten in Open Combat [Motif 300 The Dragon-Slayer]
Page 301 Line 13

in one stroke cut off all seven of his heads, put them under a stone, and buried the body in the ground.

P23 Object of Quest Obtained as Direct Result of Preceding Action
Page 301 Line 14

Then they entered the house; they passed through one room, a second, and a third, and in the fourth they found the king's second daughter sitting on a sofa. When they told her why
and how they had come there, she brightened, offered them food and drink, and begged them to rescue her youngest sister from the twelve-headed dragon. Frolka said: "Of course, that is what we were sent for. But there is fear in my heart. Well, perhaps God will help me! Give us each another cup!"

They drank and left; they walked and walked till they came to a very steep ravine. On the other side of the ravine there stood enormous pillars instead of a gate, and on the pillars were chained two ferocious lions that roared so loudly that only Frolka remained standing on his feet; his two companions fell to the ground from fear. Frolka said to them: "I have seen worse terrors, and even then I was not frightened. Come with me!" And they went on. Suddenly an old man, who looked to be about seventy, came out of the castle; he saw them, came to meet them, and said: "Whither are you going, my friends?" "To this castle," answered Frolka. "Ah, my friends," said the old man, "you are going to an evil place; the twelve-headed dragon lives in this castle. He is not at home now, else he would have devoured you at once." "But he is the very one we have come to see," said Frolka. "If so," said the old man, "come with me, I will help you get to him." The old man went up to the lions and began to stroke them, and Frolka with his companions got through to the courtyard.

They entered the castle; the old man brought them to the room where the princess lived. Upon seeing them she quickly jumped off her bed and began to question them as to who they were and why they had come. They told her. The princess offered them food and drink and began to make ready to go.

As they were preparing to leave the house, they suddenly saw the dragon flying at a verst's distance from them. The king's daughter rushed back into the house and Frolka and his companions went out to meet and fight the dragon. At first the dragon attacked them with great force, but
Frolka, a clever fellow, managed to defeat him, cut off all of his twelve heads, and cast them into the ravine. Then they returned to the house.

and in their joy revelled even more than before. Following this feast they set out on their way, stopping only for the other princesses.

Thus they all came back to their native land.

The king was overjoyed, opened his royal treasury to them, and said: "Now, my faithful servants, take as much money as you want for a reward." Frolka was generous: he brought his big three-flapped cap, the soldier brought his knapsack, and Erema brought a basket. Frolka began to fill his cap first: he poured and poured, the cap broke, and the silver fell into the mud. Frolka began to pour again; he poured, and the money dropped from the cap! "There is nothing to be done," said Frolka. "Probably all of the royal treasury will fall to me." "And what will be left for us?" asked his companions. "The king has enough money for you too," said Frolka. While there was still money, Erema began to fill his basket, and the soldier his knapsack; having done this, they went home. But Frolka remained near the royal treasury with his cap and to this very day he is still sitting there, pouring out money for himself. When his cap is filled, I shall go on with my story, but now I am too tired.

The passage-dependent network is shown below. The nodes have arbitrarily been assigned the identifiers N1, N2, etc. Most nodes are category nodes. Nodes (e.g., N7) marked with an asterisk are co-occurrence nodes. Labels are shown in brackets. Unlabelled nodes have been given descriptive names. Under each node is a list of its
immediate descendants within the passage-dependent network
and the passages in the passage tree to which it points.

N1 [Characters]
  N2, N3, N8, N13

N2 King [Tzar]
  P4, P6, P7, P31

N3 [Princesses]
  N4, N5, N6

N4 First Princess
  P12, N7

N5 Second Princess
  P18, N7, N20

N6 Third Princess
  P24, N7, N21

N7° Co-occurrences of Princesses
  P4, P5, P6, P30, N19

N8 [Heroes]
  N9, N10, N11

N9 Froika Stay-at-Home
  N12

N10 Soldier
  N12

N11 Erema
  N12

N12° Co-occurrences of Heroes
  P8, P30, P31

N13 [Villains]
  N14, N15, N16, N17

N14 Dragon from the Black Sea
  P4, P6

N15 Five-Headed Dragon
  P14, N22

N16 Seven-Headed Dragon
  P20, N23

N17 Twelve-Headed Dragon
  P26, N24

N18 Characters Discussed by Others
  N19, N20, N21, N22, N23, N24

N19 Discussion of the Princesses
  P7

N20 Discussion of the Second Princess
  P17

N21 Discussion of the Third Princess
  P17, P23

N22 Discussion of the Five-Headed Dragon
  P13

N23 Discussion of the Seven-Headed Dragon
  P17

N24 Discussion of the Twelve-Headed Dragon
  P17, P23, P25

N25 [Places]
  N26, N27, N28, N29

N26 Tsar's Kingdom
  P4, P5, P6, P7, P9, P10, P30, P31

N27 Kingdom of the Five-Headed Dragon
  P12

N28 Kingdom of the Seven-Headed Dragon
  P18

N29 Kingdom of the Twelve-Headed Dragon
  P24
Some of the information that might be indicated in a passage-independent network for Frolka Stay-at-Home is shown below. The generic form of each item is subject-relation-object. The relationships are grouped by subject; objects are underlined. To increase readability by normal English word order, acts are listed as relations with their agent as subject; other cases (and attributes) are identified in parentheses. Instead of arbitrary values for the attribute, "age", the relationship, "older than", is used to indicate the relative ages of the princesses. Relative distances among the various places mentioned in the story are similarly encoded.

Correspondence to the text is not indicated here. The passage-dependent network could be used to relate a node for each act to specific text segments.

Frolka Stay-at-Home
leader of soldier
leader of Erema
defeats (location: home of Five-Headed Dragon)
Seven-Headed Dragon
defeats (location: home of Seven-Headed Dragon)
Seven-Headed Dragon
defeats (location: home of Twelve-Headed Dragon)
Third Dragon
rescues (location: home of Five-Headed Dragon)
First Princess
rescues (location: home of Seven-Headed Dragon)
Second Princess
rescues (location: home of Twelve-Headed Dragon)
Third Princess

soldier
fights (location: home of Five-Headed Dragon)
Five-Headed Dragon
datafights (location: home of Seven-Headed Dragon)
Seven-Headed Dragon
datafights (location: home of Twelve-Headed Dragon)
Third Dragon
rescues (location: home of Five-Headed Dragon)
First Princess
rescues (location: home of Seven-Headed Dragon)
Second Princess
rescues (location: home of Twelve-Headed Dragon)
Third Princess

Erema
fights (location: home of Five-Headed Dragon)
Five-Headed Dragon
datafights (location: home of Seven-Headed Dragon)
Seven-Headed Dragon
datafights (location: home of Twelve-Headed Dragon)
Third Dragon
rescues (location: home of Five-Headed Dragon)
First Princess
rescues (location: home of Seven-Headed Dragon)
Second Princess
rescues (location: home of Twelve-Headed Dragon)
Third Princess

dragon
kidnaps (location: Tsar's Kingdom) First Princess
kidnaps (location: Tsar's Kingdom) Second Princess
gives (object: First Princess, recipient: Five-Headed Dragon, location: home of Five-Headed Dragon)
gives (object: Second Princess, recipient: Seven-Headed Dragon, location: home of Seven-Headed Dragon)
gives (object: Third Princess, recipient: Twelve-Headed Dragon, location: home of Twelve-Headed Dragon)

Five-Headed Dragon
lives in Home of Five-Headed Dragon
has heads (number: 5)

Seven-Headed Dragon
lives in Home of Seven-Headed Dragon
has heads (number: 7)
Twelve-Headed Dragon
lives in Home of Twelve-Headed dragon
has heads (number: 12)

First Princess
older than Second Princess

Second Princess
older than Third Princess

Tsar
father of First Princess
father of Second Princess
father of Third Princess
offers reward (purpose: rescue of Princess, receipient:
anyone, location: Tsar's Kingdom)

rewards (object: money, receipient: Froika, location:
Tsar's Kingdom)

rewards (object: money, receipient: Erema, location:
Tsar's Kingdom)

rewards (object: money, receipient: soldier, location:
Tsar's Kingdom)

Home of Five-Headed Dragon
far from Tsar's Kingdom

Home of Seven-Headed Dragon
far from Home of Five-Headed Dragon

Home of Twelve-Headed Dragon
far from Home of Seven-Headed Dragon

This appendix shows the passage tree and the
passage-dependent network of a strip for the children's
book, When an Animal Grows. The passage tree is shown
below. Each passage has been given a name to indicate
its contents. Leaf passages have also been given the names
"brown" or "green", shown in parentheses, to indicate
whether they are printed in brown or green ink in the text.
In addition, the arbitrary identifiers P1, P2, ... have
been assigned to the passages. The tree structure is shown
in outline form; the immediate descendants of each node are
indented beneath it.

P1 When an Animal Grows
P2 Title
P3 Body
P4 Introduction
P5 (Brown) Story of a Gorilla
P6 (Green) Story of a Lamb
P7 Animals Grow at Different Rates
P8 Mammals
P9 (Brown) Gorilla
P10 (Brown) Lamb
P11 (Brown) Mammals Grow at
Different Rates
P12 (Brown) Gorilla
P13 (Green) Lamb
P14 Birds
Birds Grow at Different Rates
(Green) Sparrows
(Brown) Ducks
(Green) All Animals

Newborn
Eating and Sleeping
(Brown) Gorilla
(Green) Lamb
With Other Animals
(Brown) Gorilla
(Green) Lamb

Straying from Mother's Side
Gorilla
(Brown) Age
(Brown) Description
Lamb
(Green) Age
(Green) Description
Diet
Gorilla
(Brown) Age
(Brown) Description
(Green) Lamb

Gorilla at 6 Months
(Brown) Age
(Brown) Description

Childhood
Playing
Follow-the-Leader
Gorilla
(Brown) Age
(Brown) Description
Lamb
(Green) Age
(Green) Description
King of the Mountain
(Green) Gorilla
(Green) Lamb
Fighting
(Brown) Gorilla
(Green) Lamb
Playing Alone
(Brown) Gorilla
(Green) Lamb
Spending Time with Mother

Eggs
(Green) Sparrows
(Brown) Ducks
Hatching
Sparrows
(Green) Age
(Green) Pecking
(Green) Number
(Green) Variety
Ducks
(Brown) Age
(Brown) Pecking
(Brown) Number
(Brown) Variety

Newly Hatched
Sparrows
(Green) Introduction
(Green) Feathers
(Green) See
(Green) Walk
Ducks
(Brown) Introduction
(Brown) Feathers
(Brown) See
(Brown) Walk
Diet
Sparrows
(Eating)
(Green) Father's Help
by Mother
Ducks
(Eating)
The text segments associated with each terminal passage are shown below. Passages are identified with the numberings given to nodes in the passage tree as well as by page and line references to the text.

P2  Title
Page 1  Line 1
When an Animal Grows

P5  (Brown) Story of a Gorilla
Page 6  Line 1
This is the story of a baby gorilla. He is very little and weak. He is one day old.

P6  (Green) Story of a Lamb
Page 7  Line 1
And this is the story of a lamb that grows up to be a sheep. This lamb was born just half an hour ago. But already she can stand up on her wobbly legs.
P9  (Brown) Gorilla
Page 8  Line 1
The baby gorilla will grow.

P10  (Brown) Lamb
Page 8  Line 2
And the lamb will grow.

P11  (Brown) Mammals Grow at Different Rates
Page 8  Line 3
But they will grow in different ways.

P12  (Brown) Gorilla
Page 8  Line 4
Some animals grow slowly, like the gorilla that is helpless at first.

P13  (Green) Lamb
Page 8  Line 6
And some grow fast, like the lamb.

P15  (Green) Birds Grow at Different Rates
Page 9  Line 1
Birds grow in different ways too.

P16  (Green) Sparrows
Page 9  Line 2
When a sparrow comes from an egg, it can't see and has no feathers.

P17  (Brown) Ducks
Page 9  Line 4
But a duck walks and swims after its mother, soon after hatching.
Either way, an animal grows. It gets bigger and bigger. And before long it is grown-up and able to take care of itself.

The baby gorilla's mother has milk to nurse her baby. She holds him tight. At night the baby gorilla sleeps with his mother in a nest of branches. He is with her all the time.

The mother sheep is not alone either. She lives with many other mother sheep and their lambs in a flock. The baby lamb follows her mother. She runs along at her side. She starts to nibble at the grass. The oldest mother sheep in the flock gives a signal when it is time to move. She is the leader. The mother sheep follows the others. The baby lamb follows her.
The baby gorilla grows and grows. Now he is three months old.

Now when his mother puts him on the ground, he can crawl away. But not too far. If he does, the mother gorilla pulls him back to her side.

The baby lamb is only a few weeks old.

When she goes too far away, the mother sheep calls "Baaaaa." And the baby lamb runs to her side. If the baby lamb gets lost, she calls "Baaaaa." The mother sheep hears her cry and goes to find her.

The three-month-old baby gorilla starts to ride on his mother's back. As he rides he slaps the leaves. Sometimes he grabs a bit of a plant and stuffs it into his mouth. He has teeth now. He can eat leaves and stems.

The little lamb nurses less and less and eats more and more grass.

The baby gorilla keeps growing. He is six months old now.
He is still getting milk from his mother, but he is eating more and more plants. Already he can climb by himself. He can walk behind his mother. He can swing sideways and upside down.

Now the baby gorilla is a year old.

When his mother rests, he plays with other baby gorillas. Four of them are playing a game. It looks like follow-the-leader that children play. The leader runs along a branch. The others follow. The leader climbs up a vine. The others follow. The leader slides down. The others slide down too. The leader holds on to a vine and stretches way out. Snap! The vine breaks. The leader falls. And down after him come his friends, vine leaves and all. They tumble softly into the leaves below.

The baby lamb plays with other lambs when she is only a few weeks old.

Lambs play follow-the-leader too. They run after each other through the green fields. They jump on the rock. They jump off the rock. If one jumps up and turns around in the air, the others jump up and turn around in the air.

Baby gorillas play another game. It is like our "king of the mountain." One baby gorilla sits on a stump. The others try to push him off. Bang! He gives them a kick. He is still king of the mountain. He stays there until another baby gorilla pushes him off. Then there is a new king.
Lambs play king of the mountain too.

Baby gorillas wrestle. Here one of them has a headlock on another one.

Here is a fight between two baby lambs. They are facing each other. Their heads are low. They charge! Crack! Their heads bang together. Then they move away. Then they do it again. It's just a game!

Sometimes baby gorillas play by themselves. The leaves are their toys. Here is a baby gorilla with a big bunch of leaves in his hand. He bangs them on the ground. He swings them over his head. Sometimes he tears the leaves with his hands and lets them slowly fall around him in a green shower. Sometimes he puts the leaves upside down on his head and just sits under his hat.

But baby lambs always play together.

Every once in a while the baby gorilla goes to sit near his mother.

The baby lamb goes to her mother too, every once in a while. She lies down next to her in the shade of a big tree.
When the mother gorilla moves through the forest the baby gorilla still rides on her back or he may walk behind her.

When the mother sheep gets on her feet and starts to walk away, the little lamb gets up too and follows her mother.

The baby gorilla gets less and less milk from his mother. By the time he is a year and a half old, he eats only stems, roots, and leaves. But he still stays close to his mother when he is not playing.

The little lamb nurses only for a few months. Then she eats only grass.

She is grown-up. When she is a year old, she can have a little lamb of her own.

But the baby gorilla is still growing. He grows and grows. He stays with his mother until he is three years old. Now he makes his own nest to sleep in at night. His mother has had another baby. The new baby is small and weak. It needs its mother. But the young gorilla is big enough now to take care of himself. He takes his place in the gorilla band.

Birds, like other animals, grow too.
These are sparrow's eggs.

These are duck's eggs.

The baby sparrows grow inside the eggs for twelve days.

Now they are pecking at their shells. They will soon be out.

Here is one. Here is another. Here is sparrow number three. And here is number four.

They are Song Sparrows.

The baby ducks grow inside the eggs for almost a month.

Then they begin to peck at their shells.
Now a wet little duckling comes out. Now another comes out. And another. And another. And another and another and another and another.

They are Mallard ducklings.

The little sparrows are in a nest hidden in the grass. They are tiny and helpless.

They have no feathers.

They can't see.

They can't walk.

How different the ducklings are.

They have soft little feathers.

They can see.
As soon as the ducklings are dry, the mother duck clucks quietly. She leaves the nest. The little ducks follow. The mother duck walks into the water. She swims. The ducklings waddle into the water. They swim too.

Here comes the mother sparrow with food. Up go four heads. Four mouths open. The baby sparrow with his head the highest and his beak open the widest gets fed. The mother sparrow flies away. She comes back with more food. Now another baby sparrow gets the food.

Father sparrow brings food too. Both parents are busy all day long catching insects to take back to the nest.

When it is cold, or if the sun is too hot, mother sparrow spreads her wings over the little sparrows.

The ducklings do not have to be fed. They find their own food on the water. They eat any bugs, beetles, or flies on top of the water. The mother duck tips down into the water and eats the plants and seeds she finds there.

At night the mother duck leads her ducklings to the nest. She spreads her wings over them and keeps them warm.

The father duck goes off by himself. The mother duck takes care of the ducklings.
The little sparrows grow bigger. Here they are three days old.

Their feathers are starting to grow. Their eyes are beginning to open. The mother and father sparrow keep bringing food.

The little sparrows grow and grow. Six days old.

The sparrows are about to hop out of the nest. Here goes one. And another. And the next. And the last. They all gone. The sparrows do not go too far. They move around in the bushes near the nest.
During the next few days

they start to fly. Each little sparrow is alone. But each one can hear the mother and father sparrow. And the mother and father sparrow can hear them. When they call "Eep!" their parents can find them in the bushes and give them food.

If danger is near, the mother or father sparrow calls "Tik, tik, tik," and the young sparrows keep very quiet. Then they are hard to find.

But where do the ducklings hide when there is danger? Here comes a turtle. The mother duck cries out. The ducklings clump together behind her. But now the turtle is too close. The mother duck quacks loudly. The ducklings hide in the water plants. The mother duck flaps her wings. The turtle follows, but she keeps ahead. When they are far from the ducklings, the mother duck flies up into the air. She goes back and calls her ducklings. They come to her. They are safe now.

The sparrow's stay in the bushes around the nest until they are seventeen days old.

Then they come out of hiding. The four baby sparrows find each other again. They can fly well now.
The sparrows are twenty-one days old.

They still follow their parents for food, even though they can find their own food now. When father sparrow sings, they pop up beside him. Sometimes they land on top of him!

At one month the sparrows are grown-up.

They feed themselves. They fly around. They call "tsip, tsip" to each other.

The ducklings take longer to grow up. All through the warm days of summer they swim, eat, and rest.

It is two months before they have all the feathers they need to fly. Here is one young duck scooting across the water. His wings are flapping. Another week or so and the whole family of ducklings is flying. The duck family is grown-up now.

The young ducks join other ducks. And one day in the fall, flock after flock takes to the air and flies south.
The young sparrows fly south in the fall too.

Next year the birds will fly back north. The sparrows and ducks that were babies the year before will raise their own families.

The passage-dependent network is shown below. The nodes have arbitrarily been assigned the identifiers N1, N2, etc. Each node has been given a descriptive name. Under each node is a list of its immediate descendents within the passage-dependent network and the passages in the passage tree to which it points.

N1 Animals
   N2, N5

N2 Mammals
   N3, N4

N3 Gorilla
   N9, N14, N17, N22, N29, N32, N34, N39, N43, N56, N58, N69, N92, N94, N96, N98, N105

N4 Lamb
   N10, N15, N18, N23, N30, N33, N35, N40, N59, N63, N93, N95, N97, N99, N101, N106

N5 Birds
   N6, N7

N6 Sparrows
   N11, N20, N24, N36, N49, N50, N53, N64, N73, N77, N79, N82, N85, N102, N107, N110, N113, N116, N119

N7 Ducks
   N12, N19, N25, N37, N51, N54, N55, N71, N74, N76, N80, N83, N86, N103, N108, N111, N114, N117, N120

N8 Age
   N9, N10, N11, N12

N9 Age of Gorilla
   P5, P30, P37, P41, P47, P68, P72

N10 Age of Lamb
   P6, P33, P50, P69, P71

N11 Age of Sparrows
   P16, P81, P113, P116, P119, P122, P125, P132, P135, P138

N12 Age of Ducks
   P17, P86, P141

N13 Story
   N14, N15

N14 Story of Gorilla
   P5

N15 Story of Lamb
   P6
Appendix C

This appendix shows the passage tree and the passage-dependent network of a stemp for the ASCII FORTRAN Supplement, along with a segmentation of the text. The passage tree is shown below. The passages have arbitrarily been assigned the identifiers P1, P2, etc. Names and labels for each passage are listed after these P numbers; labels are enclosed in square brackets.

P1 ASCII Fortran Supplement
P2 [Preliminary Material]
P3 [Title]
P4 [Introduction]
P5 [Table of Contents]
P6 [Body]
P7 FTN Compiler Option Changes
P8 [Title]
P9 E-option
P10 F-option
P11 T-option
P12 X-option
P13 Y-option
P14 Collection of FTN Programs
P15 FTN Walkback and Interactive Postmortem Dump
P16 [Title]
P17 [Introduction]
P18 Occurrences
P19 Walkback and Interactive PMD occur on
P20 Walkback also occurs on
P21 Interactive PMD also occurs on
P22 FTN Walkback
The text segments associated with each terminal passage are shown below. Passages are identified with the numberings given to nodes in the passage tree as well as by page and line references to the text.

The table of contents is treated as a single segment. When a section is not subdivided into several passages, any section heading is included as part of the single passage comprising the section. When the material in one section is segmented into several passages, the section heading is placed into a passage by itself.
This document was prepared for use at MACC as an interim supplement to the Ascii Fortran (FTN) compiler manual. It is intended to bridge the gap between the current level of documentation for FTN released by Univac and the current compiler level. Several extensions, enhancements, and changes to the compiler and its library, are described. No attempt is being made at present by MACC to correct errors in the current level of FTN documentation as this is currently being undertaken by Univac.

TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTN Compiler Option Changes</td>
<td>2</td>
</tr>
<tr>
<td>Collection of FTN Programs</td>
<td>3</td>
</tr>
<tr>
<td>Interactive Postmortem Dump</td>
<td>4</td>
</tr>
<tr>
<td>FTN Walkback Facility</td>
<td>5</td>
</tr>
<tr>
<td>Internal Subroutines</td>
<td>6</td>
</tr>
<tr>
<td>Fortran V Interface Routines</td>
<td>7</td>
</tr>
<tr>
<td>Additional/Enhanced CHECKOUT Commands</td>
<td>8</td>
</tr>
<tr>
<td>FTNR: Ascii Fortran Restart Processor</td>
<td>9</td>
</tr>
<tr>
<td>Abnormal FUNCTION Returns</td>
<td>11</td>
</tr>
<tr>
<td>ENCODE/DECODE Extensions</td>
<td>12</td>
</tr>
<tr>
<td>Cost Warnings to FTN Users</td>
<td>12</td>
</tr>
<tr>
<td>Punch Restriction</td>
<td>13</td>
</tr>
<tr>
<td>Utility Routines</td>
<td>14</td>
</tr>
</tbody>
</table>

E-option. This option, identical to the Ascii Fortran 'COMPILER(U1110=OPT)' directive, invokes a code-reordering algorithm, increasing the execution speed of the object module on U110 hardware. The speed enhancement is not reflected in CAU charges since they are based on storage reference counts -- not execution time, but can improve system throughput on "CPU bound" systems.

F-option. The interactive PMD and walkback features are enabled when the F-option is used (cf. pages 3-6).
**T-option.**

Since this option causes compilation of calls to the I/O library with LNJ's rather than LIU's and MACC's version of FTN uses the Common Bank I/O system, this option is not to be used.

**X-option.**

The X-option causes the compiler to perform an ER ABORT$ if a contingency occurs. If the compilation contains errors, an ER ABORT$ is done after the END FTN message.

**Y-option.**

This option produces both the shortened generated code listing (no octals) as well as the storage map, common blocks, and entry point listings of the D-option.

---

**Collection of FTN Programs**

Any ASCII Fortran program that is compiled without the C-option (CHECKOUT load-and-go) must be explicitly collected with @MAP using the LIB FTN*LIB. directive. If the O-option is used to compile ASCII Fortran programs with D-Banks larger than 65K, then the E-option should be used with @MAP in the collection.

**FTN Walkback and Interactive Postmortem Dump**

Both Walkback and Interactive PMD are available on collected absolutes when the F-option was used to compile symbolics into relocatables.

**Walkback and Interactive PMD occur on:**

- Math Library Error
- Error in the I/O Library
- Illegal Operation (IOPR)
- Guard Mode (IGDM)
# Error Mode

P20  Walkback also occurs on:
Page 3  Line 11

Walkback also occurs on:
  * Execution of the Fortran statement:
    CALL FTHWB

P21  Interactive PMD also occurs on:
Page 3  Line 14

Interactive PMD also occurs on:
  * Execution of the Fortran statement:
    CALL FTHPM
  * Break Keyin (BREAK followed by \@\&X C)

P23  [General Description]
Page 3  Line 18

** FTN WALKBACK **

Walkback gives the absolute address where an error or call to FTHWB occurred as well as ISN's (Internal Statement Numbers) of the current nesting of subroutine and function calls as far back as the main program if all relocatables were compiled using the F-option. Notice that the walkback includes all external subprograms whether compiled in the same or distinct elements. See example on next page.

P24  [Example]
Page 4  Line 1

** FTN WALKBACK EXAMPLE **

** FTN Source Code **

CALL S1
END

SUBROUTINE S1
CALL S2
RETURN
END

SUBROUTINE S2
DIMENSION A(2)
PRINT *,A(100 000)
RETURN
END

** Resulting Execution **

GUARD MODE  ERR-CODE:02
ERROR ADDRESS: 087677  BDI: 400025
THIS ADDRESS IS IN COMMON I/O BANK  C2PS
I/O REFERENCED AT ISN  9  OF  S2
S2 REFERENCED AT ISN  4  OF  S1
S1 REFERENCED AT ISN  1  OF  MAIN PROGRAM

***** ENTER FTN PMD *****
-D !

>>> ELEMENT NAMES <<<
A  /S2  /NAMES

** FORTRAN DUMP ROUTINE **

VARIABLE ADDRESSES...

058402  .00000000  .00000000
-EXIT

***** EXIT FTN PMD *****
USER DID AN ER EABTS
RESN ADDR:047773  BDI:200005
FTN INTERACTIVE POSTMORTEM DUMP

Demand use: (input is solicited with a '-' character)

Commands: (optional fields enclosed in brackets '[]')

D[UMP][,opt] :

  - Dumps all variables in all program units in all elements.

D[UMP][,opt] [var]/[prg]/elt

where

var is:
  - A scalar variable name
  - An array name
  - An array element with constant subscripts
  - Omitted, dumping all variables

prg is:
  - '*' for the main program
  - A subroutine name or function name
  - N for the n-th block data subroutine
  - Omitted for the first program unit in an element

elt is:
  - A relocatable element name

opt is:
  - 'A' for an ASCII alphabetic dump
  - 'O' for an octal dump
  - Omitted for a dump using the type assigned by the compiler

Blanks can appear only before var. Slashes are required.

EXIT

Causes termination of Interactive PMD and a return to the program if a BREAK key in or call to FTNPMD caused the dump.

Batch use:

To cause a symbolic PMD at the end of execution use an F-option on the EXIT card as well. This causes the commands DUMP and EXIT to occur if PMD is initiated.

FTN INTERACTIVE PMD EXAMPLE:

FTN Source code:

```
REAL R(5),A,B,C
DATA R,-3.0,-1.0,0.0,41100.5,
    A = 5.0
    B = 10.
    CALL SUB (R)
    C = 15.
    PRINT *,A,B,C
    STOP
END

SUBROUTINE SUB (X)
    REAL X(5),R,S,T
```
COMMON R,S,T
X(2) = 99.8
CALL FTN
R = X(5)
RETURN
END

Resulting execution:  (user commands in lower case)

***** ENTER FTN *****
dump,a r (4)/'name$*
 R /* /NAME$  
** FORTRAN DUMP ROUTINE ** ADDRESSES 051250 TO 051260
051250 data
-d x/sub/name$
 X /* /NAME$  
** FORTRAN DUMP ROUTINE ** ADDRESSES 051255 TO 051261
051255 .3000000+001 .9900000+002 .11039264-018
051261 .3050000+002
-dump r/sub/name$
 R /* /NAME$  
** FORTRAN DUMP ROUTINE ** ADDRESSES 051252 TO 051252
051252 .5000000+001
-d b/'name$
 B /* /NAME$  
** FORTRAN DUMP ROUTINE ** ADDRESSES 051253 TO 051253
051253 .1000000+002
-exit

***** EXIT FTN *****
30.500000 18.000000 15.000000

Internal subprograms are a newly-implemented feature of Ascll Fortran. The rules for internal subprograms regarding coding, syntax, and name scoping are the same as for Fortran V (cf. Fortran V - Compiler Reference Manual, section 4.8) The addition of this feature means you must be sure to put END cards where you intend them. Since multiple external subprograms can occur within a single element of a program file or a card deck, omitting an END statement can spell the difference between an internal and an external subprogram, resulting in very different executions after compilation. The compiler will signal no more warning than a syntax error prior to the first line in the listing of an external subprogram included in a multiple program unit. This does not occur before an internal subprogram.

EXAMPLE LISTING:

@FNT,IS
FTN 6R1Q2-12/16/76-03:49
1. C ** MAIN PROGRAM ****
00001 2. INTEGER INDEX
00002 3. DO 5 INDEX = 1,10
00003 4. CALL SQUARE
00004 5. 5 PRINT*,INDEX,'! = ',#PACT(INDEX)
00005 6. STOP
00006 7. SUBROUTINE SQUARE
** INTERNAL SQUARE ****
PRINT*,INDEX,***2=*,INDEX*INDEX
* SAME INDEX AS ABOVE LOOP *
RETURN
END

FUNCTION FACT(N)
** EXTERNAL FACT ****
FACT=1
DO 5 INDEX = 2, N
5 FACT = FACT * INDEX
C LOCAL INDEX IN THIS LOOP *
RETURN
END

END FTN 56 IBANK 52 DBANK

Fortran V Interface Routines

There are seven entry points in the FTN library to allow FTN programs to call Fortran V subprograms. The entry points are:

FN5$ to call subroutines
RFN5$ to reference REAL functions
DFN5$ to reference DOUBLE PRECISION functions
CFN5$ to reference COMPLEX functions
DCFN5$ to reference DOUBLE COMPLEX functions
NFN5$ to reference INTEGER functions
LFN5$ to reference LOGICAL functions

P38 [Calling Sequence]  
Page 8 Line 12

Any Fortran V routines must be declared EXTERNAL in the calling Ascil Fortran module. The call and references should resemble:

CALL FN5$ (buffer, subroutine, arg1, ..., argn)
or
...... xFN5$ (buffer, function, arg1, ..., argn)...

where x is C,DC,N,R,L or D and buffer is an array of dimension at least n+4. The same buffer may be used for any call or reference independent of n or the external routine as long as it is large enough. The sequence arg1, ..., argn is the argument list expected by the named subprogram.

P39 [Example]  
Page 8 Line 24

EXAMPLE:
DIMENSION CALL(12),VECTOR(100)  
EXTERNAL ursort,timset,timget
...
COMMENT - use Fortran V, MACC utility routines
CALL FN5$(CALL,timset,0)
CALL FN5$(CALL, ursort, 0, 100, VECTOR, D1, D2, 0, 0)
LAPSE = RFN5$(CALL,timget,0)
...
A warning will be issued for the second CALL on FH5$ by the compiler due to the inconsistency in number of arguments, but this may be ignored.

Additional and Enhanced CHECKOUT Commands

The commands described here are enhancements of existing commands or are newly implemented. Operation of CHECKOUT debugging mode and the rest of the available commands are described in the Fortran (Ascii) PRM. In the command formats given, fields within square brackets ([ ]) are optional.

Call subprogram([(arg1,...,argn)])

This command calls the named FTN subprogram with the specified arguments. This allows you to test only a given subprogram without having to execute the entire FTN program. Arguments may be constants, variables, arrays, labels or entry points.

eg. C:CALL SUBX('Test',3,$99,ARRAYM,$ENTRYP)

DUMP[,opt] [var]/[unit]

The scalar variable, array element, or array specified by var (or all the program unit's variables if omitted) are dumped in subprogram unit or main program if unit = * (or first unit if omitted) and are displayed according to their declared types or in octal format (with opt = O) or ASCII character format (with opt = A). One may not omit both var and unit.

GO [nL]

CHECKOUT debugging mode terminates and execution commences at the next executable statement if the nL field is omitted; otherwise the execution starts at the executable statement with label n.

BREAK, CLEAR, SET, SETBP

Variables and labels in these commands may be of the form loc[/unit] where loc is the variable or label and unit is * for the main program, n for the n-th BLOCK DATA program or the name of a subroutine. If loc is used alone, the default unit is either the first unit or the unit specified in a PROG command.
SETBP[,opt] var[/unit]

This command sets a hardware breakpoint so that CHECKOUT debugging mode is re-entered whenever the designated var is set or referenced. Var may be an array element but not an array, and if it is a character variable, only the first storage location is affected. The 1110's programmable breakpoint register is used, so there may only be one SETBP command active at a time. Debug mode is re-entered at the beginning of the next executable statement after the specified variable has been set (opt = W), referenced (opt = R), or either set or referenced (no option). The breakpoint set in this manner can be cleared either by the CLEAR command or by another use of SETBP.

WALKBACK

This command produces a step-by-step trace of subprogram references that occur during CHECKOUT program execution. The starting point is the subprogram that was executing before debug mode was entered. The end point is the main program.

Command Abbreviations

All debug commands may be abbreviated to the first letter with the following exceptions: SAVE (SA), STEP (ST), CALL (CA), SETBP (SETB).

FTNR: Asci! Fortran Restart Processor

A small separate processor, FTNR, is released by Univac with the Asci! Fortran system. This processor serves to restart previous CHECKOUT debugging sessions. It is called with the source input (SI) field of the $FTNR card containing a program file element which represents one of the SAVES from a previous FTNR session. The processor signs on in a standard manner and goes interactive after reloading the program. Using FTNR can avoid recompilations of programs over several debugging sessions if the first CHECKOUT command after the compilation is SAVE and the file specified to hold the checkpointed executions is still available.
EXAMPLE:

```
#FTH,IScz PROG,SAVEFILE.PROG1
 : [checkout program]

#EOF
 : [data and checkout commands]

C:SAVE 6
 :

#FTHB SAVEFILE.PROG1
FTRN 6R1 T281U1
ENTERING USER PROGRAM : PROG1
C:RESTORE 6
```

All CHECKOUT commands are available in FTHR. The same level of FTRN and FTHR must be used. This is true of the RESTORE command also. The same level of FTRN/FTHR must have done the corresponding SAVE. Note that the file expressed in the relocatable output (RO) field (SI if omitted) on the FTRN checkout processor card is the file used to save the state of the started FTRN program. Any I/O files used by the CHECKOUT program are not saved.

EXAMPLE:

```

ANSWER = 10.0E+3 - SQRT(PHI(BETA,KAPPA,$90))

90 PRINT *, 'PHI negative to SQRT'

END
REAL FUNCTION PHI(A,B,*)

IF(PHI.LT.0.0) RETURN 1
RETURN
END
```

Should function PHI compute a negative value, a branch to statement 90 will avoid the argument out of range to SQRT.
ENCODE / DECODE Extensions

[Errsn] is now an optional last clause in both the ENCODE and DECODE statements of Ascii Fortran. The n must be an executable statement number in the program unit and control will transfer to that executable statement if an error occurs in the execution of the ENCODE or DECODE statement.

Example:

```fortran
ENCODE(80,11,RESULT,LENGTH,ERR=99)A,B,C,...
11 FORMAT(...)
...
99 PRINT *, 'ENCODE Error, FORMAT 11.'
```

Cost Warnings to FTN Users

There are several very useful features of Ascii Fortran that are rather expensive to use on the Univac 1110. Several important examples follow.

Direct Access I/O

```
# Direct Access I/O. When a DEFINE FILE statement is encountered that describes a direct access I/O file different from the file it is attached to, the file will be skeletonized. This means empty records are written to fill the file to its maximum size and can accrue substantial I/O costs.
```
F-option Collection

* F-option Collection. The use of the F-option on Ascii Fortran compilations can result in collected absolutes up to twice the size of absolutes collected from the same compilations without the F-option. Collection cost can increase three to four-fold at the same time.

Many large subroutines

* Many large subroutines. While the Ascii Fortran compiler is capable of compiling many subprograms in a single element, there is a point at which it becomes less expensive to do several small compilations than one large one. At approximately 1500 lines of Fortran code, mass storage files are required to save results due to main memory compiler tables becoming full. For example, it is better to do two 1500 line compilations than one 3000 line compilation.

Punch Restriction

Output files created by Ascii Fortran are marked as Ascii encoded data sets. This includes punch files. At MACC, Ascii SDP files can only be translated to card punch codes on punch symbiont.

Utility Routines

CP2. Punch should be #EDIT'd to Fieldata or #SYM'd to CP2.

N$ARGS(<HPARMS>,<NAME>,<$N>) -- Argument Check

This routine may be used within an external function or subroutine to check that a subprogram was called with the correct number of parameters. N$ARGS may be used as a subroutine or a logical or integer function. <HPARMS> is the number of parameters that should have been used in the call. <NAME> is a character string name of the checked subprogram, and <N> is a statement number labelling the first statement of the checked subprogram. Zero or .FALSE. is returned as the value of the function when an improper number of arguments was used in the call. As either a subprogram or function, an error message is printed on the occurrence of an improper call.
Sample usage:

SUBROUTINE SUBX (A,B,C,*
LOGICAL N$ARGS
1  IF( .NOT.N$ARGS(4,'SUBX',1) )RETURN

The passage-dependent network is shown below. The
nodes have arbitrarily been assigned the identifiers N1, N2,

etc. Most nodes are category nodes. Nodes marked with an

asterisk are co-occurrence nodes. Labels are shown in

brackets. Unlabelled nodes have been given descriptive

names. Under each node is a list of its immediate
descendants within the passage-dependent network and the

passages in the passage tree to which it points. The
table-of-contents entries have been encoded as

cross-references.

N1 [Cross-References]
  N4, N39

N2 [Referenced Passages]
  N7, N9, N11, N13, N15, N17, N19, N21, N23, N25, N27, N29, N31, N35, N37

N3 [Referencing Passage]
  N32, N38

N4 [Internal Reference]
  N5, N33

N5 [Table of Contents Entries]
  N6, N8, N10, N12, N14, N16, N18, N20, N22, N24, N26, N28, N30

N6 References to P7 (FTN Compiler Option Changes)
  N7, N32

N7 Location of P7
  P7

N8 References to P14 (Collection of FTN Programs)
  N9, N32

N9 Location of P14
  P14

N10 References to P25 (Interactive Postmortem Dump)
  N11, N32

N11 Location of P25
  P25

N12 References to P22 (FTN Walkback Facility)
  N13, N32

N13 Location of P22
  P22

N14 References to P31 (Internal Subprograms)
  N15, N32

N15 Location of P31
  P31
References to P35 (Fortran V Interface Routines)

Location of P35

References to P41 (Additional/Enhanced Commands)

Location of P41

References to P52 (FTNR: Ascii Fortran Restart Processor)

Location of P52

References to P57 (Abnormal FUNCTION Returns)

Location of P57

References to P61 (ENCODE/DECODE Extensions)

Location of P61

References to P65 (Cost Warnings to FTN Users)

Location of P65

References to P72 (Punch Restriction)

Location of P72

References to P73 (Utility Routines)

Location of P73

Location of Table-of-Contents

References by Page

References to P15 (FTN Walkback and Interactive Postmortem Dump)

Location of P15

References to P70 (Cost of F-option Collection)

Location of P70

Passages Referenceing P15 and P70

[External Reference]

to Ascii Fortran PRM

supplemented by this document

for specific information

to Fortran V - Compiler Reference Manual, section 4.8

ASCII Fortran
Appendix D

This appendix shows the passage tree and the passage-dependent network of a stemp for the LEXICO Maintenance Guide, along with a segmentation of the text. The passage tree is shown below. The passages have arbitrarily been assigned the identifiers P1, P2, etc. Names and labels for each passage are listed after these P numbers; labels are enclosed in square brackets.

P1 LEXICO Maintenance Guide
P2 [Title Page]
P3 [Body]
P4 1. [Introduction]
P5 2. Command Language Features
P6 [Title]
P7 [Subsections]
P8 2.1 ROUTE
P9 2.2 [Options]
P10 2.3 QFILE
P11 2.4 GLOBAL
P12 2.5 PROJECT
P13 2.6 MESSAGES
P14 2.7 The Resolver
P15 2.8 START
P16 2.9 Resetting the Bad Collection Bit
P17 3. Options
P18 [Title]
P19 [Subsections]
P20 3.1 Parser Options
P21 [Introduction]
P22 [List of Options]
P23  A Option
P24  D Option
P25  E Option
P26  F Option
P27  H Option

P28  I Option
P29  L Option
P30  M Option
P31  N Option
P32  O Option
P33  P Option
P34  Q Option
P35  R Option
P36  S Option
P37  T Option
P38  U Option
P39  W Option
P40  X Option
P41  3.2 Adder and Concordor Options
P42  3.3 LISTTYPES, RESEPLT, LOOKUP
P43  3.4 Listing Rules
P44  3.5 Resolver Options
P45  [Introduction]
P46  Options List
P47  C Option
P48  D Option
P49  F Option
P50  H Option
P51  O Option
P52  S Option
P53  3.6 The Message Processor
P54  3.7 STATISTICS
P55  4. [Files]
P56 [Title]
P57 [Subsections]
P58  4.1 System Defaults
P59  4.2 The Log File
P60  4.3 The Statistics File
P61  5. Supporting Programs
P62 [Title]
P63 [Subsections]
P64  5.1 Changing Array Sizes in the Resolver
P65 [Title]
P66  What to Change
P67  How to Make the Changes
P68  5.2 Statistics
P69 [Title]
P70  Printing Statistics
P71 [Options]
P72  Data Cards
P73  Packet Types
numberings given to nodes in the passage tree as well as by page and line references to the text.

When a section is not subdivided into several passages, any section heading is included as part of the single passage comprising the section. When the material in one section is segmented into several passages, the section heading is placed into a passage by itself. Some information in this text (e.g., P22, P46, P75) is presented as a list of items in a tabular form. In order to show the interaction of concepts identified in the text-based network, each entry in these lists has been treated as a separate passage. In an actual implementation of some application of steeps, tables would probably be treated as a special case so that this material could be left unsegmented.

The text segments associated with each terminal passage are shown below. Passages are identified with the
LEXICO MAINTENANCE GUIDE

Nathan Belles
Lynne A. Price

Computer Sciences Department
University of Wisconsin
February, 1977

1. Introduction

This document partially describes maintenance of
the LEXICO system. It is a reference manual for the
programmer who is familiar with the basic components of the
system and forms only a small portion of the desirable
internal documentation. Included are command language
capabilities, options, debugging features, auxiliary files
and supporting programs.

2. Command Language Features

2.1 ROUTE

The commands

ROUTE r ;
ROUTE r n ;
ROUTE r m - n ;
ROUTE r charst;

may be used to transfer to a route r in QUEENB (after
setting NUM, NUMM and NUMN, or BCHSEQ and IS). These
statements may appear in any block. They have two uses;
they may be used to test a new command that has not yet been
resolved into the grammar and they may be used to request
system functions that have no corresponding command language
statement. Route numbers for these system statements are
all greater than 999. The special command

ROUTE 1000;

causes a display of the system route numbers and the
function of each.
Parser options (which are listed in Section 3.1) may be set on the control statement or with either of the commands

ROUTE 1005;
OPTION olist;

Both commands may be entered in any block. The former causes a display of all available options followed by prompts for the options to be changed. In the latter command, olist is a list of letters and numbers indicating options whose on/off flags are to be reversed. Reserved words appearing within olist must be enclosed in quotes. d and s are reserved throughout the system and t is reserved in EDIT blocks.

2.4 GLOBAL

The command

ROUTE 1006;

may be used in any text-specific block to inspect or change a text header; it may be used to inspect or change the collection header elsewhere. Changes may be made in core to the collection file, or both. The system will prompt for all necessary information.

2.5 PROJECT

The task command

ROUTE 1004 'proj';

may be used to work with a collection defined under a project other than the one to which the current run is being charged. The parameter is a character string—the quotes are necessary.
2.6 MESSAGES

The task command

ROUTE 1003;

starts a job that runs the message processor. The system
prompts for the qualifiers for PARSER, LIB, and UAIDS. It
also asks whether or not a listing should be generated. The
started run does not create the UAIDS file.

2.7 The Resolver

The task command

ROUTE 1002;

causes prompts for the qualifiers for LIB and PARSER and
for desired options and then starts a run to resolve the
grammar. The result is a relocatable element PARSER.GRAMMAR
which takes effect when LEXICO is remapped.

2.8 START

In response to the task command

ROUTE 1001;

the system sets the R option (unless it is already on),
asks if an off-line backup should be created, and then
starts a run.

2.9 Resetting the Bad Collection Bit

The command

ROUTE 1008;

causes the bad collection bit of the current collection to
be cleared (both in core and on the file). It may be
entered in any block but should be used with extreme
caution. If a problem occurs when changes made to basetype
rules may not have been completed on the hash file, it is
preferable to restore the collection and repeat any
intervening processes.
3. Options

3.1 Parser Options

Currently defined options in the parser are

A (6)--display route on each call to QUEENB

D (9)--display images from DYNAMO

E (10)--display runstream images

F (11)--UIOPAK (UOPEN, UREAD, WRITE, UCLOSE) trace

H (13)--UIOPAKH trace (hash file)

I (14)--input buffer and character count from UGIN and UGINA6

L (17)--create master log file (see Section 4.2)
M (18) -- display erroneous uaid packets

N (19) -- do not write log file (see Section 4.2)

O (20) -- UORBIN, UORBOT, ORBpak expansion trace

P (21) -- parser trace

Q (22) -- use developmental files: prompt for qualifier for UAIDS at initiation and for qualifier in processor calls in started runs

R (23) -- modify started runs: allow change of run limits, insertion of card images after the RUN statement and at the end of the runstream, and prompt for qualifier in processor calls

S (24) -- UFD2NF trace

T (25) -- EDIT trace

U (26) -- UIOPAKU trace (UAIDS)

W (28) -- stopword trace dump
3.2 Adder and Concoer Options

The A (6) option causes tracing information to be printed during adding or concording. The X (29) option prevents the text from being stored in the collection on an ADD. The W (28) option suppresses generation of the printable concordance.

3.3 LISTTYPES, RESPELL, LOOKUP and SLIPS

The D (9), F (11), and O (20) options have the same purpose in all ORBfile programs as in the parser. LOOKUP and SLIPS also use the H (13) option. SLIPS uses the C (8) option as a trace on reading the concordance file, and S (24; slips), B (7; base concordance), and T (25; slips file) to indicate output options.
F (11)--print finite state automaton

N (19)--list new grammar (productions sorted by length)

O (20)--list grammar in original form

S (24)--list grammar, vocabulary, and ROUTER array

3.6 The Message Processor

The S (24) option on the message processor causes a source listing.

3.7 STATISTICS

The P (21) option on the statistics program suppresses printing of packets from runs under project 9812. The D (9) option causes only packets from selected dates to be printed.
CREATE block. If this is done, the resulting hash file should be deleted.

4.2 The Log File

A log file, containing most system messages and all user input read by UGIN (or UGINA6), will be catalogued and saved for every run of LEXICO initiated without the N option whenever a master log file exists. A master log file will be created and initialized, but not saved, whenever a run is initiated with the L option on and the N option off, unless there is an existing master log. The master log file, called 5603*LOG, contains its length in words in the first location. All following locations contain names of run logs in two-word packets. If no log has been catalogued, a temporary log may be created on unit 12 by setting option N equal to 0 after initialization.
CLWORZ: 175  maximum number of user keywords  
(e.g., S, SH, SHOW)  
CLWRRZI: 12  maximum length of keywords  

P67  How to Make the Changes  
Page 11  Line 11  

To change any of these values the following steps should be taken  

1) change value in LXCO*LIB.RESOLV  
2) change value in LXCO*LIB.PROXY  
3) #FDPR,FIX LXCO*LIB.PROCS  (@ADD LXCO*LIB.PROXY)  
4) compile LXCO*LIB.RESOLV  
5) #PREP LXCO*LIB. (optionally @PACK LXCO*LIB.)  
6) @MAP LXCO*LIB.RESOLVER,LXCO*ABS.  
7) run resolver  
8) compile LXCO*PARSER.PARSER  
    LXCO*PARSER.PIWITZ  
    LXCO*PARSER.PIXTCH  
    LXCO*PARSER.QUEENB  
9) @MAP LXCO*PARSER.LEXICO,LE*ICO.  

P70  Printing Statistics  
Page 11  Line 26  

To print the statistics file, use  

    @ASG,AX LXCO*STATS.  
    @ASG,T STATS.  
    @COPY LXCO*STATS.STATS.  
    @FREE LXCO*STATS.  
    @LXCO*ABS.STATISTICS,OPTIONS  
    fdate ldte (if D option set)  
    optional data cards  

P71  [Options]  
Page 12  Line 1  

If the D option is set the first data card should contain in (2A6) the first date and last date for which statistics should be printed. If the P option is on, packeta for jobs run under 9812 will not be printed.  

P72  Data Cards.  
Page 12  Line 4  

The optional data cards specify (in free format) the numbers of the desired packet types. If no data cards are present, the entire file will be dumped.
The numbers of currently defined packet types are

1 ADD

2 CONCORD

3 ADDCONCORD

4 RESPPELL

5 LOOKUP

6 SLIPS

7 ORB EXPAND (NO COPY)

8 ORB EXPAND (WITH COPY)

9 TEXT EXPAND
P95  PACK BTR
Page 12 Line 27

20  PACK BTR

P96  PACK TEXTS
Page 12 Line 28

21  PACK TEXTS

P97  PACK CONCORDANCES
Page 12 Line 29

22  PACK CONCORDANCES

P98  CREATE HASH TABLE
Page 12 Line 30

23  CREATE HASH TABLE

P99  EXPAND HASH TABLE
Page 12 Line 31

24  EXPAND HASH TABLE

P100 CLEANUP
Page 12 Line 32

25  CLEANUP

P101 EDIT
Page 12 Line 33

26  EDIT

P102 LEXICO (SIGN-ON)
Page 12 Line 34

27  LEXICO (SIGN-ON)

P103 LEXICO (SIGN-OFF)
Page 12 Line 35

28  LEXICO (SIGN-OFF)

P104 COPY BTR FROM
Page 12 Line 36

29  COPY BTR FROM
5.3 FDUMP

To inspect or change any assigned file, enter
@LXCO*ABS.FDUMP

The system will prompt for the file name, address and number
of words.

5.4 MAIL

A message may be written for any user to receive when
he signs on to the parser. To send such a message, first
verify that the mail file (5603*MAIL) exists and has been
initialized (with @LXCO*ABS.CLEARMAIL). Then enter
@LXCO*ABS.MAIL

The system will prompt for the user number and message.

5.5 Dumping Hash Files

To dump all or part of a hash file in a readable format
use
@LXCO*ABS.DUMPHAASHFILE

where proj is the project number and on is the
collection name. The system will prompt for the file name
(on%). It will then ask if the entire letter table should
be output. If the response is y the letter table is
displayed or printed for every character which has an
undeleted hash table. If there is any other response, the
system will prompt for individual characters for which the
letter table should be produced. These should be entered in
(1X,A1) format, terminated by an E0F. The system will then
ask if all the hash tables should be output. Again a y
response indicates that they should and anything else initiates prompting for individual hash tables in (1X,A1). If there is no undeleted hash table for any requested letter, no output will be produced either for the letter table or for the hash table. Since this program can produce large listings, it should be used on-line with caution.

The passage-dependent network is shown below. The nodes have arbitrarily been assigned the identifiers N1, N2, etc. Most nodes are category nodes. Nodes marked with an asterisk are co-occurrence nodes. Functions are shown in brackets. Unlabelled nodes have been given descriptive names. Under each node is a list of its immediate descendants within the passage-dependent network and the passages in the passage tree to which it points.

Implicit cross-references have been identified from the introduction to various other parts of the manual. The reader may wish to select a few of the more important concepts (e.g., N55, the program called "the Parser"; N138, "Options"; and N88, "Files") and trace their descendants through the network, looking for the intersections of paths from these nodes.
Changes to PARSER
P67

QFILE
N17, N135, N139, N268

GLOBAL
N19, N136, N138, N267

Inspecting Files [Input/Output]
N133, N134, N135, N136, N137

Printing Statistics
P68

Dumping the Hash File
P110

Inspecting Files with QFILE
P10

Inspecting ICON with GLOBAL
P11

Inspecting a file with FDUMP
P105

Changing Files with GLOBAL
P11

Changing Files with QFILE
P10

Changing Files with FDUMP
P105

[Options]

Description of Options
P17

Parser Options
P20

Setting Options
N145

Setting Parser Options
P9

Reserved Words
N147

Parser Options that are Reserved Words
P9

A Option
N168, N190

B Option
N201

C Option
N200, N210

D Option
N37, N169, N195, N207, N210, N219, N220

E Option
N170

F Option
N40, N171, N196, N206, N212

H Option
N57, N172, N198, N199, N208

I Option
N173

L Option
N174, N175

M Option
N176
N188 Adder and Conorder
N189 Adder and Conorder Options
N190 A Option on the Adder and Conorder
N191 X Option on the Adder
N192 W Option on the Conorder
N193 LISTTYPES, LOOKUP, RESPELL, SLIPS, LISTTYPES Reversed or By Frequency
N194 ORBfile Program Options
N195 D Option on ORBfile Programs
N196 F Option on ORBfile Programs
N197 G Option on ORBfile Programs
N198 H Option on LOOKUP
N199 H Option on SLIPS
N200 C Option on SLIPS
N201 B Option on SLIPS
N202 S Option on SLIPS
N203 T Option on SLIPS
N204* Listing Rules
N205 Options on Rule Listing Programs
N206 F Option on Listing Rules
N207 D Option on Listing BTR
N208 H Option on Listing BTR
N209 Resolver Options
N210 C Option on the Resolver
N211 D Option on the Resolver
N212 F Option on the Resolver
N213 N Option on the Resolver
N214 O Option on the Resolver
N215 S Option on the Resolver
N216 S Option on the Message Processor
N217 STATISTICS Options