SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 17(1), 5–16 (JANUARY 1987)

Formatting Texts Accessed Randomly

JOHN B. SMITH AND STEPHEN F. WEISS
Department of Computer Science, The University of North Carolina, Chapel Hill, North Carolina 27514, U.S.A.

SUMMARY

Full-text systems that access text randomly cannot normally determine the format operations in effect for a given target location. The problem can be solved by viewing the format marks as the non-terminals in a format grammar. A formatted text can then be parsed using the grammar to build a data structure that serves both as a parse tree and as a search tree. While processing a retrieved segment, a full-text system can follow the search tree from root to leaf, collecting the format marks encountered at each node to derive the sequence of commands active for that segment. The approach also supports the notion of a 'well formatted' document and provides a means for verifying the well-formedness of a given text. To illustrate the approach, a sample set of format marks and a sample grammar are given suitable for formatting and parsing the article as a sample text.

KEY WORDS Text formatting Full-text retrieval Format grammar

INTRODUCTION

Storing, retrieving and displaying text are increasingly important computing activities. Commercial full-text databases now range from the research literature for chemistry[1] to clippings from the popular press,[2] from legal codes[3] to literary works.[4] Because of the novelty of this new resource, users have accepted relatively primitive forms of textual output. Some services provide data in only upper case; others offer output that includes upper and lower case and paragraph indentation. But no full-text system makes effective use of more sophisticated low-cost output devices, such as laser printers and graphics terminals.

One could argue that enhanced formatting of data extracted from a full-text database is not essential—that the new service is so valuable that the user should be glad to have *any* form of output. However, full, accurate formatting is not *just* aesthetic. Consider the following hypothetical situation. U. S. Congressmen frequently read into the *Congressional Record* comments made by others. Although they often quote those who agree with their position, they sometimes quote those who disagree in order to rebut or ridicule that person or that point of view. A full-text search of the *Congressional Record* for passages containing certain combinations of words could locate a passage quoted by a Congressman that represented a position opposite to his own. If the display did not signal through formatting that the extracted portion was a quotation, instead of the Congressman's own words, the user could be badly misled. Format information can, thus, contribute to the *substance* of a text as well as to its appearance. However,

0038–0644/87/010005–12$06.00

Received 26 December 1985
Revised 1 April and 6 July 1986

storing and using format information for full-text systems that employ random access methods presents several problems.

At the time of display, format information is manifest in the pixel image of the text or in some other analogue form. One could store the actual pixel image of the original text in the database, but the volume of data required makes this option impractical as well as undesirable from the standpoint of search. Normally, format information is stored in the form of commands interspersed through the character stream that represents the text. Those commands, in turn, activate various functions in a formatting program that operate on the data or the display device. That is, they may activate a shift to an alternative fount, such as italics, to mark titles or they may activate a shift to the right to mark long quotations.

Other problems arise from the way in which format information is used. When a text is processed *sequentially*, as presumed by virtually all formatting systems, the system 'knows' that it has processed or sent to the output device all formatting commands active for the current segment of text. However, when a text is processed *randomly*, using some form of indexed or inverted file structure, the system would not process the entire text sequentially but, instead, would jump directly to the passage identified and begin processing the text from that point. Thus, it does not 'know' what format commands may be in effect at that point (e.g. in the middle of a long quotation, an italicized passage, or a heading). The system could be instructed to 'back-up' to some predefined check-point, such as the beginning of a section, across which no format command would be permitted to span. But this approach could result in long lists of format marks stored multiple times, as well as arbitrarily long scans of the intervening text between the section beginning and the target point. Thus, check-pointing is restrictive and unpredictable in terms of performance for random access systems.

For full-text systems that support random access, a more efficient and more predictable approach is needed. We are currently developing a system, called MICROARRAS, to provide high-performance search, retrieval and analysis of textual data, and to support sophisticated output devices. The approach is to view the set of format commands as the symbols in a format grammar. The string of format commands can then be parsed using the grammar to build a data structure that serves both as a parse tree and as a search tree. While processing a retrieved text segment, the system follows the rather shallow search tree and pipes out the format commands encountered at each node to accumulate the format commands active for that segment. Below, we describe each of these steps in more detail. To illustrate our approach, we use this article as a sample text and show the application of these methods to it.

FORMAT COMMANDS

For practical as well as theoretical reasons, individual format commands identify generic classes of information in a document. That is, a command that identifies a long quotation signals that fact, not the indentation or other formatting convention by which the quotation is marked on the printed or displayed page. Thus, the set of format commands can be viewed as the architectural principles that give physical form to the textual substance. This perspective has been voiced most strongly in IBM's General Markup Language in its concept of *document architecture*,[5] but a similar view is also found in Scribe,[6] Microsoft's Word[7] and other more recent formatting systems.[8] Efforts

Table I

Sections		
\body-b	\body-e	body of text begin, end
\section-b	\section-e	section begin, end
\para		paragraph
\header-b	\header-e	section header begin, end
\backmatter-b	\backmatter-e	backmatter begin, end
Footnotes		
\footnote-r		reference to footnote
\footnote-b	\footnote-e	footnote begin, end
Figures		
\figure-b	\figure-e	Figure begin, end
\figure-body-b	\figure-body-e	Figure body begin, end
\figure-caption-b	\figure-caption-b	Figure caption begin, end
List		
\list-num-e	\list-num-b	numbered list begin, end
\list-bul-b	\list-bul-e	bulleted list begin, end
\list-b	\list-e	list begin, end
\item-l		item left
\item-r		item right
\item		item
Quotes		
\quote-long-b	\quote-long-e	long quote begin, end
\quote-short-b	\quote-short-e	short quote begin, end
Production rules (Special-purpose)		
\production-b	\production-e	production begin, end
\production-l		left component
\production-m		middle component
\production-r		right component
Title page		
\titlepage-b	\titlepage-e	title page begin, end
\title-b	\title-e	title begin, end
\author-b	\author-e	author begin, end
\address-b	\address-e	address begin, end
\date-b	\date-e	date begin, end
Emphasis		
\emphasis-b	\emphasis-e	\emphasis begin, end

are also under way to develop a standard set of components for specifying document structure and format.[9]

Table I gives the set of high-level generic commands necessary to format this paper plus a few extras. In most cases, commands come in pairs: the first defines the beginning of the domain of the operation; the second defines its end. Commands are signalled by a reserved symbol—in this case the backslash (\).

From one perspective, these commands are simply macro names and could be implemented that way using a number of different formatting systems. However, from a different perspective no symbol says anything directly about physical appearance. Each simply identifies, within the text sequence, a category of information or the end of that category.

One restriction is that format domains may be nested but may not overlap. The result, then, is a hierarchy of format functions and domains. Although this restriction does not rule out automatic section numbering, it does preclude derived references to remote positions in the text, such as cross-references to another section or a derived index. For example, one can say 'see Format Grammar' but not 'see Section 3: Format Grammar' where the '3' is derived by the system. However, full-text systems with random access can provide equivalent function through search and retrieval rather than through explicit references as expected in printed documents.

FORMAT GRAMMAR

A context-free grammar specifies the set of well-formed formatted texts and formally captures the hierarchical structure imposed by the format operations. Format operations are the non-terminals; word tokens are the terminals. In practice, the parser ignores the text words except to note the position of a format mark within the numerical sequence of word tokens. The generic word marker w appears in the grammar in place of actual words.

This grammar differs from a traditional context-free grammar in that the right-hand sides of the productions may contain regular expressions made up of terminals, non-terminals, and special operators:

- $[x]$ Material inside the brackets is optional
- $x|y$ Choice operator: $x|y$ means either x or y. The | operator has lower precedence than concatenation.
- x^* Kleene star: operand may appear 0 or more times.
- x^{*2} Modified Kleene star: operand may appear 0, 2 or more times.
- x^+ Shorthand for xx^*: one or more occurrence of x
- x^{+2} Shorthand for xxx^*: two or more occurrences of x
- () Parentheses used to define grouping

Although strictly speaking these modified productions are not context-free, they are actually just a notational shorthand for a much larger set of context-free productions that could have been specified. Additionally, the modified production rules allow the grammar to produce a parse tree whose structure more accurately reflects the true structure of a document.

Following is a format grammar, using the format commands listed above, adequate to parse this article.

0. root→\text-b \text-e
1. \text-b→ [\titlepage-b \titlepage-e] \body-b \body-e[\backmatter-b \backmatter-e]
2. \titlepage-b →\title-b \title-e
   (\author-b \author-e)*
   (\address-b \address-e)*
   [\date-b \date-e]

3. \title-b→t
4. \author-b→t
5. \address-b→t
6. \date-b→t
7. \body-b→\para*(\section-b \section-e)*2
8. \para→t
9. \section-b→[\header-b \header-e]\para*(\section-b \section-e)*2
10. \header-b→t
11. t→(w|
    \emphasis-b \emphasis-e|
    \quote-short-b \quote-short-e|
    \quote-long-b \quote-long-e|
    \list-b \list-e|
    \list-num-b \list-num-e|
    \list-bul-b \list-bul-e|
    \footnote-b \footnote-e|
    \footnote-r|
    \figure-b \figure-e|
    \production-b \production-e
)*
12. \emphasis-b→t
13. \quote-short-b→t
14. \quote-long-b→t \para*
15. \list-b→\item+2
16. \list-num-b→\item+2
17. \list-bul-b→\item+2
18. \item→t|\production-b \production-e|\item-l\item-r
19. \item-l→t|\production-b \production-e
20. \item-r→t|\production-b production-e
21. \footnote-r→literal
22. \footnote-b→t \para*
23. \figure-b→\figure-body-b \figure-body-e \figure-caption-b \figure-caption-e
24. \figure-body-b→literal
25. \figure-caption-b→t
27. \production-b→\production-l \production-m \production-r
28. \production-l→w+
29. \production-m→ "→"
30. \production-r→t
31. \backmatter-b→t
32. {any operation}-e→*e* (each scope terminating operator is replaced by the empty string)

The non-terminal t is not associated with any format operator, but is instead a shorthand for the right-hand-side of production 11. In the actual parse tree, t's are eliminated; the children of each t are lifted and become the children of t's parent node.

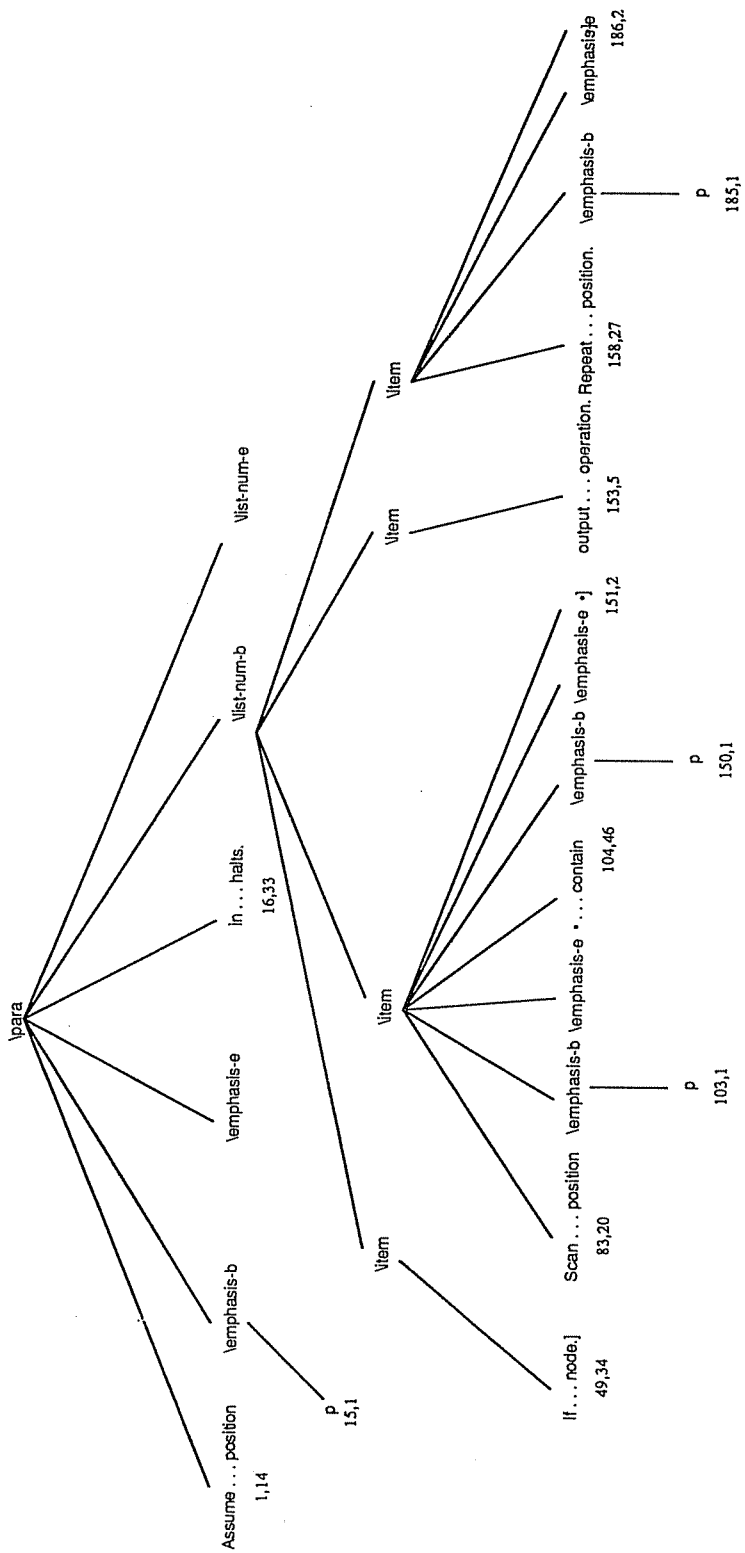

Figure 1. Parse of a paragraph with embedded emphasis and numbered list

THE PARSE/SEARCH TREE

Figure 1 shows the parse tree associated with the second paragraph of the next section (beginning 'Assume that we are searching...'). Contiguous strings of words are indicated using ellipses. The parse tree serves four distinct and useful roles. First, the tree indicates the structure and well-formedness of the text with respect to the format operations. Secondly, a left-to-right scan of the leaves of the tree yields the text without format operations. Thirdly, a pre-order traversal of the tree yields the complete text with format operations in place. And finally, with the addition of some search information at each node, a top to bottom scan of a path in the tree from the root to an arbitrary word w recovers all the formatting operations that apply to w. Since the height of the parse tree is typically proportional to the logarithm of its number of leaves, recovering the formatting environment in this way is far faster than a sequential scan from the beginning of the text.

Of the four capabilities, only the first and fourth are important. The text file is a far more efficient source of running text (with or without format operations), and we assume the availability of such a file. (In our system, MICROARRAS, this file consists of pointers to a lexicon rather than the text stored in character form). Thus, the parse tree is used only for determining well-formedness and for recovering the formatting environment for an arbitrary token in the text. For these applications there is no need to actually store the word tokens in the parse tree. Instead, individual word tokens are represented by pointers into the sequential text file. Strings of contiguous tokens (not containing any format operations) are represented by pairs of integers: a pointer to the beginning of the string in the sequential text file and the length of the string. This provides for a very compact representation of text strings and greatly reduces the size of the parse tree. These integer pairs are shown in Figure 1.

Figure 2 shows the sample text, ten tokens per line.

Figure 3 shows the sample parse tree into which search information has been added. Two numbers are associated with each node. The first is the linear position in the text of the first token within the domain of the operation associated with that node. The second is the number of tokens within the domain. Empty domains, for operations such as \emphasis-e, have zero length. Thus, in the example, the domain of \para is from token 1 to token 187; the domain of the first \emphasis-b is token 15, and \emphasis-e has an empty domain.

Since the parse/search tree is not used for text reconstruction, it need not actually contain contiguous blocks of text nor operations with empty domains. Figure 4 shows the tree from previous Figures as it is actually stored.

SEARCH ALGORITHM

Recovering the format environment for a particular token in the text requires a top to bottom scan of the tree. In the algorithm, the term *current node* refers to the node currently being examined; the *current operation* is the format operation associated with the current node.

Assume that we are searching for the format environment for the token at position p in the text. Initially, the current node is the root of the search tree. We search by repeating steps 1, 2, and 3 until the algorithm halts.

1. If the current node has no children, halt. If the current node has children, then

Assume that we are searching for the format environment for
the token at position p in the text . Initially
, the current node is the root of the search
tree . We search by repeating steps 1 , 2
, and 3 until the algorithm halts . If the
current node has no children , halt . If the
current node has children , then descend to its eldest
child . [This child now becomes the current node
.] Scan the current node and its siblings ,
left to right , looking for an operation whose domain
includes position p . If such a node is found
, make it the current node ; if not ,
stop . [Operations have been defined so that domains
can be nested but cannot overlap ; therefore the domain
of at most one of the siblings will contain p
.] Output the current operation . Repeat the process
. [When the algorithm stops, the string of
operations sent to the output comprises the format environment for
the word at position p .]

Figure 2. Sample paragraph, ten tokens per line

descend to its eldest child. [This child now becomes the current node.]
2. Scan the current node and its siblings, left to right, looking for an operation whose domain includes position p. If such a node is found, make it the current node; if not, stop. [Operations have been defined so that domains can be nested but cannot overlap; therefore the domain of at most one of the siblings will contain p.]
3. Output the current operation.
4. Repeat the process. [When the algorithm stops, the string of operations sent to the output comprises the format environment for the word at position p.]

If, for example, we search Figure 4 for the formatting environment of the token at position 103, the search would visit six nodes of the tree: the root, the two nodes at the next level, the first two children of \list-num-b and the first child of the second item. The operations encountered along the direct path from the root to token 103 comprise the formatting operations that apply to this token. Specifically, the token at position 103 is emphasized within the second item of a numbered list within a paragraph.

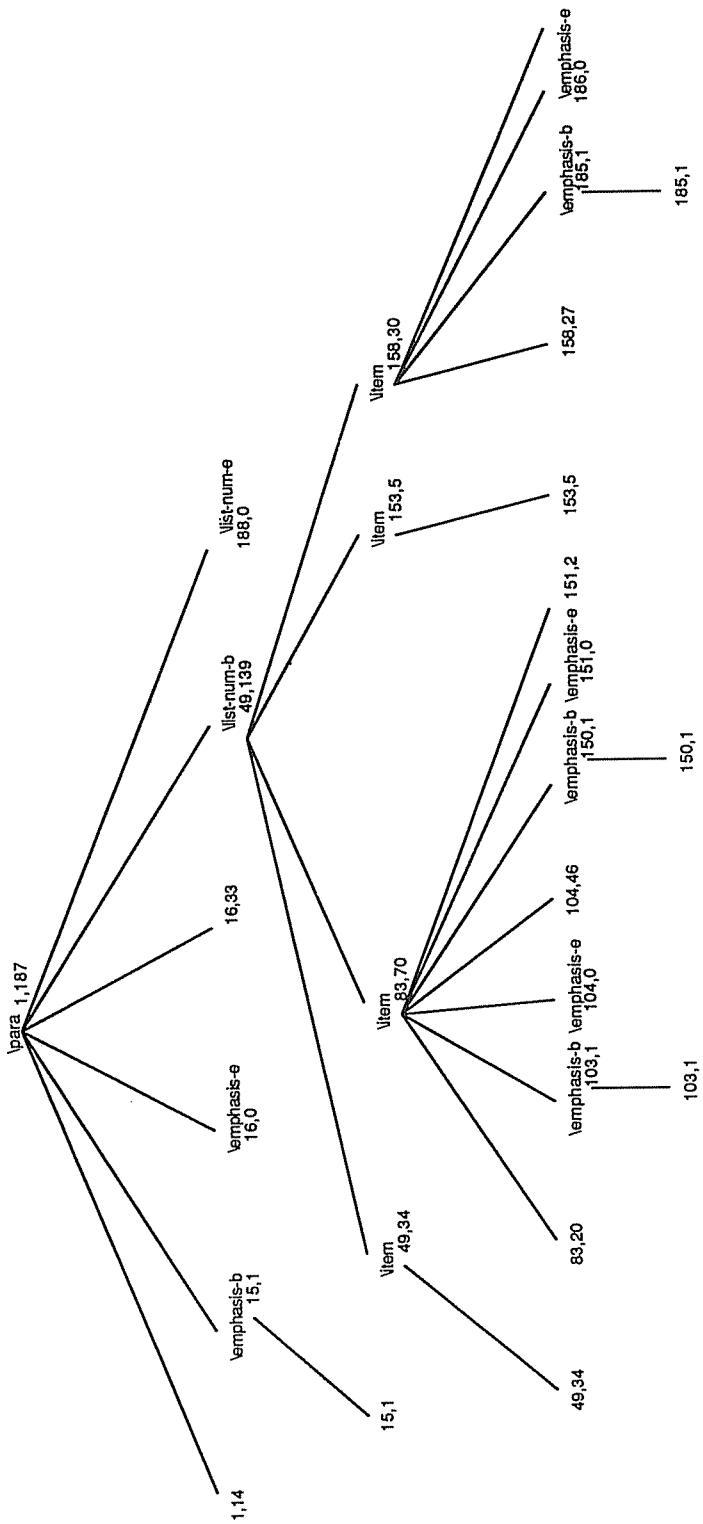

Figure 3. Parse of a paragraph with search information added

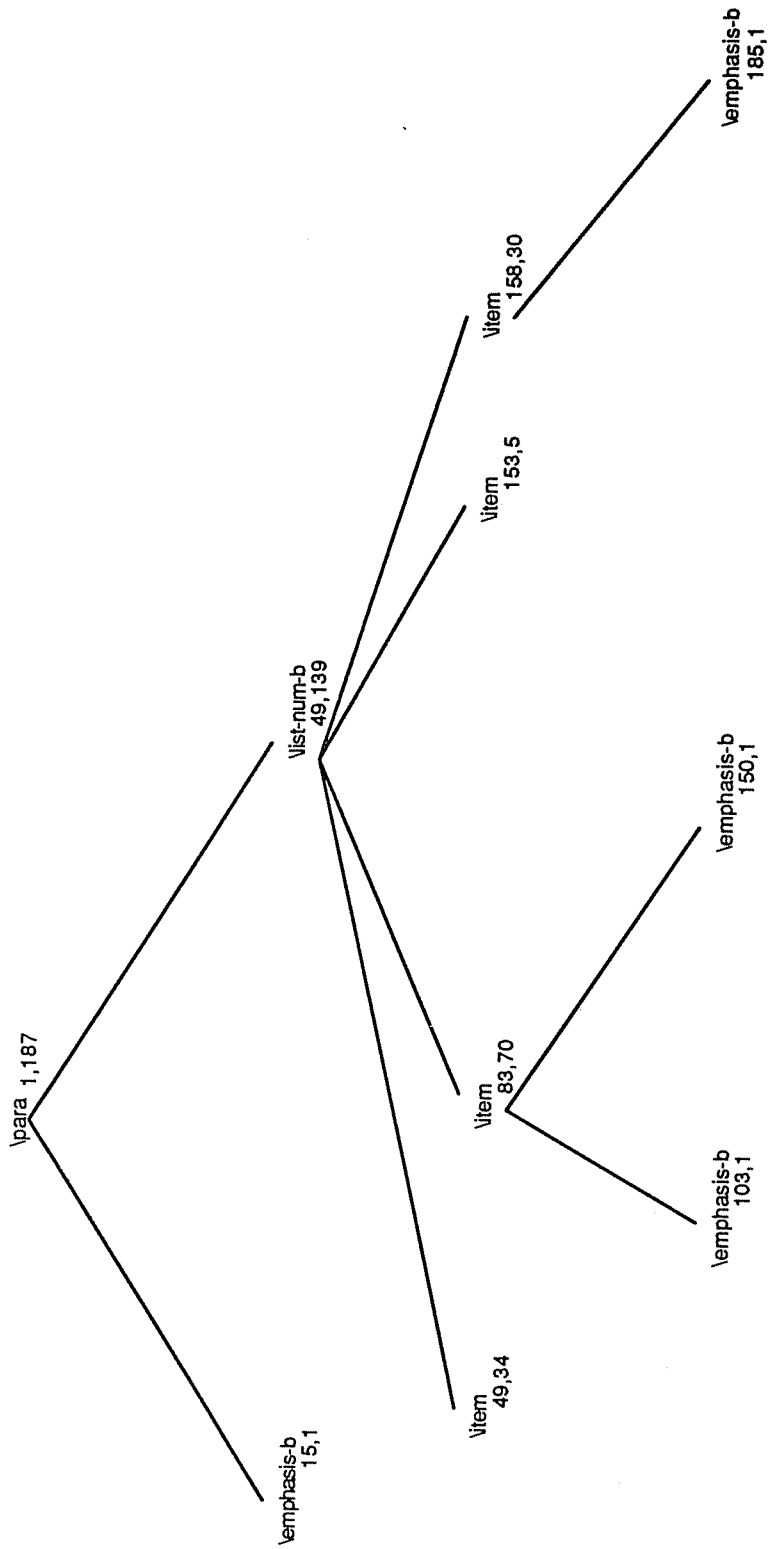

Figure 4. Parse tree as it is actually stored.

THE PARSE ALGORITHM

The parsing algorithm constructs the appropriate parse/search tree from a well-formed sequential string of tokens (words and format operators). For non-well-formed strings, it provides appropriate diagnostics. This process differs from the general context-free parsing problem in that the input string contains both terminals (words) and non-terminals (format operators) rather than just terminals. The input is in essence the linearized pre-order traversal of the parse tree; the parser's job is, thus, to reconstruct that tree.

A standard LL(1) parsing algorithm,[10] can parse a text and construct the parse/search tree in a single left-to-right scan with parse time proportional to text length. The parser can also generate error messages specific to the particular error encountered. The parser is described in more detail in Reference 11.

CONCLUSION

To get a sense of the efficiency of this approach, consider the format tree for this document. It contains 449 nodes, approximately one node for each operation used to format the paper. However, this is not really representative of most texts since 381 nodes come from just two paragraphs containing long lists. The remainder of the document, more typical of conventional texts, requires only 88 nodes. The maximum path length from the root of the tree to a leaf is 9. And although the number of children of a node is as high as 32 in one case (in the list of 32 productions), the average number of children per interior node is only 2·5. This means that recovering the format environment for a particular word requires, on the average, looking at fewer than 20 nodes. This represents at least two orders of magnitude improvement over a sequential scan of the text.

This paper is roughly equivalent to one chapter of a book. Each order of magnitude increase in text size (for example, from chapter to book, and from book to collection of books), should increase the format tree by one level. Since the points of major complexity in the tree tend to be at the lowest levels (for example, in a paragraph containing a complex list), the average number of children per node remains about the same even for large text collections. Hence, the format tree allows extremely fast recovery of the format environment for any point, even in a very large collection of texts.

REFERENCES

1. *User's Guide: American Chemical Society Experimental Full-text Primary Journal Database,* American Chemical Society, Columbus, Ohio, 1981.
2. *The Information Bank II: BRS/SEARCH Protocol User Guide,* The New York Times, New York, 1981.
3. A. R. Menanteaux, 'A user's companion to Westlow and Lexis', *Legal Reference Services Quarterly,* **2**, (2), 19–23 (1982).
4. R. Morrissey and C. Del Vigna, 'A large natural language data base: American and French research on the treasury of the French language', *Educom,* **18**, (1), 10–13 (1983).
5. *Document Composition Facility Generalized Markup Language: Starter Set Reference,* IBM Corporation, General Products Division, Tucson, Arizona, #SH20–9187–0, 1980.
6. B. K. Reid, *Scribe: A Document Specification Language and Its Compiler,* Carnegie-Mellon University Tech. Rep. CMU–CS–81–100, Pittsburgh, PA, 1980.
7. *Microsoft Word,* Microsoft Corporation, Bellevue, WA, 1985.

8. R. Futura, J. Schofield and A. Shaw, 'Document formatting systems', *Computing Surveys,* **14**, (3), 417–472 (1982).
9. *Standard Generalized Markup Language Manual*, Graphics Communication Association, Arlington, VA, 1980.
10. A. V. Aho, R. Sethi and J. D. Ullmann, *Compilers, Principles, Techniques and Tools,* Addison Wesley, Reading MA, 1986.
11. J. B. Smith and S. F. Weiss, *Formatting Texts Accessed Randomly,* The Univesity of North Carolina at Chapel Hill, Computer Science Tech. Rep. 85–031, Chapel Hill, NC, 1985.