An Algebra for Structured Text Search and A Framework for its Implementation

Charles L. A. Clarke G. V. Cormack F. J. Burkowski

Department of Computer Science*
University of Waterloo, Waterloo, Canada

Technical Report CS-94-30
August 24, 1994
Revised December 15, 1994

Abstract

A query algebra is presented that expresses searches on structured text. In addition to traditional full-text boolean queries that search a pre-defined collection of documents, the algebra permits queries that harness document structure. The algebra manipulates arbitrary intervals of text, which are recognized in the text from implicit or explicit markup. The algebra has seven operators, which combine intervals to yield new ones: *containing*, *not containing*, *contained in*, *not contained in*, *one of*, *both of*, *followed by*. The ultimate result of a query is the set of intervals that satisfy it.

An implementation framework is given based on four primitive access functions. Each access function finds the solution to a query nearest to a given position in the database. Recursive definitions for the seven operators are given in terms of these access functions. Search time is at worst proportional to the time required to evaluate the access functions for occurrences of the elementary terms in a query.

*Email concerning this technical report should be sent to `claclark@plg.uwaterloo.ca`. A version of this report will appear in *The Computer Journal*, 1995.

1. INTRODUCTION

A text database organizes a collection of documents to facilitate searching. A simple text database might represent each document as a sequence of words. Each word in each document would be indexed, with the exception of a *stoplist* of a few common words such as "the" and "of". Queries would be expressed using boolean operators to select documents on the basis of the words that they do or do not contain; a concatenation operator would permit searching for phrases.

A simple text database of this form is suitable for small applications. Large, extensible text databases require more sophisticated indexing and search capabilities. A text database should capture document structure; its query language should permit efficient searches to be expressed in terms of this structure. We address three issues concerning databases for structured text:

1. *Capturing document structure.* It should be possible to refer to the structural elements of a document when formulating queries. This requirement necessitates some form of indexing of the structural elements of the documents in the database. We present a simple scheme for indexing the structural elements of documents. The scheme does not depend on a specific format for marking structural elements, permitting documents in a variety of formats to be stored in the same text database and to be searched as a group. Despite its simplicity the scheme is at least as powerful as existing schemes.

2. *A query algebra for structured text search.* We present a query algebra that allows the expression of a wide variety of searches on structured text. The algebra is based on a single data type: the *generalized concordance list* or *GC-List.* The operators in the algebra use GC-lists for both their operands and results; each word or other term indexed by the database is a GC-list.

3. *A framework for efficient implementation of structured text search.* We present a stream implementation based on four access functions defined on GC-lists. For each operator in our algebra we show how the four access functions for its result are implemented in terms of the access functions for its operands. The access functions for index terms may be implemented using standard data structures for inverted lists. Structural elements are indexed exactly as words; a rich structure imposes no special overhead in the implementation.

Background and related work

Most commercial text database systems provide an extended boolean algebra for formulating queries. Salton and McGill [28, chapter 2] review a number of such systems. These systems provide boolean operators — AND, OR and NOT — that operate over sets of documents. The AND operator intersects two document sets; the OR operator combines two document sets. The NOT operator usually implements set difference, taking the complement of a document set with respect to a second set. Words act as elementary terms, each representing the set of documents containing that word. For example, the query

"Birnam" AND "Dunsinane"

would evaluate to the set of documents that contain both the words "Birnam" and "Dunsinane". Various extensions are incorporated into the basic algebra: Word truncation operators select documents containing a word beginning with a specified prefix. Proximity operators select documents on the basis of word adjacency, word concatenation or similar criteria.

Such systems provide limited support for document structure. Generally, a document is divided into several predefined fields, typically title, author, date, abstract and body. Queries may then refer to these fields. For example, a query might specify that the body of a selected document should contain the word "Birnam" and that the author field should contain "Shakespeare". Large blocks of text are often divided into sentences, paragraphs or other predefined units. Documents may then be selected on the basis of words appearing in the same sentence or paragraph. Unfortunately, these techniques for dealing with document structure are excessively rigid. Document structure that cannot be mapped into predefined fields or textual units is lost and cannot be referenced in a query.

Many proposals for dealing with document structure have been made [22]. Generally, these proposals view document structure as hierarchical. Documents are assumed to be structured according to a fixed document description or schema, and queries are formulated in terms of this schema. Often, the schema is assumed to take the form of an SGML *document type definition* (DTD) or equivalent [5, 18]. The result of a query is a set of documents or document components that satisfy a specified search predicate. Bertino et al. [2], Chritophides et al. [8], Macleod [23], and Sacks-Davis et al. [26] all follow this approach. Following similar ideas, Güting et al [14], the draft Structured Fulltext Query Language (SFQL) standard [1], and Blake et al. [3] extend the relational model in various ways to support hierarchically structured documents.

Gonnet and Tompa [13] describe an algebra of operations for manipulating a text database that has been parsed according to a context-free grammar. Colby and Van Gucht [10] build on the grammar-based model of Gonnet and Tompa, providing grammar templates for a number of traditional data models and extending the grammar-based model to support hypertext. Gyssens et al. [15] take a similar approach, presenting two languages for transforming parsed text and showing the languages to be equivalent. The grammar-based model has been proven in practice for restructuring a large body of highly-structured text [4], but has limited use for general searching.

Burkowski [7] proposes a general query algebra that exploits containment relationships between levels in a pre-defined document structure hierarchy. The algebra operates over a uniform data type, called a *concordance list*, which is a sorted set of non-overlapping substrings of a text database.

PAT is a system for free search of structured text developed at the University of Waterloo for use with the New Oxford English Dictionary [11, 12, 27]. In the PAT system, text is not assumed to be structured according to a particular schema. Instead, tags delimiting document structure are indexed as if they are words. A variety of search operators are provided. The result of query is either a set of *match points* — a set of character positions in the text — or a set of *regions* — a set of non-overlapping substrings of the text. The PAT algebra is unique in not requiring text in the database to adhere to a hierarchical schema.

The work described in the present paper improves substantially on both Burkowski's containment algebra and on the PAT algebra. Our algebra is more expressive than either. Our text model provides more flexibility in indexing document structure. The seven operators of our algebra operate uniformly over GC-lists, avoiding semantic inconsistencies present in both. Our algebra is given a formal declarative semantics using standard set notation. A set of equations provide an operational definition of the algebra equivalent to the declarative definition. Applying the equations directly, the algebra may be implemented using standard inverted-list data structures. This is contrast to PAT, which is implemented using special-purpose data structures.

Organization of the paper

The remainder of this paper addresses the three issues listed earlier. In the next section we discuss the issue of document structure and present our model for capturing document structure. The third section describes the query algebra and gives a number of examples of its use. The fourth section details an implementation framework for the query algebra. The concluding sections discuss our work in the context of existing approaches, summarize our results, and outline future research. A useful generalization of the algebra is included as an appendix.

2. STRUCTURED TEXT

Text has natural structure. A document may divide into chapters, pages, sentences, paragraphs, sections, subsections, books, volumes, issues, lines, verses or stanzas. A document may include a title, a preface, an abstract, an epilogue, quotes, references, emphasised passages, digressions and notes. Characteristics of documents vary greatly. A document may have an identified author, or it may be anonymous; it may be precisely dated, or it may be undated; it may be written in Russian using cyrillic characters; it may be written in Japanese using Kanji; it may be part of a larger work; it may stand alone. Each document is structured differently and the structure may vary even within a document.

A fixed schema cannot be assumed.

A text database should permit queries to be expressed in terms of the natural structure of the documents stored within it. Suppose we are interested in prophesies by supernatural creatures. We might wish to make the following queries of a text database that included the *Works* of Shakespeare:

1. Find plays that contain "Birnam" followed by "Dunsinane".

2. Find fragments of text that contain "Birnam" and "Dunsinane".

3. Find the pages on which the word "Birnam" is spoken by a witch.

4. Find speeches that contain "toil" or "trouble" in the first line, and do not contain "burn" or "bubble" in the second line.

5. Find a speech by an apparition that contains "fife" and that appears in a scene along with the line "Something wicked this way comes".

Not only do these examples use document structure to express the query, but the required result — be it play, page, speech, line or merely fragment of text — is also specified in terms of this structure.

A system for capturing document structure should be flexible enough to accommodate the variations in structure that occur naturally. Unfortunately, this requirement is at odds with attempts to impose a fixed schema on the database. It should be possible to index all structure in a document thought to be important at the time that the document is added to the database; it should be possible to add further structural indexing at a later time. Furthermore, when a structural element is irrelevant to the document at hand there should be no artificial requirement to index that structural element — it should not be necessary to break a poem into paragraphs.

A structural hierarchy cannot be assumed.

Several researchers have used hierarchical relationships to describe document structure [7, 13, 14, 15]. However, document structure is not always strictly hierarchical — paragraphs stretch across pages, sentences stretch across lines. Nonetheless, containment of one structural element within another is often significant to a document's structure — sentences are usually wholly contained within a paragraph, lines are usually wholly contained on a page. It might be argued that with the increasing availability of documents in electronic form, structural elements such as pages and lines are irrelevant and are merely artifacts of an older technology. This is not the case. For example, page numbers are essential in citing U.S. federal court decisions in U.S. federal courts. Recently ownership of page numbers for citation purposes has been enforced by the major publisher of U.S. legal decisions [30]. Databases not licensed by this publisher cannot index and report citations using these copyrighted page numbers, making an unlicensed database effectively worthless. While this is an extreme example, pages and lines cannot be ignored while the primary form of text remains the printed page.

Indexing structure is not indexing markup.

When documents are stored and manipulated electronically, document structure is specified by some form of *markup*: a tagging scheme used to delimit the beginning and end of various structural elements. A markup tag often takes the form of a special character sequence embedded in the text. Standard document formats — including SGML [5, 18], ODA [19], TEX [21] and **`troff`** [25] — all use embedded tagging schemes to delimit structural elements.

The presence of markup tags in these document formats suggests that indexing the tags might be an effective approach to capturing document structure. Several problems exist with this approach. Different document formats use different syntax for tags. Despite these differences it is often desirable that delimiters for equivalent structural elements be indexed together. For example, all paragraph boundaries should be indexed together regardless of the particular paragraph delimiters used by the various document formats. Some document formats do not explicitly tag certain structural elements. It still must be possible to index these structural elements. At the same time it should remain possible to index tags specific to a document format. A particular word processor format might not clearly delimit paragraph boundaries; it might be necessary to use a heuristic to identify these boundaries. In this case, indexing the document-specific tags in addition to paragraph boundaries would permit queries on the actual structure of the document as well as the inferred structure of the document. In some circumstances this inferred structure might not accurately reflect the true structure.

A metric is necessary to specify the proximity of elements in the text. It is desirable to view the text as a sequence of words (or other basic textual units). Tags should not be treated as words for proximity purposes. The distance between words might otherwise depend on variations in tagging schemes. For example, two words that are visually adjacent in the printed form of a document should be indexed as adjacent regardless of the presence or absence of a tag that changes the font from one word to the next. Our solution is to assign integer positions to words and to permit tags to take on rational values. Tags may then be indexed arbitrarily at or between word positions. This approach simplifies incremental indexing of a document. For example, if font changes were not indexed when a document was added to the database this indexing could be added at a later time without re-indexing the entire document.

Our model

We model a text database as a string of concatenated symbols $a_1 ... a_N$ drawn from a *text alphabet* Σ_T and a *stoplist alphabet* Σ_S, where $\Sigma_T \cap \Sigma_S = \emptyset$. An index function $\mathcal{I}_T : \Sigma_T \rightarrow 2^{\{1...N\}}$ maps each symbol in the text alphabet to the set of positions in the database string where the symbol appears. No equivalent index function for symbols from the stoplist alphabet is defined; symbols from the stoplist serve merely to occupy positions in the database string and to maintain proximity relationships. The text alphabet and the stoplist alphabet are together referred to as the *database alphabet* $\Sigma_D (= \Sigma_T \cup \Sigma_S)$. The document is marked up using symbols drawn from a *markup alphabet* Σ_M, where $\Sigma_D \cap \Sigma_M = \emptyset$. An index function $\mathcal{I}_M : \Sigma_M \rightarrow 2^{\mathcal{Q}}$ maps each symbol in the markup alphabet to a set of rational numbers corresponding to the associated positions in the database string. Symbols in the markup alphabet do not appear in the database string; they are used only for indexing purposes. Combining the text alphabet and the markup alphabet into a single *index alphabet* $\Sigma = \Sigma_A \cup \Sigma_M$, we define the index function $\mathcal{I} : \Sigma \rightarrow 2^{\mathcal{Q}}$ as the union of $\mathcal{I}_A$ and $\mathcal{I}_M$. For convenience we define the value ϵ to be the smallest positioning quantum in the database — ϵ is the largest rational number for which p/ϵ is an integer for every value p in the range of the index function $\mathcal{I}$.

Figure 1 is a portion of a structured document. The document includes stage directions, speakers, speeches, lines, pages, acts, and scenes. The document demonstrates the importance of containment relationships and illustrates that a strict hierarchy is too rigid a model to capture these relationships. The first speech by the FIRST WITCH ("When shall we three...") consists of the first two lines on the page. The eighth line on the page ("I come Grey-Malkin...") contains a speech by each of the witches. In one case several lines are contained in a speech, in the other several speeches are contained in a line.

One possible representation of the document in our text database model uses words as the text alphabet:

$$\Sigma_T = \{ \texttt{again}, \texttt{air}, \texttt{alarum}, \texttt{all}, \texttt{anon}, \texttt{attendants}, \texttt{battle}, \texttt{be}, \texttt{bleeding}, \\ \texttt{bloody}, \texttt{burly}, \texttt{calls}, \texttt{can}, \texttt{captain}, \texttt{come}, \texttt{donalbain}, ... \}.$$

The stoplist alphabet consists of words that occur most commonly in English text:

$$\Sigma_S = \{ \texttt{and}, \texttt{for}, \texttt{in}, \texttt{is}, \texttt{of}, \texttt{that}, \texttt{the}, \texttt{to}, \texttt{said} \}.$$

In this instance we made the arbitrary choice to ignore case and punctuation in creating the database alphabet.

The symbols from database alphabet are concatenated in the order they appear textually to form the database string:

```
thunder and lightning enter three witches first witch when shall we three
meet again in thunder lightning or in rain second ...
```

To represent symbols in the markup alphabet we use the notation "`[`*name*" to represent the start of a named structural element and "*name*`]`" to represent the end of the named structural element. Using this notation, the start of a scene would be indexed by the symbol "`[scene`" and the end of a scene would be indexed by the symbol "`scene]`". Indexing for a portion of our example document is given in figure 2. Where possible, we choose to index markup symbols at integer positions. It is only in the case that a structural element begins and ends at the same word that we index a markup symbol halfway-between two database symbols (and so $\epsilon = \frac{1}{2}$ in this case). There are alternatives to this indexing. We could choose to index all markup symbols at the halfway point between database symbols, or choose to order the markup symbols between database symbols and give each a unique position. The exact choice depends on details of implementation and loading; the results of this paper are independent of this choice. In our experience it is usually best to limit positions to rational numbers where the denominator is a small fixed power of 2. It is then necessary to store only the numerators of these rational numbers in the database.

3. THE QUERY ALGEBRA

We represent the result of a search as a set of ranges or *extents* in the database string. Each extent is of the form (p, q), where $p \in \mathcal{Q}$ is the starting position of the extent and $q \in \mathcal{Q}$ is the end position of the extent. The length of an extent is $q - p + \epsilon$, making the smallest extents of length ϵ. The largest possible extent is $\mathcal{Z} = (-\infty, \infty)$. An extent (p, q) *overlaps* an extent (p', q') if either $p' \le p \le q'$ or $p' \le q \le q'$ but not both. An extent (p, q) is *nested* in an extent (p', q') if $p' \le p \le q \le q'$.

Let M be the cardinality of the range of the index function $\mathcal{I}$. In other words, M is the number of (rational-valued) positions in the database string that are indexed by $\mathcal{I}$. A search may be satisfied by $O(M^2)$ extents. If the query is for a particular word, every extent that includes an occurrence of the word may be considered a solution to the query. A search to find a word that occurs exactly once in the database is satisfied by at least M extents and by as many as $(\lceil M/2 \rceil + 1)(\lfloor M/2 \rfloor + 1)$, depending on the position of the word in the database string. However, many of these extents overlap and nest. In order to reduce the number of extents that result from a search, we do not include in the result set those extents that would have other result extents nested in them.

Generalized concordance lists

We refer to a set of non-nested extents as a *generalized concordance list*, or simply *GC-list*, after the concordance lists of Burkowski [7]. Burkowski's concordance lists have the property that the element extents must be non-overlapping as well as non-nesting. In the case of a search for a single word that occurs once in the database, the resultant generalized concordance list contains a single extent of length ϵ that begins and ends at the word's position. The index function $\mathcal{I}$ may be

Thunder and lightning. Enter three Witches. I.1

FIRST WITCH
When shall we three meet again?
In thunder, lightning, or in rain?

SECOND WITCH
When the hurly-burly's done,
When the battle's lost and won.

THIRD WITCH
That will be ere the set of sun.

FIRST WITCH
Where the place?

SECOND WITCH Upon the heath

THIRD WITCH
There to meet with Macbeth.

FIRST WITCH
I come Grey-Malkin.

SECOND WITCH Paddock calls!

THIRD WITCH Anon!

ALL
Fair is foul and foul is fair,
Hover through the fog and filthy air. *Exeunt*

Alarum within I.2
Enter King Duncan, Malcom, Donalbain, Lennox,
with Attendants, meeting a bleeding Captain

KING
What bloody man is that? He can report,

53

Figure 1: Text Structure in *Macbeth*.

61
`first`
`[speaker`
`[line`
	62
	`witch`
	`speaker]`
		63
		`i`
		`[speech`
			64
			`come`
				65
				`grey`

66
`malkin`
`speech]`
	67
	`second`
	`[speaker`
		68
		`witch`
		`speaker]`
			69
			`paddock`
			`[speech`
				70
				`calls`
				`speech]`

71
`third`
`[speaker`
	72
	`witch`
	`speaker]`
		$72\frac{1}{2}$
		
		`[speech`
			73
			`anon`
			`speech]`
			`line]`
				$73\frac{1}{2}$
				
				`[speaker`
				`[line`

74
`all`
`speaker]`
	75
	`fair`
	`[speech`
		76	77
			`foul`	

Figure 2: Indexing for a portion of *Macbeth*.

viewed as mapping symbols in the index alphabet onto GC-lists: The elements of the results are interpreted as extents that begin and end at a single position.

We formalize the reduction of a set of extents to a generalized concordance list as a function $\mathcal{G}$ (S). If $a = (p, q)$ and $b = (p', q')$ are extents from the database string, we use the notation $a \sqsubset b$ to indicate that a nests in b. We define the function $\mathcal{G}$ over sets of extents as:

$$\mathcal{G}\ (S) = \{a \mid a \in S \text{ and } \nexists\ b \in S \text{ such that } b \neq\ a \text{ and } b \sqsubset a\}$$

For an arbitrary set of extents S, every GC-list that is a subset S is a subset of $\mathcal{G}$ (S). In this sense, $\mathcal{G}$ (S) is the "most general" GC-list that is a subset of S. No similar function can be defined for Burkowski's concordance lists. For an arbitrary set of extents S, there is no concordance list that is a subset of S and a superset of every other concordance list that is a subset of S.

It is easily shown that no GC-list may contain more than M elements. For otherwise two elements of the GC-list would share an end point and one would nest in the other.

The elements of a GC-list are totally ordered by their end points. If $a = (p, q)$ and $b = (p', q')$ are distinct elements of a GC-list either $p < p'$ and $q < q'$, or $p > p'$ and $q > q'$. In the first case we say $a < b$ and in the second case $a > b$.

The query algebra

Each operator in the query algebra is defined over GC-lists and evaluates to a GC-list. The operators are presented in figure 3. The operators fall into three classes. The *containment operators* select the elements of a GC-list that are contained in, not contained in, contain, or do not contain the elements of a second GC-list. The containment operators may be used to formulate queries that refer to the hierarchical characteristics of structural elements in the database. The expression on the right-hand side of a containment operator acts as a filter to restrict the expression on the left-hand side — the result of the operation is a subset of the result of the left-hand side. The two *combination operators* are similar to the standard boolean operators AND and OR. The "both of" operator is similar to AND: Each extent in the result contains an extent from each operand. The "one of" operator merges two GC-lists: Each extent in the result is an extent from one of the operands. The *ordering operator* generalizes concatenation: Each extent in the result starts with an extent from the first operand and ends with an extent from the second operand. The ordering operator may be used to connect markup alphabet symbols that delineate structural elements, producing a GC-list in which each extent corresponds to one occurrence of the structural element. Examples of the use of these operators are given at the end of this section.

Elementary terms

As mentioned, the index function $\mathcal{I}$ may be viewed as mapping symbols in the index alphabet onto GC-lists. We add other types of elementary terms to our algebra. We use the symbol Σ^n to represent the GC-list of all extents of length n. The GC-list represented by Σ^n has a member extent beginning at each position in the database. The expression Σ^ϵ represents the GC-list of all extents of smallest size.

It is possible to synthesize extents from sources external to our model. For example, it is often desirable to select documents on the basis of their publication dates. The range of possibilities for date-related queries makes it difficult to represent publication date information in our model.

Nonetheless, the results of these queries can be expressed as GC-lists and manipulated using the algebra.

Properties

The operators exhibit several basic properties. The combination operators are associative and commutative:

$$\begin{aligned} A \bigtriangleup B &= B \bigtriangleup A \\ (A \bigtriangleup B) \bigtriangleup C &= A \bigtriangleup (B \bigtriangleup C) \\ A \bigtriangledown B &= B \bigtriangledown A \\ (A \bigtriangledown B) \bigtriangledown C &= A \bigtriangledown (B \bigtriangledown C) \end{aligned}$$

The $\bigtriangleup$ operator distributes across $\bigtriangledown$:

$$A \bigtriangleup (B \bigtriangledown C) = (A \bigtriangleup B) \bigtriangledown (A \bigtriangleup C)$$

The ordering operator is associative but not commutative:

$$\begin{aligned} (A \diamond B) \diamond C &= A \diamond (B \diamond C) \\ A \diamond B &\neq B \diamond A \end{aligned}$$

The containment operators exhibit an interesting version of commutativity, commutativity of containment criteria applied to a particular GC-list:

$$(A \ominus B) \oplus C = (A \oplus C) \ominus B, \text{ where } \ominus, \oplus \in \{ \lhd , \rhd , \not\lhd , \not\rhd \}$$

Query Examples

Our algebra may be used to express the *Macbeth* queries given earlier in the paper:

1. *Find plays that contain "Birnam" followed by "Dunsinane".*

   $$(\mathcal{I}(\text{“}\texttt{[play}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{play]}\text{”})) \rhd (\mathcal{I}(\text{“}\texttt{birnam}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{dunsinane}\text{”}))$$

   The index function $\mathcal{I}$ maps symbols in the index alphabet onto GC-lists. The ordering operation is used build a GC-list of plays and a GC-list of text fragments that begin with "Birnam" and end with "Dunsinane". Each extent in the result of the expression $\mathcal{I}(\text{“}\texttt{[play}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{play]}\text{”})$ exactly delimits the extent of a play. The GC-list of text fragments that begin with "Birnam" and end with "Dunsinane" is used to select from the GC-list of plays.

2. *Find fragments of text that contain "Birnam" and "Dunsinane".*

   $$\mathcal{I}(\text{“}\texttt{birnam}\text{”}) \bigtriangleup \mathcal{I}(\text{“}\texttt{dunsinane}\text{”})$$

   Since no ordering is specified, the "both of" operator is used. Member extents of the resulting GC-list either begin with "Birnam" and end with "Dunsinane", or begin with "Dunsinane" and end with "Birnam".

Containment Operators

Contained In:

$$A \lhd B = \mathcal{G}\ (\{a \mid a \in A \text{ and } \exists\ b \in B \text{ such that } a \sqsubset b\})$$

Containing:

$$A \rhd B = \mathcal{G}\ (\{a \mid a \in A \text{ and } \exists\ b \in B \text{ such that } b \sqsubset a\})$$

Not Contained In:

$$A \not\lhd B = \mathcal{G}\ (\{a \mid a \in A \text{ and } \nexists\ b \in B \text{ such that } a \sqsubset b\})$$

Not Containing:

$$A \not\rhd B = \mathcal{G}\ (\{a \mid a \in A \text{ and } \nexists\ b \in B \text{ such that } b \sqsubset a\})$$

Combination Operators

Both Of:

$$A \bigtriangleup B = \mathcal{G}\ (\{c \mid c \sqsubset\ \mathcal{Z} \text{ and } \exists\ a \in A \text{ such that } a \sqsubset c \text{ and } \exists\ b \in B \text{ such that } b \sqsubset c\})$$

One Of:

$$A \bigtriangledown B = \mathcal{G}\ (\{c \mid c \sqsubset\ \mathcal{Z} \text{ and } \exists\ a \in A \text{ such that } a \sqsubset c \text{ or } \exists\ b \in B \text{ such that } b \sqsubset c\})$$

Ordering Operator

Followed by:

$$A \diamond B = \mathcal{G}\ (\{c \mid c = (p, q') \text{ where } \exists\ (p, q) \in A \text{ and } \exists\ (p', q') \in B \text{ such that } q < p'\})$$

Figure 3: Definitions for the operators in the query algebra.

3. *Find the pages on which the word "Birnam' is spoken by a witch.*

$$\text{PAGES} \rhd (\, \mathcal{I}(\text{“}\mathtt{birnam}\text{”}) \lhd (\text{BIRNAM} \lhd (\text{S} \rhd \text{WITCH})))$$

where

$$\begin{aligned} \text{PAGES} &\equiv \mathcal{I}(\text{“}\mathtt{[page}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{page]}\text{”}) \\ \text{BIRNAM} &\equiv (\, \mathcal{I}(\text{“}\mathtt{[speech}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{speech]}\text{”})\,) \rhd \mathcal{I}(\text{“}\mathtt{birnam}\text{”}) \\ \text{S} &\equiv \mathcal{I}(\text{“}\mathtt{[speaker}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{speech]}\text{”}) \\ \text{WITCH} &\equiv (\, \mathcal{I}(\text{“}\mathtt{[speaker}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{speaker]}\text{”})\,) \rhd \mathcal{I}(\text{“}\mathtt{witch}\text{”}) \end{aligned}$$

The expressions WITCH and BIRNAM specify speakers that are witches and speeches that contain "Birnam" respectively. The expression S links speaker and speech together. The query is arranged to use the actual occurrence of the word "Birnam" to select pages. If a speech by a witch stretched between two pages and contained an occurrence of "Birnam" on each page, both pages would be selected.

4. *Find speeches that contain "toil" or "trouble" in the first line, and do not contain "burn" or "bubble" in the second line.*

$$\text{SPEECHES} \rhd (\text{FIRST2LINES} \rhd (\text{T} \diamond \text{B}))$$

where

$$\begin{aligned} \text{SPEECHES} &\equiv \mathcal{I}(\text{“}\mathtt{[speech}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{speech]}\text{”}) \\ \text{FIRST2LINES} &\equiv \mathcal{I}(\text{“}\mathtt{[speech}\text{”}) \diamond \text{LINES} \diamond \text{LINES} \\ \text{T} &\equiv \text{LINES} \rhd (\, \mathcal{I}(\text{“}\mathtt{toil}\text{”}) \bigtriangledown \mathcal{I}(\text{“}\mathtt{trouble}\text{”})\,) \\ \text{B} &\equiv \text{LINES} \not\rhd (\, \mathcal{I}(\text{“}\mathtt{burn}\text{”}) \bigtriangledown \mathcal{I}(\text{“}\mathtt{bubble}\text{”})\,) \\ \text{LINES} &\equiv \mathcal{I}(\text{“}\mathtt{[line}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{line]}\text{”}) \end{aligned}$$

The expressions T and B select extents that match criteria on the contents of the lines. The expression FIRST2LINES evaluates to a GC-list consisting of the first two lines after the start of each speech. This expression does nothing to guarantee that both lines are contained within the speech. The outermost containment operator ensures this requirement.

5. *Find a speech by an apparition that contains "fife" and that appears in a scene along with the line "Something wicked this way comes".*

$$((\text{FIFE} \lhd (\text{S} \rhd \text{APPARITION}))) \lhd (\text{SCENES} \rhd \text{B})$$

where

$$\begin{aligned} \text{FIFE} &\equiv \text{SPEECHES} \rhd \mathcal{I}(\text{“}\mathtt{fife}\text{”}) \\ \text{S} &\equiv \mathcal{I}(\text{“}\mathtt{[speaker}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{speech]}\text{”}) \\ \text{APPARITION} &\equiv (\, \mathcal{I}(\text{“}\mathtt{[speaker}\text{”}) \diamond \mathcal{I}(\text{“}\mathtt{speaker]}\text{”})\,) \rhd \mathcal{I}(\text{“}\mathtt{apparition}\text{”}) \end{aligned}$$

$$
\begin{aligned}
\text{B} &\equiv \Sigma^5 \triangleright \text{LINES} \triangleright \text{BRDBRY} \\
\text{SPEECHES} &\equiv \mathcal{I}(\text{“}\texttt{[speech}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{speech]}\text{”}) \\
\text{SCENES} &\equiv \mathcal{I}(\text{“}\texttt{[scene}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{scene]}\text{”}) \\
\text{LINES} &\equiv \mathcal{I}(\text{“}\texttt{[line}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{line]}\text{”}) \\
\text{BRDBRY} &\equiv \mathcal{I}(\text{“}\texttt{something}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{wicked}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{this}\text{”}) \\
&\quad \Diamond\ \mathcal{I}(\text{“}\texttt{way}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{comes}\text{”})
\end{aligned}
$$

This example illustrates the use of Σ^n. The expression B ensures that only lines that exactly match the quote are selected. Lines such as “Something purple and wicked this way comes” are eliminated.

The query expressions given above assume a schema on the database. The expression

$$\mathcal{I}(\text{“}\texttt{[speaker}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{speech]}\text{”})$$

occurs in several of the examples to associated speakers with their speeches. In using this expression we make the assumption that names of speakers are immediately followed by the speeches that they make. While the algebra does not depend on this assumption holding, the correctness of the query does. The schema of the database is independent of the algebra and must be described by mechanisms external to the algebra.

The algebra can be used as a tool to enforce a schema. If all speakers must be followed by a speech and all speeches must be proceeded by a speaker, the following expressions must evaluate to the empty GC-list:

$$(\,\mathcal{I}(\text{“}\texttt{speaker]}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{[speaker}\text{”})\,) \not\triangleright (\,\mathcal{I}(\text{“}\texttt{[speech}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{speech]}\text{”})\,)$$

$$(\,\mathcal{I}(\text{“}\texttt{speech]}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{[speech}\text{”})\,) \not\triangleright (\,\mathcal{I}(\text{“}\texttt{[speaker}\text{”}) \ \Diamond\ \mathcal{I}(\text{“}\texttt{speaker]}\text{”})\,)$$

4. A FRAMEWORK FOR IMPLEMENTATION

Start points and end points place identical total orders on the elements of a GC-list. We exploit this total order to develop a framework for efficiently implementing our algebra. The approach consists of indexing into GC-lists. The total order is used as the basis for this indexing. Given a GC-list and a position in the database we index into the GC-list to find the extent that is in some sense “closest to” that position in the database. We begin with an example and follow this with a formal exposition of the framework.

Consider evaluating the expression $A \Diamond B$ (see figure 4). An extent from the resultant GC-list starts with an element from A and ends with an element from B. Suppose (p, q) is the first extent in A. If (p', q') is the first extent from B with $p' > q$ then q' must be the end of the first extent of $A \Diamond B$. We index into B to find the first extent with $p' > q$. The last extent from A that ends before p' starts the first extent of $A \Diamond B$. We index into A to find the greatest extent (p'', q'') where $q'' < p'$. The extent (p'', q') is the first solution to $A \Diamond B$. Indexing first into B and then into A in this manner gives us the first extent in $A \Diamond B$ directly in two steps. The next solution

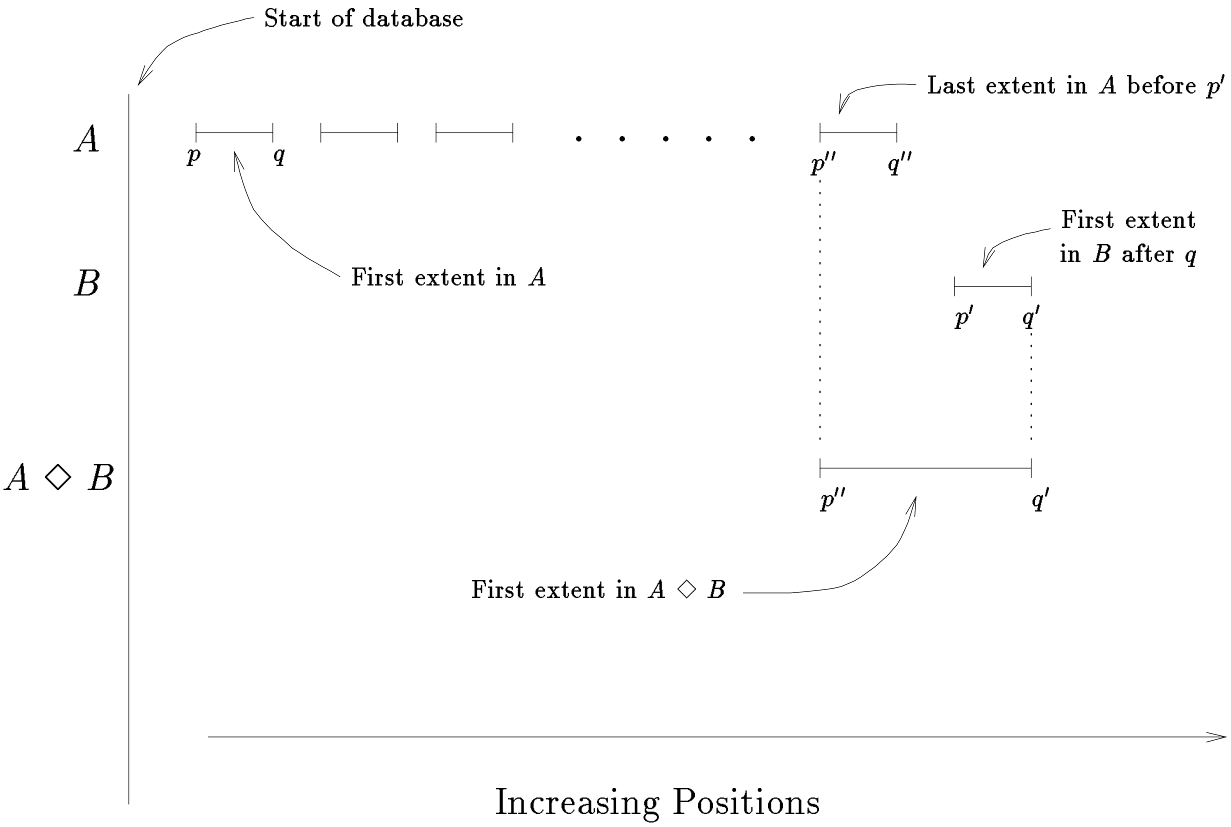

Figure 4: Evaluating $A \diamond B$.

to $A \diamond B$ begins after p''. We index into A to produce the first extent after p''. This procedure of successively indexing into A and B can be continued to find the remaining extents in $A \diamond B$.

Our implementation framework consists of four access functions that allow indexing into GC-lists in various ways. Each of the access functions represents a variation on the notion of "closest extent" in a GC-list to a specified position in the database. We implement the four access functions for each operator in our algebra using the access functions of its operands.

The access function $\tau\ (S, k)$ represents the first extent in the GC-list S starting at or after the position k:

$$\tau\ (S,k) = \begin{cases} (p,q) & \text{if } \exists\ (p,q) \in S \text{ such that } k \leq p \\ & \quad \text{and } \nexists\ (p',q') \in S \text{ such that } k \leq p' < p \\ (\infty,\infty) & \text{if } \nexists\ (p,q) \in S \text{ such that } k \leq p \end{cases}$$

The access function $\rho\ (S, k)$ represents the first extent in S ending at or after the position k:

$$\rho\ (S,k) = \begin{cases} (p,q) & \text{if } \exists\ (p,q) \in S \text{ such that } k \leq q \\ & \quad \text{and } \nexists\ (p',q') \in S \text{ such that } k \leq q' < q \\ (\infty,\infty) & \text{if } \nexists\ (p,q) \in S \text{ such that } k \leq q \end{cases}$$

The access functions $\tau'\ (S, k)$ and $\rho'\ (S, k)$ are the converses of τ and ρ. The access function $\tau'\ (S, k)$ represents the last extent in S ending at or before the position k; the access function $\rho'\ (S, k)$ represents the last extent in S starting at or before the position k:

$$\tau'\ (S,k) = \begin{cases} (p,q) & \text{if } \exists\ (p,q) \in S \text{ such that } k \geq q \\ & \quad \text{and } \nexists\ (p',q') \in S \text{ such that } k \geq q' > q \\ (-\infty,-\infty) & \text{if } \nexists\ (p,q) \in S \text{ such that } k \geq q \end{cases}$$

$$\rho'\ (S,k) = \begin{cases} (p,q) & \text{if } \exists\ (p,q) \in S \text{ such that } k \geq p \\ & \quad \text{and } \nexists\ (p',q') \in S \text{ such that } k \geq p' > p \\ (-\infty,-\infty) & \text{if } \nexists\ (p,q) \in S \text{ such that } k \geq p \end{cases}$$

Figure 5 and figure 6 give definitions of τ and ρ over the operators when $k < \infty$. For simplicity, the case of $k = \infty$ is omitted from the figures. In that case we have:

$$\tau\ (S,\infty) = \rho\ (S,\infty) = (\infty,\infty).$$

The notation used in the figures is loosely based on the functional programming language ML [24]. An expression of the form "**let** *definitions* **in** *expression*" yields the value of the expression following the **in**, evaluated in the context of the definitions following the **let**. A conditional expression " **if** *condition* **then** *expression* **else** *expression* " evaluates to the expression following the **then** if the boolean condition following the **if** is true, and evaluates to the expression following the **else** if the boolean condition is false. Equations for τ' and ρ' are not given; they can be easily inferred from those for τ and ρ.

We examine in detail the equation for $\tau\ (A \lhd B, k)$ (see figure 7). This equation yields the first element of $A \lhd B$ that starts at or after the position k. The extent $(p, q) = \tau\ (A, k)$ is taken as a candidate solution. For (p, q) to be the solution it must be contained in an extent of B. An extent of B containing (p, q) must end at or after q. The first such extent is $(p', q') = \rho\ (B, q)$.

Containment

$$
\begin{array}{l}
\tau\ (A \lhd B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \tau\ (A, k) \\
\quad\quad (p', q') = \ \rho\ (B, q) \\
\quad \textbf{in} \\
\quad\quad \textbf{if } p' \leq p \textbf{ then} \\
\quad\quad\quad (p, q) \\
\quad\quad \textbf{else} \\
\quad\quad\quad \tau\ (A \lhd B, p')
\end{array}
$$

$$
\begin{array}{l}
\rho\ (A \lhd B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \rho\ (A, k) \\
\quad \textbf{in} \\
\quad\quad \tau\ (A \lhd B, p)
\end{array}
$$

$$
\begin{array}{l}
\tau\ (A \rhd B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \tau\ (A, k) \\
\quad \textbf{in} \\
\quad\quad \rho\ (A \rhd B, q)
\end{array}
$$

$$
\begin{array}{l}
\rho\ (A \rhd B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \rho\ (A, k) \\
\quad\quad (p', q') = \ \tau\ (B, p) \\
\quad \textbf{in} \\
\quad\quad \textbf{if } q' \leq q \textbf{ then} \\
\quad\quad\quad (p, q) \\
\quad\quad \textbf{else} \\
\quad\quad\quad \rho\ (A \rhd B, q')
\end{array}
$$

$$
\begin{array}{l}
\tau\ (A \ntriangleleft B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \tau\ (A, k) \\
\quad\quad (p', q') = \ \rho\ (B, q) \\
\quad \textbf{in} \\
\quad\quad \textbf{if } p' > p \textbf{ then} \\
\quad\quad\quad (p, q) \\
\quad\quad \textbf{else} \\
\quad\quad\quad \rho\ (A \ntriangleleft B, q' + \epsilon)
\end{array}
$$

$$
\begin{array}{l}
\rho\ (A \ntriangleleft B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \rho\ (A, k) \\
\quad \textbf{in} \\
\quad\quad \tau\ (A \ntriangleleft B, p)
\end{array}
$$

$$
\begin{array}{l}
\tau\ (A \ntriangleright B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \tau\ (A, k) \\
\quad \textbf{in} \\
\quad\quad \rho\ (A \ntriangleright B, q)
\end{array}
$$

$$
\begin{array}{l}
\rho\ (A \ntriangleright B, k) = \\
\quad \textbf{let} \\
\quad\quad (p, q) = \ \rho\ (A, k) \\
\quad\quad (p', q') = \ \tau\ (B, p) \\
\quad \textbf{in} \\
\quad\quad \textbf{if } q' > q \textbf{ then} \\
\quad\quad\quad (p, q) \\
\quad\quad \textbf{else} \\
\quad\quad\quad \tau\ (A \ntriangleright B, p' + \epsilon)
\end{array}
$$

Figure 5: τ and ρ for the containment operators.

Combination

$\tau\ (A \bigtriangleup B, k) =$
let
$(p, q) = \tau\ (A, k)$
$(p', q') = \tau\ (B, k)$
$(p'', q'') = \tau'\ (A, \max(q, q'))$
$(p''', q''') = \tau'\ (B, \max(q, q'))$
in
$(\min(p'', p'''),\ \max(q'', q'''))$

$\rho\ (A \bigtriangleup B, k) =$
let
$(p, q) = \tau'\ (A \bigtriangleup B, k - \epsilon)$
in
$\tau\ (A \bigtriangleup B, p + \epsilon)$

$\tau\ (A \bigtriangledown B, k) =$
let
$(p, q) = \tau\ (A, k)$
$(p', q') = \tau\ (B, k)$
in
if $q < q'$ **then**
(p, q)
else if $q > q'$ **then**
(p', q')
else
$(\max(p, p'),\ q)$

$\rho\ (A \bigtriangledown B, k) =$
let
$(p, q) = \tau'\ (A \bigtriangledown B, k - \epsilon)$
in
$\tau\ (A \bigtriangledown B, p + \epsilon)$

Ordering

$\tau\ (A \diamond B, k) =$
let
$(p, q) = \tau\ (A, k)$
$(p', q') = \tau\ (B, q + \epsilon)$
$(p'', q'') = \tau'\ (A, p' - \epsilon)$
in
(p'', q')

$\rho\ (A \diamond B, k) =$
let
$(p, q) = \tau'\ (A \diamond B, k - \epsilon)$
in
$\tau\ (A \diamond B, p + \epsilon)$

Figure 6: τ and ρ for the combination and ordering operators.

There are now two cases: 1) If $p' \leq p$ then (p, q) is contained in (p', q') and (p, q) is the solution to $\tau\ (A \lhd B, k)$. 2) Otherwise $p' > p$ and (p, q) is not contained in (p', q'). In this case, (p, q) is not contained in any extent of B. For if there existed an extent (p'', q'') in B that contained (p, q) we would have:

$p' > p$	since (p', q') does not contain (p, q)
$p \geq p''$	since (p'', q'') contains (p, q)
$p'' \geq p'$	since (p'', q'') is after (p', q') in the GC-list B

This is a contradiction and (p, q) is not a solution to $\tau\ (A \lhd B, k)$. The solution to $\tau\ (A \lhd B, k)$ must start at or after p'. Thus, $\tau\ (A \lhd B, k) = \tau\ (A \lhd B, p')$.

A similar case analysis may be applied to understand the remaining equations in figures 5 and 6. This case analysis may be formalized into a straightforward but tedious proof of correctness for the equations.

Interpreting the equations operationally as recursive functions expressed in a functional-style programming language gives us the core of a text database search algorithm. Two additional pieces are missing from the algorithm: an implementation of the access functions for elementary terms in the algebra — symbols from the index alphabet and Σ^n — and a top level driver procedure that evaluates a query and generates a GC-list. A discussion of the implementation of the access functions for the elementary terms in the algebra appears later in this section. One possible driver procedure is in figure 8. The driver procedure uses iterative calls to τ to generate the resultant GC-list. An equivalent driver procedure can be written using ρ. The corresponding driver procedures using either τ' or ρ' generate the GC-list in reverse order.

During the evaluation of a query using the driver procedure $\mathcal{P}$ of figure 8 the number of calls to access functions for sub-queries by the equations of figure 5 and figure 6 is linear in the sum of the size of the GC-lists for the sub-queries. This observation ignores the effects of indexing into the GC-lists. These effects can be considerable. A direct evaluation of the expression in the first example on page 11 requires at most

$$O(\min(\mid \mathcal{I}(\text{``}\texttt{[play}\text{''}) \mid, \mid \mathcal{I}(\text{``}\texttt{play]}\text{''}) \mid, \mid \mathcal{I}(\text{``}\texttt{birnam}\text{''}) \mid, \mid \mathcal{I}(\text{``}\texttt{dunsinane}\text{''}) \mid))$$

calls to access functions for index alphabet symbols. Quantifying the effects of this indexing requires modelling of expected queries and occurrence patterns of the symbols in the index alphabet; this analysis is beyond the scope of this paper.

Fixed-Size extents

The symbol Σ^n represents the GC-list of all extents of length n. Implementation is straightforward:

$$\begin{aligned}
\tau\ (\Sigma^n, k) &= (k, k + n - \epsilon) \\
\rho\ (\Sigma^n, k) &= (k - n + \epsilon, k) \\
\tau'\ (\Sigma^n, k) &= \rho\ (\Sigma^n, k) \\
\rho'\ (\Sigma^n, k) &= \tau\ (\Sigma^n, k)
\end{aligned}$$

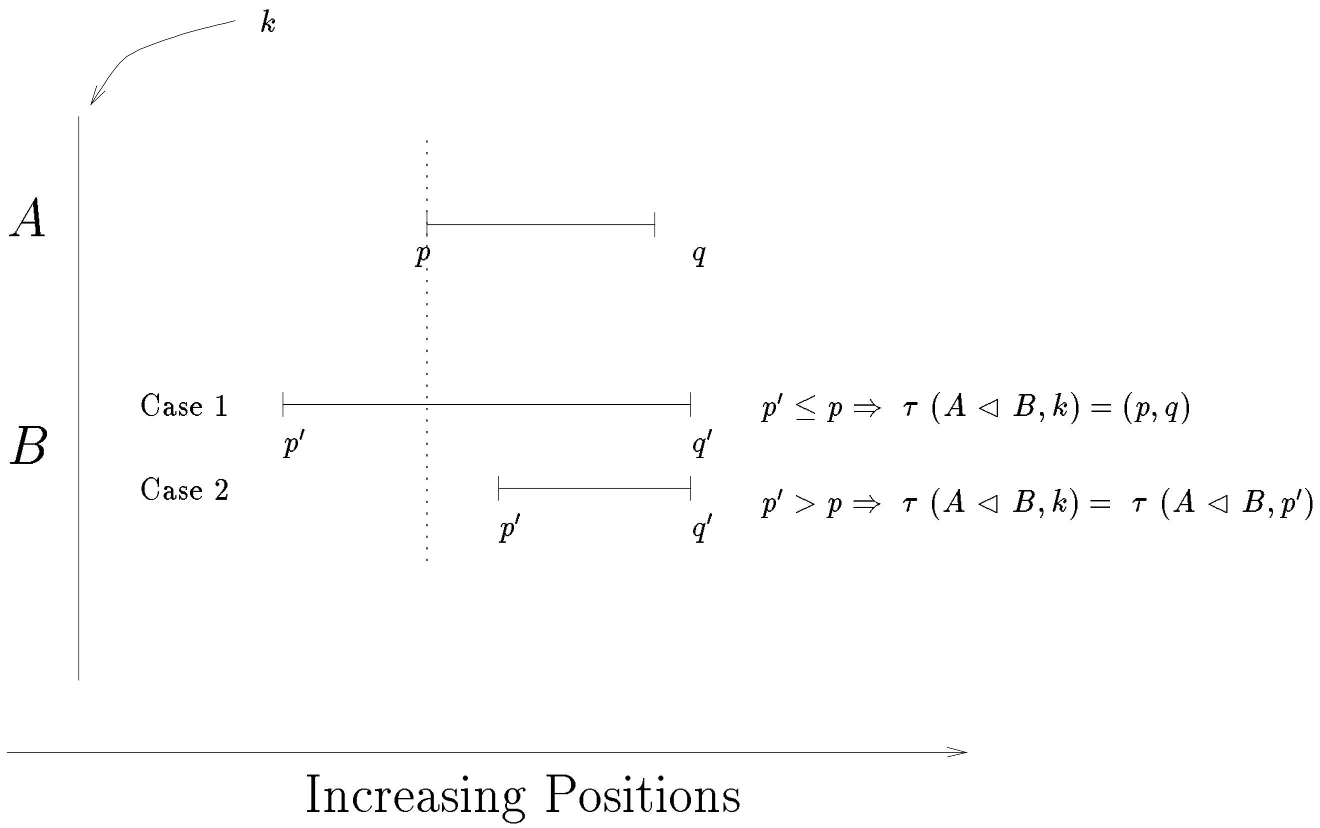

Figure 7: Evaluating $A \lhd B$.

$\mathcal{P}(S) =$
  $(p, q) = \tau\ (S, -\infty)$
  while $p \neq \infty$ **loop**
    Output (p, q)
    $(p, q) = \tau\ (S, p + \epsilon)$
  end loop

Figure 8: Driver procedure

Index organization

Standard data structures for inverted lists may be used to build implementations of τ and ρ for the database index [20, pages 552–554]. Figure 9 shows the organization of an inverted list data structure. The dictionary maps each index symbol into a range in the index. For each index symbol, the index contains a sorted list of database positions where the symbol occurs. For a particular symbol, a binary search implements the four access functions with $O(\log n)$ efficiency, where n is the number of occurrences of the symbol in the database. Other data structures, such as B-trees [20, pages 473–479] or surrogate subsets [6], may be used to provide $O(\log n)$ implementations that additionally permit efficient insertions and deletions.

By slightly extending the data structure of figure 9 it is possible to store any GC-list as an inverted list. Each index element would contain the start and end position for an extent. Since the start and end positions place identical total orders on the elements of a GC-list, any data structure usable to implement access functions for index symbols will serve equally well to implement access functions for GC-lists. In this way, GC-lists for frequently-posed queries could be pre-computed and stored as part of the database.

4. DISCUSSION

The standard boolean search algebra is a special case of the algebra presented in this paper. If we assume that the symbol DOCS represents the GC-list of documents in a database, the three basic boolean operators are implemented as:

$$\begin{aligned} \text{A AND B} &\equiv (\textsc{docs} \triangleright \text{A}) \bigtriangleup (\textsc{docs} \triangleright \text{B}) \\ \text{A OR B} &\equiv (\textsc{docs} \triangleright \text{A}) \bigtriangledown (\textsc{docs} \triangleright \text{B}) \\ \text{A NOT B} &\equiv (\textsc{docs} \triangleright \text{A}) \not\triangleright (\textsc{docs} \triangleright \text{B}) \end{aligned}$$

Similar definitions exist for proximity, concatentation, and containment in titles, abstracts, paragraphs or other pre-defined textual units.

Sacks-Davis et al. [26] have classified types of queries that a retrieval system for structured text should support. The queries in this classification are representative of the queries supported by the various proposals for searching hierarchically-organized text reviewed in the introduction to this paper. The query types generalize the standard boolean search algebra in a number of ways:

1. *Restriction of query scope to document components.* For example,

   ```
   Find document with <section> containing "text" and "retrieval"
   ```

   This class of query retrieves a set of documents that contain a component of the specified type that satisfies the specified predicate.

2. *Retrieval of a component other than a document.* For example,

   ```
   Find <section> containing "text" and "retrieval"
   ```

   In this case the result of the query is a set of document sections, rather than entire documents.

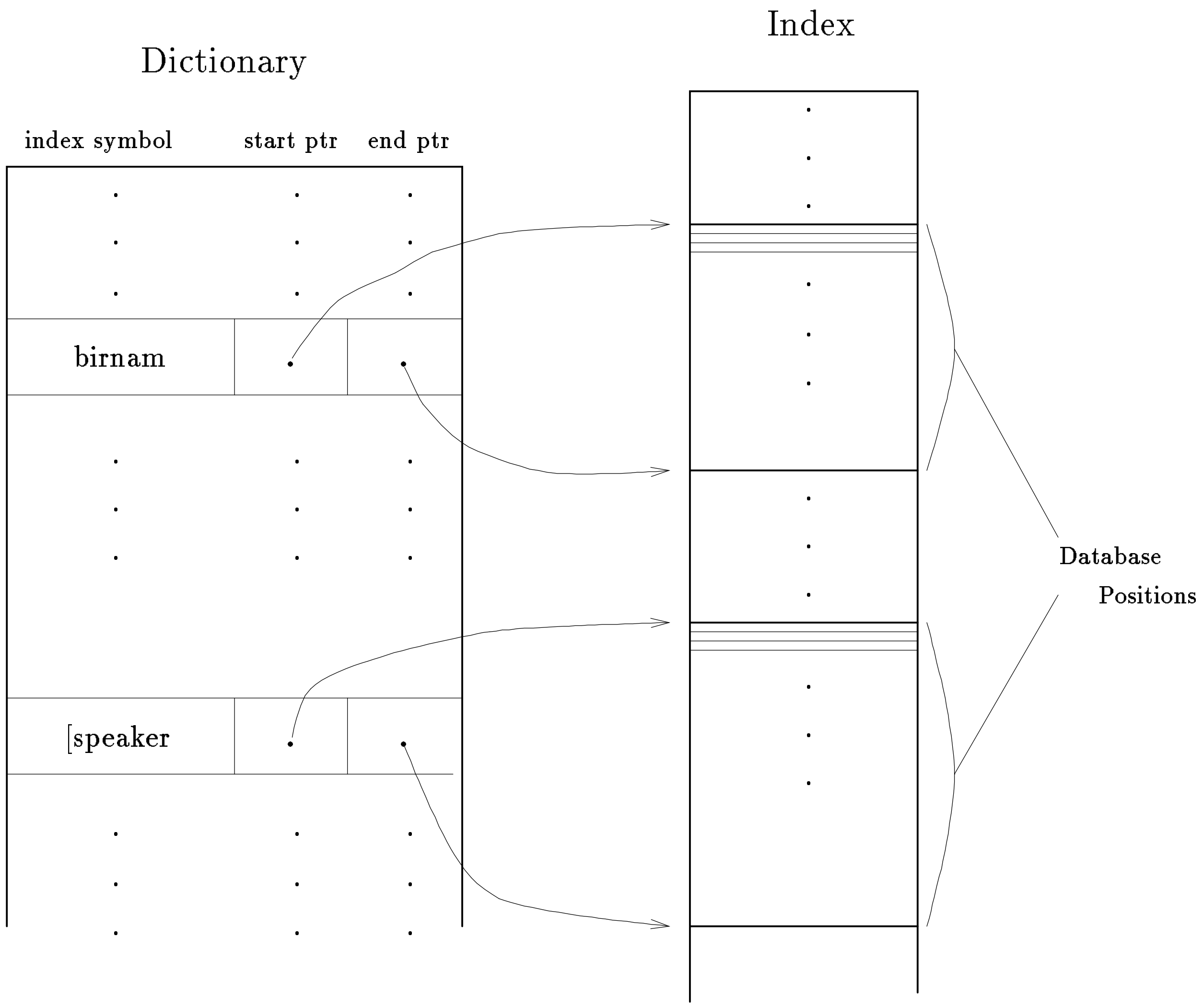

Figure 9: Inverted lists.

3. *Retrieval based on document structure.*

```
Find <heading> within a <section>
```

4. *Retrieval of multiple component types.*

```
Find <article> or <book> containing "text" and "retrieval"
```

The view here is that a query retrieves a set of (usually named) document components that satisfy a specified selection criteria. In this view, the result of query is effectively given a data type by the schema. As the translation of the boolean expressions makes apparent, a notion of containment must alway be present, either implicitly or explictly, in a query such as "A AND B". The results of this paper are in direct contrast with this view. Our algebra can express all of the queries outlined above, but operates over a single uniform data type.

The two properties that characterize the extents in a GC-list — non-nesting and overlapping — are key to the simplicity and expressiveness of the algebra. Given a corpus that contains the speech

All hail MacBeth! Hail to thee, Thane of Cawdor

the solution to the query

$\mathcal{I}(\text{“}\texttt{hail}\text{”}) \ \triangle \ \mathcal{I}(\text{“}\texttt{Macbeth}\text{”})$

will include both the occurrence of the string "hail Macbeth" and the occurrence of the string "Macbeth! Hail" that appear in the quote. Without the non-nesting property, a great many extents would be included in the solution. Without the overlapping property, an arbitrary choice would have to made concerning which of the two to include. No implicit containment is required.

The PAT text search system [11, 27] is closest in spirit to our work. A hierarchical text structure is not assumed. However, our maxim of indexing structure not markup is not followed, and the indexing of structure in the PAT system is limited in comparison to our approach. Markup tags are treated the same as any other text, and the only structure that can be queried is that which is explicit in the tagging. It is not possible, for example, to add indexing of analogous structural elements from documents with differing formats. Query operators in PAT are similar to the operators of our algebra, but no equivalent of our "both of" ($\triangle$) operator (or consequently of the generalized combination operator of the appendix) is provided. With a few superficial exceptions, queries expressible in PAT are a subset of those expressible in our algebra. PAT does not use a uniform data type for query results, and this causes significant semantic problems [27]. In particular, the result type of a query expression can depend on subtle properties of the text in the database. The addition of a single document to the database can change the result type of a query. Finally, PAT is implemented with special-purpose data structures [12]. PAT has no implementation framework amenable to the use of inverted lists.

5. CONCLUSIONS AND FUTURE WORK

This paper presents a simple model for structured text and a search algebra based on the model. The expressiveness of the algebra is illustrated with a variety of examples and by a comparison with existing work. The algebra uses GC-lists uniformly as both results and operands. A key feature of the algebra is the use of containment relationships rather than hierarchical relationships. The algebra does not preclude the enforcement of a hierarchy or other schema. Rather, the algebra is independent of any schema and can be used as a tool to ensure that a schema holds. Finally, the algebra may be efficiently implemented.

The research described in this paper could be extended in several directions:

1. *Support for indirection.* Queries based on indirect document structure (such as footnotes, references, or hypertext links) are not supported by our algebra. Ideally, we would be able to formulate queries that reference text that is indirectly associated with an extent.

2. *Co-existence with the relational model.* We have discussed the generation of synthetic extents. These synthetic extents may be generated through the use of queries to a relational database. It is also possible to extend the relational algebra with text query capabilities [1, 3, 14] and our algebra would form a useful basis for such an effort. Finally, it is possible to view tables in a relational database as structured text and search them using our algebra. A companion paper to this work [9] contains some further development of this last idea.

3. *Use of the algebra as a intermediate language.* A user needs a reasonable level of sophistication to work directly with the algebra. In some cases, the algebra is more suitable as an intermediate language between a user interface layer and an underlying search engine. The user interface would likely be graphical in nature; relevance feedback [16] and other heuristic techniques might be incorporated into this user interface.

4. *Extensions to ranked retrieval.* Our algebra may be viewed be view as a generalization of the standard boolean query algebra. Statistically-based ranked retrieval is another important and standard retrieval technique [17, 28, 29]. The results in the appendix to this paper and the discussion of reference [26] provide starting points for the application of relevance ranking techniques to structured text search.

5. *Improved evaluation strategies* Query evaluation by the direct application of the framework in this paper proceeds in a bottom-up fashion. The framework may be used as foundation for developing combined access plans for entire queries or sub-queries. Caching the solutions to frequently-posed queries should also speed evaluation. We are actively researching this area.

ACKNOWLEDGEMENTS

This research is part of a larger investigation into multi-server, multi-user retrieval engines for very large text databases. The authors gratefully acknowledge the financial support of this effort by the Government of Ontario through its Information Technology Research Centre. The Natural Science and Engineering Research Council of Canada provided additional financial support.

REFERENCES

[1] Air Transport Association. *Advanced Retrieval Standard — Structured Fulltext Query Language (SFQL).* ATA 89-9C.SFQL.

[2] Elisa Bertino, Fausto Rabitti, and Simon Gibbs. Query processing in a multimedia document system. *ACM Transactions on Office Information Systems*, 6(1):1–41, January 1988.

[3] G. E. Blake, M. P. Consens, P. Kilpeläinen, P.-Å. Larson, T. Snider, and F. W. Tompa. Text/relational database management systems — Harmonizing SQL and SGML. In *Proceedings of the First International Conference on Applications of Databases*, pages 267–280, Vadstena, Sweden, June 1994.

[4] G. Elizabeth Blake, Tim Bray, and Frank Wm. Tompa. Shortening the OED — Experience with a grammar-defined database. *ACM Transactions on Information Systems*, 10(3):213–232, July 1992.

[5] M. Bryan. *SGML — An Author's Guide to the Standard Generalized Markup Language.* Addison-Wesley, Reading, Massachusetts, 1988.

[6] Forbes J. Burkowski. Surrogate subsets — A free space management strategy for the index of a text retrieval system. In *Proceedings of the 13th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 211–226, Brussels, 1990.

[7] Forbes J. Burkowski. An algebra for hierarchically organized text-dominated databases. *Information Processing and Management*, 28(3):333–348, 1992.

[8] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 313–324, Minneapolis, Minnesota, May 1994.

[9] Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. Schema-independent retrieval from hetrogeneous structured text. Technical Report CS-94-39, University of Waterloo Computer Science Department, Waterloo, Ontario, Canada N2L 3G1, November 1994. Available by anonymous ftp as `ftp://cs-archive.uwaterloo.ca/cs-archive/CS-94-39`. A version of this report will appear in the *Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, April 1995.

[10] Latha S. Colby and Dirk Van Gucht. A grammar model for databases. Technical Report 282, Indiana University Computer Science Department, Bloomington, Indiana 47405-4101, June 1989.

[11] Heather Fawcett. *A Text Searching System — PAT 3.3 User's Guide.* Centre for the New Oxford English Dictionary, University of Waterloo, 1989.

[12] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. New indices for text — PAT trees and PAT arrays. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval — Data Structures and Algorithms*, chapter 5, pages 66–82. Prentice Hall, Englewood Cliffs, NJ, 1992.

[13] Gaston H. Gonnet and Frank Wm. Tompa. Mind your grammar — A new approach to modelling text. In *Proceedings of the 13th VLDB Conference*, pages 339–346, Brighton, England, 1987.

[14] Ralf Hartmut Güting, Roberto Zicari, and David M. Choy. An algebra for structured office documents. *ACM Transactions on Office Information Systems*, 7(4):123–157, April 1989.

[15] Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. A grammar-based approach towards unifying hierarchical data models. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 263–272, Portland, Oregon, 1989.

[16] Donna Harman. Relevance feedback and other query modification techniques. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval — Data Structures and Algorithms*, chapter 11, pages 241–263. Prentice Hall, Englewood Cliffs, NJ, 1992.

[17] Donna Harman and Gerald Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, December 1990.

[18] International Standards Organization. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, October 1986. ISO 8879.

[19] International Standards Organization. *Information Processing — Text and Office Systems — Office Document Architecture (ODA) and Interchange Format*, March 1988. ISO 8613.

[20] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, Massachusetts, 1973.

[21] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984.

[22] Arjan Loeffen. Text databases — A survey of text models and systems. *SIGMOD Record*, 23(1):97–106, March 1994.

[23] I. A. Macleod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3), 1991.

[24] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, Cambridge, Massachusetts, 1990.

[25] J. F. Ossanna. NROFF/TROFF user's manual. Computing Science Technical Report 54, Bell Laboratories, Murray Hill, New Jersey, 1976.

[26] Ron Sacks-Davis, Timothy Arnold-Moore, and Justin Zobel. Database systems for structured documents. In *Proceedings of the International Symposium on Advanced Database Technologies and Their Implementation*, pages 272–283, Nara, Japan, October 1994. Invited paper.

[27] Airi Salminen and Frank Wm. Tompa. PAT expressions — An algebra for text search. Technical Report OED-92-02, UW Centre for the New Oxford English Dictionary, University of Waterloo, July 1992.

[28] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Computer Science Series. McGraw-Hill, New York, 1983.

[29] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, second edition, 1979.

[30] Gary Wolf. Who owns the law? *Wired*, May 1994.

APPENDIX. A GENERALIZED COMBINATION OPERATOR

Each extent in the solution to $A_0 \bigtriangleup A_1 \bigtriangleup A_2 ... \bigtriangleup A_{m-1}$ contains an extent from each of $A_0 ... A_{m-1}$. Each extent in the solution to $A_0 \bigtriangledown A_1 \bigtriangledown A_2 ... \bigtriangledown A_{m-1}$ consists of an extent from one of $A_0 ... A_{m-1}$. These two expressions represent extremes of a more general operation: the combination of n extents from m GC-lists. The combination operators may be be used in concert to build these combinations. For example, each extent in

$$(A \bigtriangleup (B \bigtriangledown C)) \bigtriangledown (B \bigtriangleup (A \bigtriangledown C)) \bigtriangledown (C \bigtriangleup (A \bigtriangledown B))$$

contains an extents from two of the three GC-lists: A, B and C. Unfortunately, following this pattern, an expression for combining n extents from m GC-lists has size

$$m \binom{m}{n-1}.$$

Nonetheless, the operation has intuitive appeal and is of significant practical use. A common situation in which this operation is of particular use is in selecting documents that contain a few of a large number of terms. During the early stages of a search session this operation can assist in narrowing down a list of search terms to those that retrieve the most relevant documents. We extend our algebra with an "n of m" operator that has a direct implementation.

Formally, we define the "n of m" operator as follows:

$$n \bigtriangleup (A_0, ..., A_{m-1}) = \mathcal{G}\ (\{c \mid\ \ |\{A \mid A \in \{A_0, ..., A_{m-1}\} \text{ and } \exists a \in A \text{ such that } a \sqsubset c\}| = n\ \})$$

Each extent in $n \bigtriangleup (A_0, ..., A_{m-1})$ contains an element from exactly n of the $A_0, ..., A_{m-1}$.

Definitions for the access functions τ and ρ are generalizations of those for $\bigtriangleup$ and $\bigtriangledown$:

$$\begin{aligned}
&\tau\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k) = \\
&\quad \textbf{let} \\
&\qquad (p_i, q_i) = \tau\ (A_i, k) \qquad (0 \le i < m) \\
&\qquad q \in \{q_i\} \text{ such that } |\{q_i \mid q_i \le q\}| = n \\
&\qquad \{B_0, ..., B_{n-1}\} = \{A_i \mid q_i \le q\} \\
&\qquad (p'_j, q'_j) = \tau'\ (B_j, q) \qquad (0 \le j < n) \\
&\quad \textbf{in} \\
&\qquad (\min(p'_0, ..., p'_{n-1}),\ q)
\end{aligned}$$

$$\begin{aligned}
&\rho\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k) = \\
&\quad \textbf{let} \\
&\qquad (p, q) = \tau'\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k - \epsilon) \\
&\quad \textbf{in} \\
&\qquad \tau\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ p + \epsilon)
\end{aligned}$$

The equation for $\tau\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k)$ first evaluates $\tau\ (A_i, k)$ for each of the sub-queries $A_0, A_1, ..., A_{m-1}$. Then let q be the end point of the first n of the resultant extents and let $B_0, ..., B_{n-1}$ be those members of $A_0, A_1, ..., A_{m-1}$ that end before q. The expression $\tau'\ (B_j, q)$ is evaluated for each of the $B_0, ..., B_{n-1}$. The resulting extents span the solution to $\tau\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k)$.