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

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

Dept. of Computer Science[*]
University of Waterloo, Waterloo, Canada

Technical Report CS-94-30
August 24, 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 solve the elementary terms in the query. Inverted indices yield search times that compare favourably to those for full-text boolean searches.

[*]Email concerning this paper should be sent to `claclark@plg.uwaterloo.ca`.

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 lazy 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.

Related work

Most commercial text database systems provide an extended boolean algebra for formulating queries. Salton and McGill [14] 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.

Several proposals for dealing with document structure have been made. All of these proposals view document structure as hierarchical. Gonnet and Tompa [5] propose the use of a context-free grammar to express a text database schema and describe an algebra of operations manipulating parsed text. Gyssens et al. [7] also propose a grammar-based model. Burkowski [4] proposes a query algebra that exploits containment relationships between levels in a document structure hierarchy. Both Güting et al [6] and the draft Structured Fulltext Query Language (SFQL) standard [1] extend the relational model to support hierarchically structured documents. Document structure that is not hierarchical in nature is not well supported by these proposals. Gonnet and Tompa are unique in not making the assumption that a single hierarchy is sufficient to describe all structure in a document; they provide operators for reparsing and transforming parsed text according to different grammars.

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 final section summarizes our work and discusses future work. A simple and useful extension to 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 structured text model must be flexible.

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 [4, 5, 6, 7]. 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 [15]. Databases not licenced by this publisher cannot index and report citations using these copyrighted page numbers, making an unlicenced 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.

Document markup must be handled cleanly.

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 [2, 8], ODA [9], TEX [11] and `troff` [13] — 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 $\mathcal{X} = a_1 ... a_N$ 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 $\mathcal{I}_M : \Sigma_M \to 2^{\mathcal{Q}}$ 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 $\mathcal{I} : \Sigma \to 2^{\mathcal{Q}}$ 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 — p/ϵ is an integer for any 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:

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

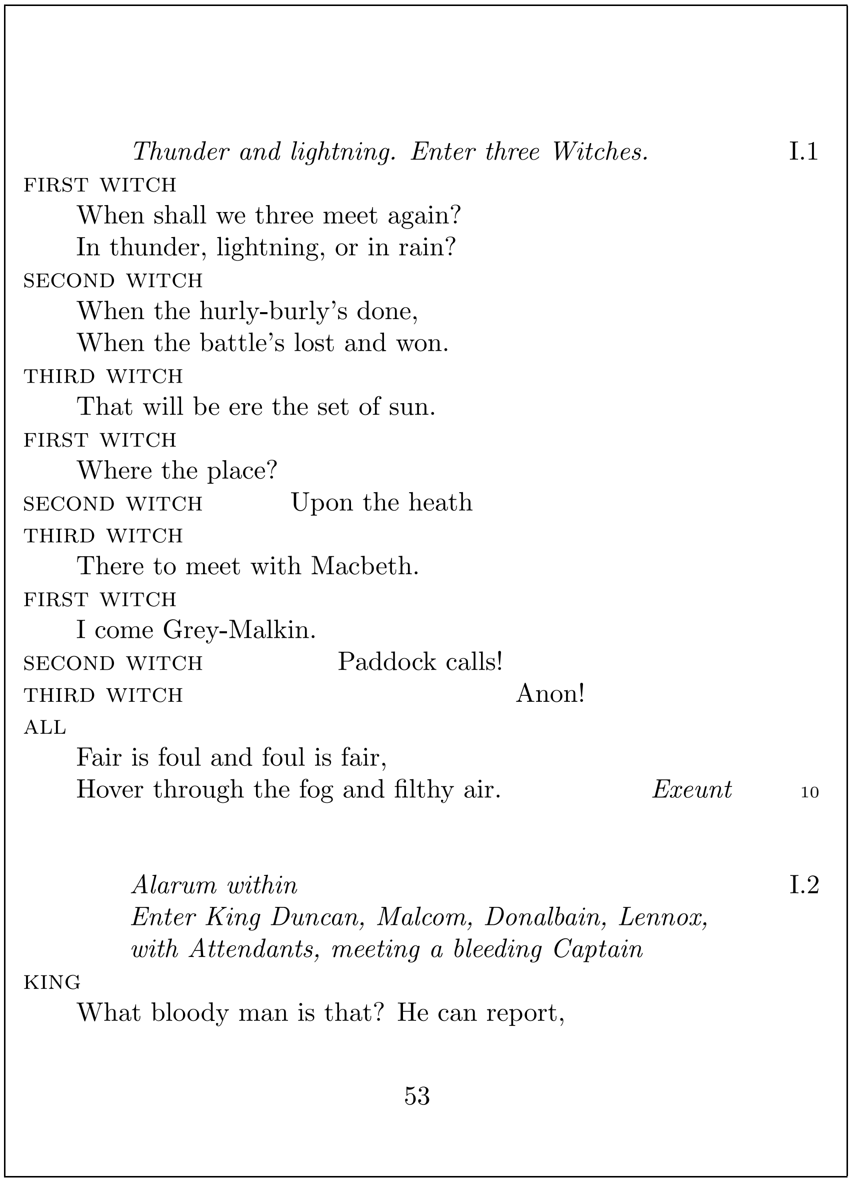
The stoplist alphabet consists of words that occur most commonly in English text:

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

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.



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*.

3. THE QUERY ALGEBRA

The result of a search is a set of ranges or *extents* in the database string that satisfy the specified query. 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.

Let M be the cardinality of the range of the index function $\mathcal{I}$. 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. A search to find a particular 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. Every extent that includes the word is a solution to the query. However, these extents overlap and nest. In order to reduce the number of extents that result from a search we apply the simple rule of eliminating extents that wholly contain other extents.

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 [4]. 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 that begins and ends at the word's position, a much more satisfactory result. The index function $\mathcal{I}$ may be 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 is a subextent of b (i.e. $p \geq p'$ and $q \leq q'$). We define the function $\mathcal{G}$ over sets of extents as:

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

Every GC-list that is a subset of a set of extents 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. Burkowski's concordance lists requires that the element extents must be non-overlapping and therefore have no such general subset list.

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 would nest. 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 $a < b$ 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 are used to formulate queries that refer to the hierarchical characteristics of structural elements in the database. 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 two 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 $\bigtriangleup$ and $\bigtriangledown$ operators together form a field, where the empty GC-list $\emptyset$ is the additive identity and Σ^ϵ is the multiplicative identity.
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:

$$(A \ominus B) \oplus C = (A \oplus C) \ominus B, \text{ where } \ominus, \oplus \in \{ \lhd , \rhd , \ntriangleleft , \ntriangleright \}$$

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

One Of:
$A \bigtriangledown B = \mathcal{G}\ (\{c \mid c \sqsubset\ (-\infty, \infty) \text{ and } \exists\ a \in A \text{ and } \exists\ b \in B \text{ such that } a \sqsubset c \text{ or } b \sqsubset c\})$

Ordering Operator

Followed by:
$A \diamond B = \mathcal{G}\ (\{c \mid c \sqsubset\ (-\infty, \infty) \text{ and } \exists\ a \in A \text{ and } \exists\ b \in B \text{ such that } c = ayb\})$

Figure 3: Definitions for operators in the query algebra.

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 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".*

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

   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".

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

   $$\textsc{pages} \rhd (\,\mathcal{I}(\text{“}\texttt{birnam}\text{”}) \;\lhd\; (\textsc{birnam} \lhd (\textsc{s} \rhd \textsc{witch})))$$

   where

   $$\begin{aligned} \textsc{pages} &\equiv \mathcal{I}(\text{“}\texttt{[page}\text{”}) \;\Diamond\; \mathcal{I}(\text{“}\texttt{page]}\text{”}) \\ \textsc{birnam} &\equiv (\,\mathcal{I}(\text{“}\texttt{[speech}\text{”}) \;\Diamond\; \mathcal{I}(\text{“}\texttt{speech]}\text{”})\,) \rhd \mathcal{I}(\text{“}\texttt{birnam}\text{”}) \\ \textsc{s} &\equiv \mathcal{I}(\text{“}\texttt{[speaker}\text{”}) \;\Diamond\; \mathcal{I}(\text{“}\texttt{speech]}\text{”}) \\ \textsc{witch} &\equiv (\,\mathcal{I}(\text{“}\texttt{[speaker}\text{”}) \;\Diamond\; \mathcal{I}(\text{“}\texttt{speaker]}\text{”})\,) \rhd \mathcal{I}(\text{“}\texttt{witch}\text{”}) \end{aligned}$$

   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.*

   $$\textsc{speeches} \rhd (\textsc{first2lines} \rhd (\textsc{t} \Diamond \textsc{b}))$$

   where

   $$\begin{aligned} \textsc{speeches} &\equiv \mathcal{I}(\text{“}\texttt{[speech}\text{”}) \;\Diamond\; \mathcal{I}(\text{“}\texttt{speech]}\text{”}) \\ \textsc{first2lines} &\equiv \mathcal{I}(\text{“}\texttt{[speech}\text{”}) \;\Diamond\; \textsc{lines} \;\Diamond\; \textsc{lines} \\ \textsc{t} &\equiv \textsc{lines} \rhd (\,\mathcal{I}(\text{“}\texttt{toil}\text{”}) \;\bigtriangledown\; \mathcal{I}(\text{“}\texttt{trouble}\text{”})\,) \\ \textsc{b} &\equiv \textsc{lines} \not\rhd (\,\mathcal{I}(\text{“}\texttt{burn}\text{”}) \;\bigtriangledown\; \mathcal{I}(\text{“}\texttt{bubble}\text{”})\,) \\ \textsc{lines} &\equiv \mathcal{I}(\text{“}\texttt{[line}\text{”}) \;\Diamond\; \mathcal{I}(\text{“}\texttt{line]}\text{”}) \end{aligned}$$

The expressions $\textsc{t}$ and $\textsc{b}$ select extents that match criteria on the contents of the lines. The expression $\textsc{first2lines}$ 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".*

$$((\textsc{fife} \lhd (\textsc{s} \rhd \textsc{apparition}))) \lhd (\textsc{scenes} \rhd \textsc{b})$$

where

$$\begin{array}{rcl}
\textsc{fife} & \equiv & \textsc{speeches} \rhd \mathcal{I}(\text{“}\texttt{fife}\text{”}) \\
\textsc{s} & \equiv & \mathcal{I}(\text{“}\texttt{[speaker}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{speech]}\text{”}) \\
\textsc{apparition} & \equiv & (\mathcal{I}(\text{“}\texttt{[speaker}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{speaker]}\text{”})) \rhd \mathcal{I}(\text{“}\texttt{apparition}\text{”}) \\
\textsc{b} & \equiv & \Sigma^5 \rhd \textsc{lines} \rhd \textsc{brdbry} \\
\textsc{speeches} & \equiv & \mathcal{I}(\text{“}\texttt{[speech}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{speech]}\text{”}) \\
\textsc{scenes} & \equiv & \mathcal{I}(\text{“}\texttt{[scene}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{scene]}\text{”}) \\
\textsc{lines} & \equiv & \mathcal{I}(\text{“}\texttt{[line}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{line]}\text{”}) \\
\textsc{brdbry} & \equiv & \mathcal{I}(\text{“}\texttt{something}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{wicked}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{this}\text{”}) \\
 & & \diamond\ \mathcal{I}(\text{“}\texttt{way}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{comes}\text{”})
\end{array}$$

This example illustrates the use of Σ^n. The expression $\textsc{b}$ 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, the following expressions must evaluate to the empty GC-list:

$$(\mathcal{I}(\text{“}\texttt{speaker]}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{[speaker}\text{”})) \not\rhd (\mathcal{I}(\text{“}\texttt{[speech}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{speech]}\text{”}))$$

$$(\mathcal{I}(\text{“}\texttt{speech]}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{[speech}\text{”})) \not\rhd (\mathcal{I}(\text{“}\texttt{[speaker}\text{”}) \diamond \mathcal{I}(\text{“}\texttt{speaker]}\text{”}))$$

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 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:

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

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

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

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:

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

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

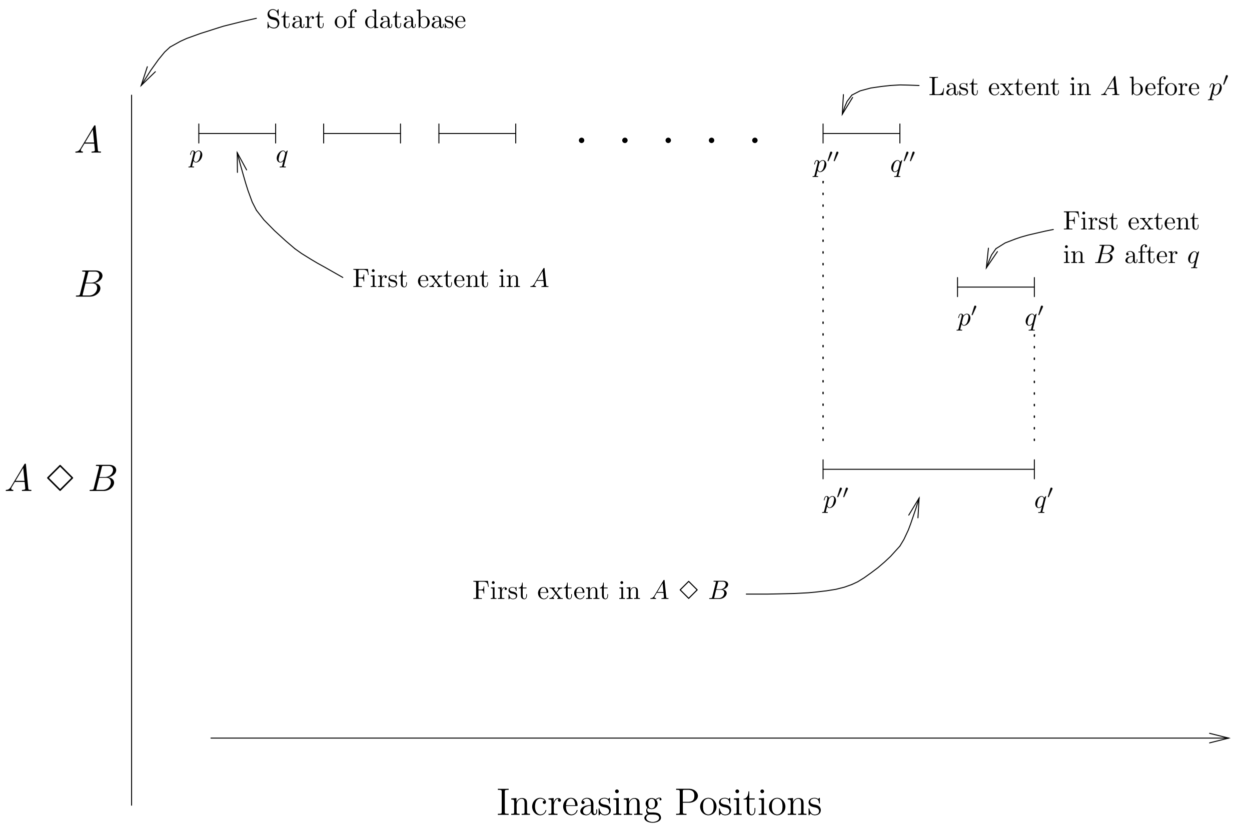


Figure 4: Evaluating $A \diamond B$.

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 [12]. 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 $(p', q') = \rho\ (B, q)$. There are now two cases: 1) If $p' \le p$ 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)
$p \ge p''$	since (p'', q'') contains (p, q)
$p'' \ge p'$	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 elementary terms is linear in the sum of the size of the GC-lists for the elementary terms. This observation ignores the effects of indexing into the GC-lists. These effects can be considerable. The evaluation of the expression in the first example on page 12 requires at most

$$O(\min(\ \mathcal{I}(\text{“}\texttt{[play}\text{”})\ ,\ \mathcal{I}(\text{“}\texttt{play]}\text{”})\ ,\ \mathcal{I}(\text{“}\texttt{birnam}\text{”})\ ,\ \mathcal{I}(\text{“}\texttt{dunsinane}\text{”})\ ))$$

Containment

```
$\tau\ (A \lhd B, k) =$
  let
    $(p, q) = \tau\ (A, k)$
    $(p', q') = \rho\ (B, q)$
  in
    if $p' \leq p$ then
      $(p, q)$
    else
      $\tau\ (A \lhd B, p')$
```

```
$\rho\ (A \lhd B, k) =$
  let
    $(p, q) = \rho\ (A, k)$
  in
    $\tau\ (A \lhd B, p)$
```

```
$\tau\ (A \rhd B, k) =$
  let
    $(p, q) = \tau\ (A, k)$
  in
    $\rho\ (A \rhd B, q)$
```

```
$\rho\ (A \rhd B, k) =$
  let
    $(p, q) = \rho\ (A, k)$
    $(p', q') = \tau\ (B, p)$
  in
    if $q' \leq q$ then
      $(p, q)$
    else
      $\rho\ (A \rhd B, q')$
```

```
$\tau\ (A \not\lhd B, k) =$
  let
    $(p, q) = \tau\ (A, k)$
    $(p', q') = \rho\ (B, q)$
  in
    if $p' > p$ then
      $(p, q)$
    else
      $\rho\ (A \not\lhd B, q' + \epsilon)$
```

```
$\rho\ (A \not\lhd B, k) =$
  let
    $(p, q) = \rho\ (A, k)$
  in
    $\tau\ (A \not\lhd B, p)$
```

```
$\tau\ (A \not\rhd B, k) =$
  let
    $(p, q) = \tau\ (A, k)$
  in
    $\rho\ (A \not\rhd B, q)$
```

```
$\rho\ (A \not\rhd B, k) =$
  let
    $(p, q) = \rho\ (A, k)$
    $(p', q') = \tau\ (B, p)$
  in
    if $q' > q$ then
      $(p, q)$
    else
      $\tau\ (A \not\rhd B, p' + \epsilon)$
```

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
    $(p, q) = \rho\ (A, k)$
    $(p', q') = \rho\ (B, k)$
  in
    if $q < q'$ then
      $(p, q)$
    else if $q > q'$ then
      $(p', q')$
    else
      $(\max(p, p'),\ q)$
```

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.

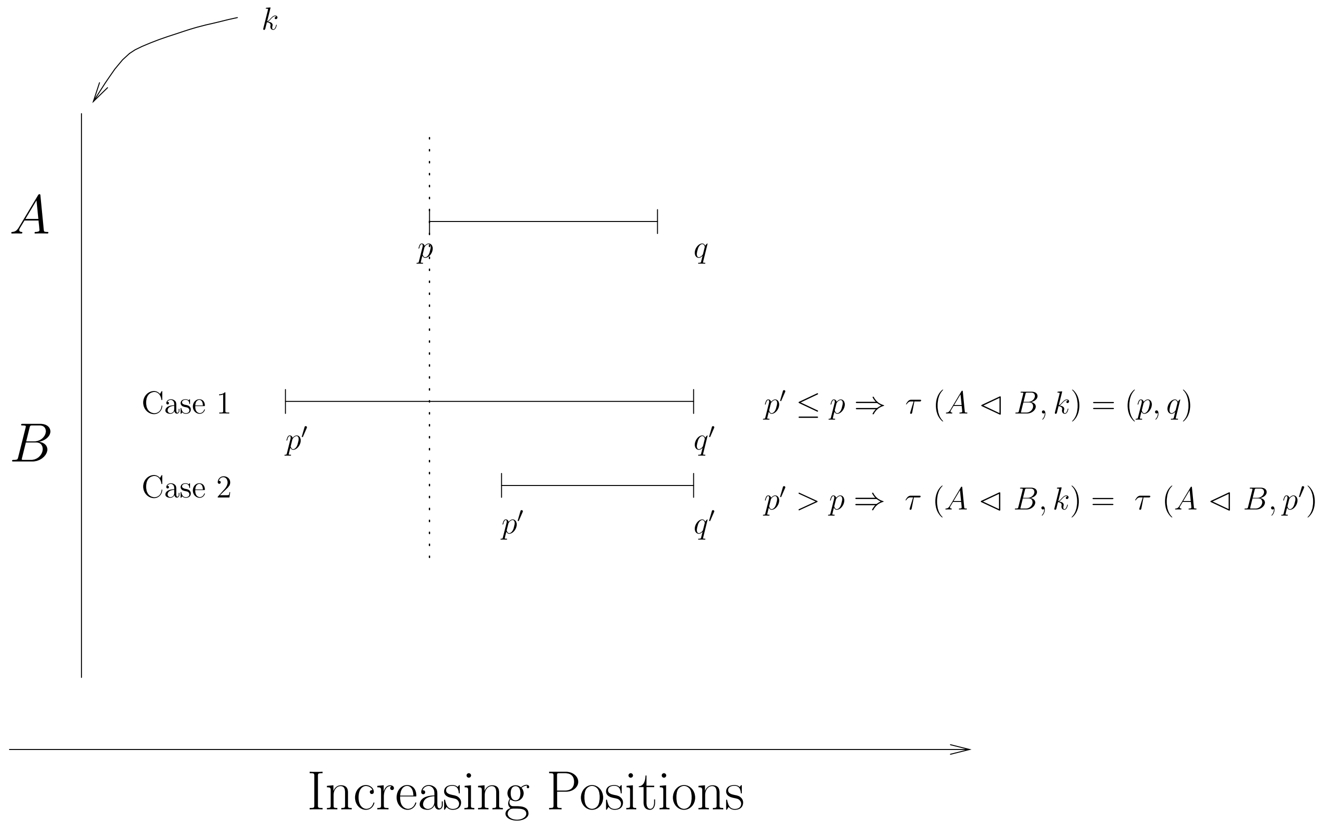

k
A
p
q
B
Case 1
p'
q'
$p' \leq p \Rightarrow \tau\ (A \lhd B, k) = (p, q)$
Case 2
p'
q'
$p' > p \Rightarrow \tau\ (A \lhd B, k) = \tau\ (A \lhd B, p')$
Increasing Positions

Figure 7: Evaluating $A \lhd B$.

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

Figure 8: Driver procedure

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:

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

Index organization

Standard data structures for inverted lists may be used to build implementations of τ and ρ for the database index [10, 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 [10, pages 473–479] or surrogate subsets [3], may be used to provide $O(\log n)$ implementations that additionally permit efficient insertions and deletions.

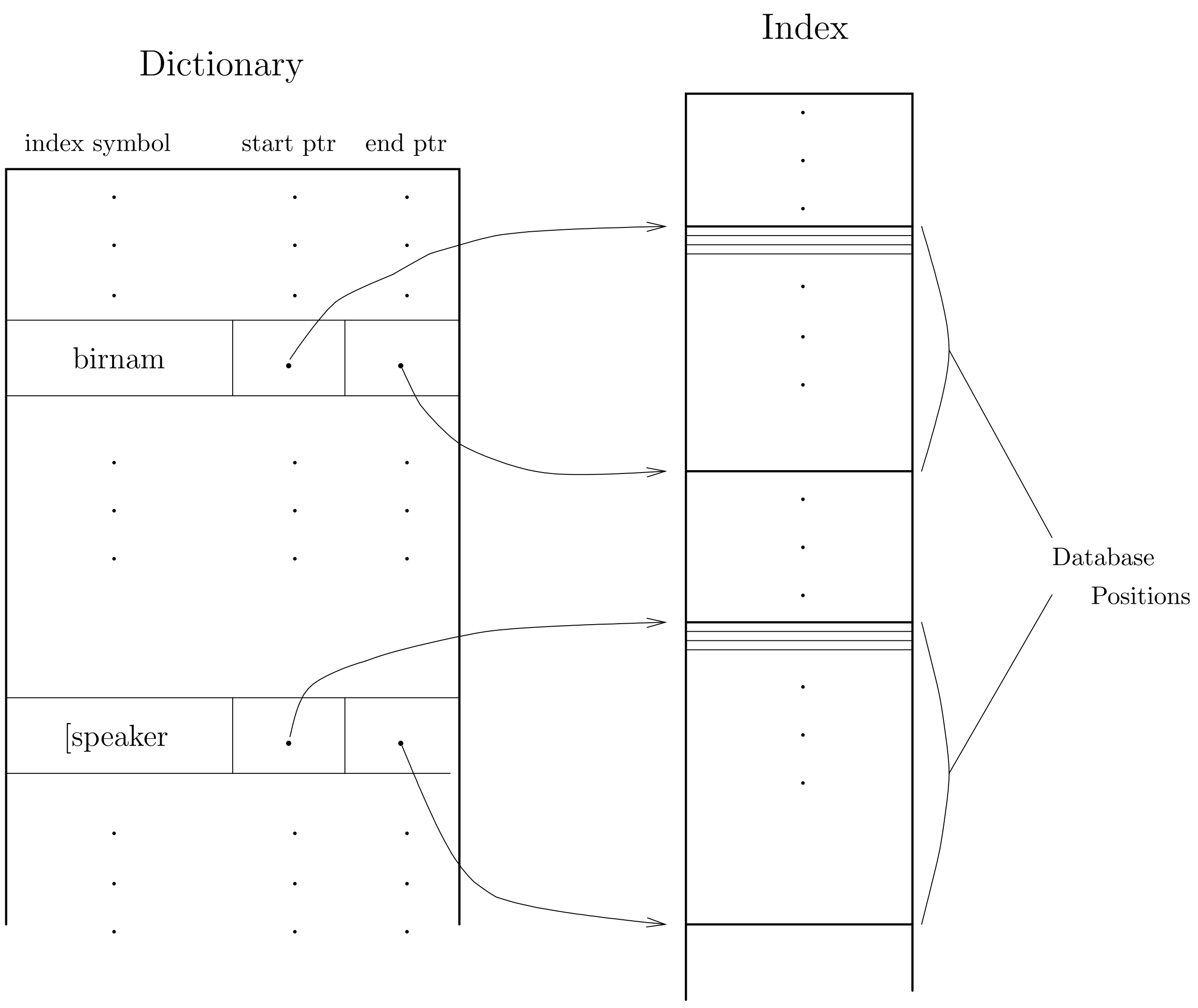


Figure 9: Inverted lists.

5. CONCLUSIONS

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. 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 the algebra. Ideally, we would be able to formulate queries that address text associated with an extent by way of one of these indirect mechanisms.

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. The SFQL standard [1] is an existing attempt to provide these capabilities as an extension to SQL. Finally, it is possible to view tables in a relational database as structured text and search them using our algebra.

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 ranking and other heuristic techniques might be incorporated into this user interface.

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 support of the Government of Ontario through the Information Technology Research Centre.

References

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

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

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

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

[5] 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.

[6] 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.

[7] 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.

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

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

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

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

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

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

[14] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*, chapter 2, pages 24–51. McGraw-Hill Computer Science Series. McGraw-Hill, New York, 1983.

[15] 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

$$m \begin{pmatrix} m \\ n-1 \end{pmatrix}.$$

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, efficient implementation.

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

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

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 $\bigtriangledown$:

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

```
$\rho\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k) =$
  let
    $(p, q) = \tau'\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k - \epsilon)$
  in
    $\tau\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ p + \epsilon)$
```

The implementation of $\tau\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k)$ 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 $\tau\ (n \bigtriangleup (A_0, A_1, ..., A_{m-1}),\ k)$.