Algebras for Querying Text Regions

(Extended Abstract)

Mariano P. Consens
Department of Computer Science
University of Waterloo
Waterloo, Canada N2L 3G1
mconsens@uwaterloo.ca

Tova Milo[*]
Department of Computer Science
Tel Aviv University
Tel Aviv, Israel 69978
milo@math.tau.ac.il

Abstract

There is a significant amount of interest in combining and extending database and information retrieval technologies to manage textual data. The challenge is becoming more relevant due to the increased availability of documents in digital form. Document data has a natural hierarchical structure, which may be made explicit due to the use of markup conventions (as it is the case with SGML). An important aspect of managing structured and semi-structured textual data consists of supporting the efficient retrieval of text components based both on their content and structure.

In this paper we study issues related to the expressive power and optimization of a class of algebras that support combining string (or pattern) searches with queries on the hierarchical structure of the text. The *region algebra* studied is a set-at-a-time algebra for manipulating *text regions* (substrings of the text) that supports finding out nesting and ordering properties of the text regions. The region algebra is part of the language in use in commercial text retrieval systems, and can be implemented very efficiently.

The results in this work are obtained by showing a close relationship between the region algebra and the monadic first order theory of binary trees. We show that queries in the algebra can be optimized, but the optimization can be difficult (Co-NP-Hard in the general case, although there is an important class of queries that can be optimized in polynomial time). On the negative side, we show that the language is incapable of capturing some important properties of the text structure, related to the nesting and ordering of text regions. We conclude by suggesting possible extensions to increase the expressive power of the language, focusing again on optimization and expressibility.

[*]Research done while the author was at the University of Toronto.

1 Introduction

Recently, there has been much interest in developing database tools for manipulating structured documents. Work in this area has studied high level languages for expressing queries and updates on files, and efficient execution engines for file manipulation [ACM93, BGMM93, CM94, GNOT92, Pae93, SLS$^+$93]. A key observation is that it is impractical to fully scan large documents while processing on-line queries. To provide reasonable response time, some of the data must be indexed. Text indexing systems usually provide one or two index types: word index recording location of words in the text, and region index recording location of various regions.

For example, consider a file containing the source code of a large program. The word index may record the location of certain keywords in the file (or even the location of all the words). The region index may record where each procedure of the program starts and ends, where in the program appear variable definitions, loop constructs, etc. The interfaces suggested for text indexing systems range from interfaces supporting only simple queries like "find where a given word w appears in the text", to powerful set-based algebraic languages, *ala* relational algebra[AFS93, ST92, Bur92, NBY95, CCB95].

Current research has mainly focused on the design of the interface language and on providing an efficient execution engine for it. There has been very little effort to characterize the capabilities of the resulting systems, and to answer questions like: what kind of structure-related information can be extracted by the system?; what kind of queries cannot be expressed?; is it possible to support more complex queries without harming the performance?

Our goal in this work is to find a suitable query interface for text indexing systems. We look for a language that on the one hand is powerful enough to exploit the structure embedded in the text, and on the other hand can be evaluated efficiently. By "efficient evaluation" we not only mean that the language belongs to a low worst-case complexity class, but also that it can have an efficient implementation. We concentrate on algebraic-based languages, and in particular on certain subsets of relational algebra. Clearly such languages cannot express some queries (e.g. parity[Ehr61]). We are, however, not very interested here in the expressibility of arbitrary queries. How many time does a user need to know the parity of the number of commands in a program? We are much more interested in the expressibility of queries with obvious practical use like "find the name of the procedure where the variable x was declared".

We start by studying an algebra that is the core of the PAT text retrieval system[Ope93]. We chose this algebra because it includes most of the operators considered in other proposals for index algebras. Also, the algebra has already been implemented in the PAT system, and is known to have a very efficient evaluation engine. Showing a close relationship between this algebra and the monadic first order theory of binary trees, we were able to study issues such as optimization and expressive power of the language. We show that queries in the algebra can be optimized, but the optimization can be difficult (Co-NP-Hard) in the general case. Nevertheless, there is an important class of queries that can be optimized in polynomial time. On the negative side, we show that the language is incapable of capturing some important properties of the text structure, related to the nesting and ordering of text regions. We then suggest some extensions to increase the expressive power of the language, focusing again on optimization and expressibility.

In Section 2 we present the algebra and the concept of a region inclusion graph. The relationship between the algebra and monadic first order theory of binary trees is studied in Section 3. Section 4 presents two key properties of the algebra, that are used in Section 5 to show that certain queries are not expressible. Extensions to the language are considered in Section 6. Finally, Section 7 presents the conclusions.

2 Text Regions

We start by presenting a modified version of the the PAT algebra [Gon87, ST92]. Algebras with a similar approach in terms of recognizing and manipulating text structure have been studied recently in [Bur92, NBY95, CCB95]. Following the introduction of the algebra we discuss an important kind of constraint that is imposed by the nature of the format of the text that is indexed.

2.1 The Region Algebra

PAT is a set-at-a-time algebra for text queries, that has a very efficient evaluation engine. There are two types of sets in the algebra: sets of match points and sets of regions. The match points correspond to the positions in the text of indexed strings (the entries of the word index), while each region is a substring of the indexed text, and is defined by a pair of positions in the text corresponding to the beginning and the end of the region.

Different text indexing systems support different kinds of word index. Some systems only enable the user to search for specific words, while other more sophisticated systems allow searching for strings having patterns specified in some pattern language, (e.g., they use don't care symbols, regular expressions, etc.). To treat these indexes uniformly, we make no assumptions about the specific pattern language being used, and represent a word index by a binary predicate W, such that $W(r,p)$ holds for a region r and a pattern p, iff according to the word index, the text stored in the region r contains the pattern specified by p. We do assume, however, that we are given a specific set of named regions on the indexed text. This assumptions are captured in the definition below.

Definition 2.1 A **region index** $\mathcal{I}$ is a set of region names $R_1, \ldots, R_n$, together with a word index W. An **instance** of a region name R_i is a set of regions in a file. An **instance** I of a region index $\mathcal{I}$ is a mapping associating an instance

$R_i(I)$ to each region name R_i, and a mapping associating a boolean value to $W(r,p)$, for every region $r \in R_i(I)$, $1 \leq i \leq n$, and pattern p in the pattern language.

As a notational convenience when I is understood from the context, we use R_i for both the region name and the instance $R_i(I)$.

Many researchers have concentrated on files with hierarchical structure [ACM93, Bur92, CACS94, GT87, GZC89]. Indeed, many text databases (e.g., programs, news, patents, reports, SGML documents in general) have a structure described by a grammar defining a hierarchy of nested regions. Following this approach we assume below that the region index defines a hierarchy of nested regions, where every region r belongs to only one region set R_i, and every two regions are either disjoint, or one is strictly included in the other.

Note that a region index can be viewed as a relational database (with one relations per region type, and one relation for the word index), and that relational algebra can be used to express queries on the index. By restricting the join capabilities of the algebra, and the use of the word index, one obtains a restricted algebra that can be implemented very efficiently. The PAT algebra is an example of such a restricted algebra that is used by a commercial system [Ope93]. To simplify the presentation, (and highlight the aspects of the PAT algebra that are of interest to us), we describe below a modified subset of the algebra[1], that concentrates on the manipulation of sets of regions. We call this algebra the **region algebra**.

Definition 2.2 Region algebra *expressions* over $\mathcal{I}$ are expressions generated by the grammar

$$e \rightarrow R_i \mid e \cup e \mid e \cap e \mid e - e \mid$$

$$e \supset e \mid e \subset e \mid e < e \mid e > e \mid \sigma_p(e) \mid (e)$$

where the R_i's are the region names in $\mathcal{I}$.

The semantics of the algebra are described in the definition below. We use the notation $left(r)$ (resp. $right(r)$) to denote the location of the left (resp. right) endpoint of a region r. Similarly, we use $r \supset s$, where r,s are two regions, to denote the fact that the region r strictly includes the region s (i.e., $left(r) < left(s)$ and $right(r) \geq right(s)$, or $left(r) \leq left(s)$ and $right(r) > right(s)$). Finally, we use $r < s$, to denote the fact that the region r precedes the region s (i.e., $right(r) < left(s)$).

[1]Note that the full PAT algebra is capable of constructing sets of regions dynamically. From the point of view of this work we can treat regions defined dynamically as if they were *views*. On the other hand, we consider nested region sets, which are not supported by PAT.

Definition 2.3 Union ($\cup$), intersection ($\cap$)), and difference ($-$) are the usual set theoretic operations on sets of regions. The *including* ($\supset$) and *included* ($\subset$) operations take two sets of regions R and S and return the sets

$$R \supset S = \{r \in R : \exists s \in S, r \supset s\}$$

$$R \subset S = \{r \in R : \exists s \in S, r \subset s\}$$

The *follows* ($>$) and *precedes* ($<$) operations take two sets of regions R and S and return the sets

$$R > S = \{r \in R : \exists s \in S, r > s\}$$

$$R < S = \{r \in R : \exists s \in S, r < s\}$$

Finally, the *selection* (σ_p) operation takes a pattern p and a set of regions R and returns the regions $r \in R$ s.t. $W(r,p)$ is true.

For an expression e, and an instance I we use $e(I)$ to denote the result of e when evaluated on I. Note that $\supset$, $\subset$, $<$, $>$ are not associative. For brevity, we omit parentheses and assume that the operations are grouped from the right.

2.2 The Region Inclusion Graph (RIG)

Observe that files of a specific format have specific inclusion relationships among regions. For instance, consider a file containing source code of programs. Assume that each program has a header including the program name, and a body containing definition of variables and procedures. Each procedure has header including its name, and a body that may define more variables and contain definitions of other procedures. Let $\mathcal{I} = \{Prog, Prog\_header, Prog\_body, Proc, Proc\_header, Proc\_body, Name, Var\}$, be the region index. Consider the two expressions

$e_1 = Name \subset Proc\_header \subset Proc \subset Program$

$e_2 = Name \subset Proc\_header \subset Program$

These two queries do not necessarily have the same result for arbitrary instances of $\mathcal{I}$. But if only instances describing programs of the above structure are considered, then the two expressions do have the same result: they both retrieve the names of all procedures. This is because in programs all the *Proc_header* regions are included in some *Proc* region. Thus the test for inclusion in *Proc* can be omitted. Note that we cannot further omit the test for inclusion in *Proc_header*, since we need to distinguish between names of programs and names of procedures. The key observation is that the second expression has less operations than the first, and can be evaluated more efficiently. In general, we would like to use the knowledge about the structure of files when analyzing queries, in particular to rewrite queries so that they can be evaluated more efficiently.

To describe the relationships between regions, we introduced in [CM94] a **region inclusion graph** (*RIG*, for short). The nodes of the graph are region names, and the edges state the possible inclusion relationships between the corresponding region instances. For an instance I and two regions $r, s \in I$, we say that r **directly includes** s in I, if $r \supset s$ and there is no other region $t \in I$ s.t. $r \supset t$ and $t \supset s$. An edge (R_i, R_j) is in the RIG, iff an R_i region can directly include an R_j region. In general, the RIG may contain cycles (e.g., self-nested regions). The graph is used to characterize a set of instances that obey certain inclusion restrictions.

Definition 2.4 An instance I of a region index $\mathcal{I} = \{R_1, \ldots, R_n\}$ *satisfies* a RIG (region inclusion graph) $G = (\mathcal{I}, E)$ iff for every two regions $r_i \in R_i(I), r_j \in R_j(I)$, if r_i directly includes r_j then $(R_i, R_j) \in E$. The set of all instances of $\mathcal{I}$ that satisfy a RIG G is denoted $\mathcal{I}_G$.

We next consider equivalence of region expressions. In the standard database approach, two queries over a given schema are equivalent iff they have the same result for every instance of the database. In the context of queries in the region algebra, a RIG can be viewed as schema. We therefore have the following definition.

Definition 2.5 Two region expressions e_1, e_2 are **equivalent** with respect to a RIG $G = (\mathcal{I}, E)$ iff for every instance $I \in \mathcal{I}_G$, $e_1(I) = e_2(I)$.

For example, the program files mentioned above are described by the RIG shown in Figure 1. The queries e_1 and e_2 above are equivalent with respect to that RIG.

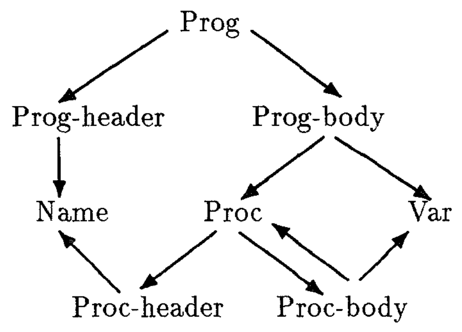

Figure 1: A RIG for source code regions.

Note that if the structure of the file follows some grammar G (where G could be a context free grammar), then the RIG can be automatically derived from G. The nodes are the non-terminals of G, and the graph has an edge (A_i, A_j) iff G has a rule where A_i appears as the left side, and A_j as the right side.

A similar approach can be used to take into account the relative order of regions, and characterize instances obeying certain order restrictions. We can define a **region order graph** (*ROG*) that describe the possible direct precedence relationships among regions. The nodes of the graph are region names, and the edges state the possible precedence relationships between the corresponding region instances. For an instance I and two regions $r, s \in I$, we say that r **directly precedes** s in I, if $r < s$ and there is no other region $t \in I$ s.t. $r < t$ and $t < s$. An edge (R_i, R_j) is in the ROG, iff an R_i region can directly precede an R_j region. As it was the case with RIG's, a ROG can also be derived from a grammar.

3 Relationship with Monadic Tree Theory

The region algebra is closely related to the first order monadic theory of finite binary trees (FMFT, for short) [Tho90, CH90, IK89]. This relationship helps to identify some of the properties of the language.

To simplify the presentation we use here a variant of the theory, described in what follows. Models of the theory have the form

$$t = (\{0,1\}^*, \supset, <, Q_1, \ldots, Q_n)$$

where $\supset$ is the proper prefix order relation over $\{0,1\}^*$, $<$ is the lexicographical order relation over $\{0,1\}^*$, and $Q_1, \ldots, Q_n$ are finite subsets of $\{0,1\}^*$ [2]. For a model t, we use the term the *words* in t to refer to the set of binary strings in $\bigcup_{1 \leq i \leq n} Q_i$.

FMFT formulas are constructed using variables, words over $\{0,1\}^*$, atomic formulas of the form $x = y$, $x \supset y$, $x < y$, $Q_i(x)$, and using connectives and quantifiers. The semantics of formulas is defined in the standard way (see [Tho90] for more details). We use the notation $\phi(t)$ to denote the result of the formula ϕ when evaluated w.r.t to a model t. We are especially interested here in a specific class of formulas, that we call **restricted FMFT formulas**.

Definition 3.1 Restricted FMFT formulas are FMFT formulas with one free variable, and the structure defined below:
(1) formulas $Q_i(x)$ are restricted formulas,
(2) if ϕ_1, ϕ_2 are restricted formulas, then the following are restricted formulas as well:
(i) $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\phi_1 \wedge \neg\phi_2$, where ϕ_1, ϕ_2 have the same free variable,
(ii) $(\exists y)\phi_1 \wedge \phi_2 \wedge x \circ y$, and $(\exists y)\phi_1 \wedge \phi_2 \wedge y \circ x$, where $\circ$ is one of $\supset$,$<$, and ϕ_1,ϕ_2 have two distinct free variables x, y resp.

We next study the relationship between queries in the region algebra and restricted formulas. For a model t and two words u, v in t, we say that u is a *direct prefix* of v in t if there is no other word w in t s.t. u is a prefix of w and w is a prefix of v. Observe that a model t can be viewed as an ordered forest (not necessarily binary) where the nodes are the words in t, a word u is a parent of a word v iff u is a direct prefix of v, and a word u precedes v in the forest iff u precedes v in the lexicographical order. Also observe that the operators in the region algebra test the relative location of regions, but the exact position of region endpoints is not explicitly used. A forest representation of instances, recording inclusion and order relationships, but ignoring the exact position of regions in the file, is therefore very convenient for reasoning about the properties of the algebra. The relationship between models and region instances is captured by the next definition.

Definition 3.2 Let $\mathcal{I} = \{R_1, \ldots, R_n\}$ be a region index. Let $P = \{p_1, \ldots, p_k\}$ be a set of patterns. Let $t = (\{0,1\}^*, \supset, <, Q_1, \ldots, Q_{n+k})$ be a model. We say that t **represents** an instance I of $\mathcal{I}$ w.r.t. P iff all the Q_i, $1 \leq i \leq n$, are disjoint, all the words in Q_j, $n < j \leq n+k$ appear in some Q'_i, $1 \leq i' \leq n$, and there is a 1-1 mapping (denoted $region_I$) from the words in t to the regions in I, such that
(1) a word u in t is a direct prefix of a word v in t iff $region_I(u)$ directly includes $region_I(v)$ in I,
(2) a word u in t precedes a word v in t (that does not have u as a prefix) iff $region_I(u) < region_I(v)$,
(3) a word u in t belongs to Q_i, for $1 \leq i \leq n$ iff $region_I(u) \in R_i$,
(4) a word u in t belongs to Q_{n+j}, $1 \leq j \leq k$ iff $W(region_I(u), p_j)$ is true.

The next proposition demonstrates the relationship between algebra queries and restricted formulas. It states that the algebra and the restricted FMFT formulas express the same queries on regions.

Proposition 3.3 *For every region algebra expression e using patterns in P, there exists a restricted FMFT formula ϕ, such that for every instance I, every model t representing I w.r.t. P, and every word w in t, $region_I(w) \in e(I)$ iff $w \in \phi(t)$. Conversely, for every restricted formula ϕ there exists an algebra query e using patterns in P, such that for every every instance I, every model t representing I w.r.t. P, and and every word w in t, $region_I(w) \in e(I)$ iff $w \in \phi(t)$.*

Proof: The proof follows the lines of the classical algebra-calculus equivalence proofs [Ull88] and works by induction on the structure of queries. We

[2]The literature often uses $<$ instead of $\supset$ to denote the prefix order, and $\preceq$ instead of $<$ to denote lexicographical order. We chose the above notation to make the correspondence to the region algebra straightforward. FMFT is a subset of the more familiar second order theory of two successors (S2S) [Tho90, Rab69] with models of the form $t = (\{0,1\}^*, succ_0, succ_1, \supset, Q_1, \ldots, Q_n)$, where $succ_0, succ_1$ are two successor functions. The lexicographical order ($<$) can be defined in terms of $succ_0$ and $succ_1$.

outline here the construction to obtain a restricted formula from a region algebra expression (the converse is completely analogous).

As the basis for induction, an algebraic expression of the form R_i is translated to the formula $\{x|Q_i(x)\}$. Now, if we are given two algebraic expressions e_1 and e_2, (by the inductive hypothesis) we can obtain their corresponding restricted formulas ϕ_1 and ϕ_2 (and we can further assume that the only free variable in each of ϕ_1 and ϕ_2 has been renamed to x and y, resp.). The translation for the expressions $e_1 \cup e_2$, $e_1 \cap e_2$, $e_1 - e_2$ is $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\phi_1 \wedge \neg\phi_2$, respectively. The translation for the expressions $e_1 \supset e_2$, $e_1 < e_2$, is $(\exists y)\phi_1(x) \wedge \phi_2(y) \wedge x \circ y$, where $\circ$ is $\supset$,$<$, respectively. The translation for the expressions $e_1 \subset e_2$, $e_1 > e_2$ is $(\exists y)\phi_1(x) \wedge \phi_2(y) \wedge y \circ x$, where $\circ$ is $\supset$,$<$, respectively. Finally, the translation for $\sigma_{p_i}(e_1)$ is $\phi_1 \wedge Q_{n+i}(x)$. □

Consider a region index $\mathcal{I} = \{R_1, \ldots, R_n\}$, and a set of patterns P. It is easy to see that every hierarchical instance I has many models t that represent it w.r.t. P. On the other hand, every model $t = (\{0,1\}^*, \supset, <, Q_1, \ldots, Q_{n+k})$ where (i) all the sets Q_i, $1 \leq i \leq n$, are disjoint, and (ii) all the words $u \in Q_{n+j}$, $1 \leq j \leq k$, belong to some Q_i, $1 \leq i \leq n$, represents some region instance. We have that

Theorem 3.4 *For every region algebra expression e, there exists a restricted FMFT formula ϕ s.t. $e(I)$ is empty for all instances I iff ϕ is unsatisfiable.*

The proof follows from Proposition 3.3, and from the fact that restriction conditions (i) and (ii) above can be expressed by a restricted formula.

Satisfiability of FMFT formulas is decidable [Rab69], and thus testing if $e(I)$ is empty for every instance I is decidable. We call this the *emptiness testing* problem. Emptiness testing can be used to optimize queries. Suppose we have a price function p estimating the expected cost of an algebra expression. Assume also that every operation adds some cost to the price of an expression. To optimize an expression e we can look for an equivalent expression with lowest price (because of the assumptions we need to check only a finite number of expressions). Two expressions e_1, e_2 are equivalent iff $(e_1 - e_2) \cup (e_2 - e_1)$ is empty for all instances.

The algorithm for testing satisfiability of FMFT formulas has non elementary complexity [Rab69, CH90], thus the above optimization technique is very expensive. Note that in our case the algebra queries correspond to a very limited class of FMFT formulas (restricted formulas). The following theorems shows that, even in this case, emptiness testing and optimization cannot be done in polynomial time (unless P=NP).

Theorem 3.5 *Emptiness testing in the region algebra is Co-NP-Hard.*

The proof is by reduction from the problem of checking if a 3-CNF formula is unsatisfiable, known to be Co-NP-complete. There are, however, some important practical cases where emptiness testing and optimization can be done in polynomial time. We present such cases in Section 5.1.

As mentioned in Section 2.2, given a RIG G, we are not interested in arbitrary region instances, but only in instances that satisfy G. We can refine theorem 3.4 and show that

Theorem 3.6 *Let G be some RIG. For every region algebra expression e, there exists an FMFT formula ϕ s.t. $e(I)$ is empty for all instances I satisfying G iff ϕ is unsatisfiable.*

The proof is based on the fact that the direct inclusion restrictions imposed by the RIG can be expressed by FMFT formulas. The refined theorem can now be used to optimize queries w.r.t a RIG. Note that the theorem uses **general** FMFT formulas and not restricted formulas. This is because (as we show later in Section 5.1) direct inclusion cannot be expressed by restricted formulas.

4 Properties of the Region Algebra

We now present two important properties of the algebra, that are used to prove the results of the following section. The key observation is that, under certain conditions, queries are not affected by the deletion of elements from the input.

4.1 Deletion

Let I be some region instance, and let S be a set of regions in I. We say that an instance I' is an S-**deleted** version of I, if it was obtained from I by deleting some regions, but leaving all the regions in S. The next theorem shows that a careful selection of S can guarantee that deletion from the input does not affect the output of queries.

Theorem 4.1 *Let e be an algebra expression. For every instance I, there exists a set of regions $S \subset I$ with region nesting at most $2|e|$, s.t. for every S-deleted version I' of I, the following hold*
(1) $e(I) = \emptyset$ iff $e(I') = \emptyset$.
(2) for every region r that belongs to both to I and I', $r \in e(I)$ iff $r \in e(I')$.

The proof works by induction on the number of operations in e, and constructively builds the desired S.

4.2 Reduction

We next consider a more refined deletion of regions that preserves some containment, ordering and word indexing properties of the original input. For that, we use an auxiliary notion of *region isomorphism*, and define a **reduce** operation. For an instance I and a region $r \in I$, we use the notation S_r to denote the set of regions containing r, and all the regions in I that are included in r.

Definition 4.2 Let I be in instance, and let P be a set of patterns. We say that two regions $r_1, r_2 \in I$ are *isomorphic* w.r.t. P, if there is a 1-1 mapping τ from S_{r_1} to S_{r_2} s.t. (i) τ preserves the inclusion and precedence relationship of regions, and (ii) for every region $r \in S_{r_1}$, every region name R_i, and and every pattern $p \in P$, $r \in R_i$ iff $\tau(r) \in R_i$ and $W(r,p)$ holds iff $W(\tau(r),p)$ holds. The operation *reduce*(I, r_1, r_2) tests if r_1,r_2 are isomorphic, and if so deletes from I all the regions in S_{r_1}.

Observe that the reduction can be used to define a mapping h from the regions of I to the regions of I'. First consider a single reduction step. Let r be some region in I. A mapping h can be defined as: (i) if $r \in I'$, then $h(r) = r$, and (ii) if $r \notin I'$, and was deleted due to an isomorphic mapping τ, then $h(r) = \tau(r)$. Now, let I' be an instance obtained from I by a sequence of reduce operations. The mapping h defined by a sequence of reductions is simply the composition of the mappings defined by each individual reduction. We use the mapping to define a refined notion of reduction that preserves certain order relationships between regions.

Definition 4.3 Let P be a set of patterns. An instance I' is a **0-reduced** version of I w.r.t. P, if it was obtained from I by a sequence of reduce operations. An instance I' is a **k-reduced** $(k > 0)$ version of I w.r.t. P iff,
(1) it was obtained from I by a sequence of reductions (defining a mapping h_k from I to I'), and
(2) it has a (k-1)-reduced version I'' (with some mapping h_{k-1} from I to I''), s.t. for every two regions $r, s \in I$, $r < s$ in I iff there exists $t \in h_{k-1}^{-1}(h_{k-1}(h_k(s)))$ s.t. $h_k(r) < t$ in I'.

The next theorem shows that if enough order related information is preserved while reducing the size of the instance, the modifications to the input do not effect the behaviour of a query.

Theorem 4.4 *Let e be some algebra expression, P the set of patterns in e, and k the number of $<$ and $>$ operations in e. For every instance I, and every k-reduced version I' of I w.r.t P, the following hold*
(1) $e(I) = \emptyset$ iff $e(I') = \emptyset$.
(2) for every region r that belongs both to I and I', $r \in e(I)$ iff $r \in e(I')$

The above result is proved using the following proposition.

Proposition 4.5 *Let e be some algebra expression, P the set of patterns in e, k the number of $<$ and $>$ in e, I some instance, I' a k-reduced version of I w.r.t P, and h the mapping defined by the reduction. For every region r in I, $r \in e(I)$ iff $h(r) \in e(I')$*

Proof: The proposition is proved by induction on the number of operations in e. It is used to prove Theorem 4.4 as follows. Part (2) follows immediately from Proposition 4.5, and the fact that if r belongs to I' then $h(r) = r$. For part (1), observe that $e(I) \neq \emptyset$ iff there exists some region $r \in I$ s.t. $r \in e(I)$. But from Proposition 4.5 this happens iff $h(r) \in e(I')$, thus iff $e(I') \neq \emptyset$. □

5 Expressiveness

We next use the deletion and reduction theorems to study the limitations of the region algebra. In particular we show its inability to capture a refined notion of inclusion of region, called *direct inclusion*, and its incapability to mutually capture inclusion and order relationships. We motivate the practical importance of the kinds of queries considered in their respective subsections.

5.1 Direct Inclusion

Consider the file containing source code of a program, mentioned above. Assume that we want to find the procedures that define a specific variable, say x. Note that looking for *Proc* regions that contain a *Var* region defining x, which can be expressed as $Proc \supset Proc\_body \supset \sigma_{``x"}(Var)$, will not generate the required result. This is because procedures can be nested. A procedure may be selected not because it defines x, but because it contains another procedure defining x. What we want is to select *Proc* regions that *directly* include (i.e., no other region resides in between) a *Proc_body* region, also *directly* including the required *Var* region, This notion of direct inclusion is captured by the operators *directly including* ($\supset_d$) and *directly included* ($\subset_d$) defined below:

$$R \supset_d S = \{r \in R : \exists s \in S, r \supset s \wedge \neg \exists t \in I, r \supset t \wedge t \supset s\}$$

$$R \subset_d S = \{r \in R : \exists s \in S, s \supset r \wedge \neg \exists t \in I, s \supset t \wedge t \supset r\}$$

Now we can express the query to find *Proc* regions that directly contain a *Var* region defining x as $Proc \supset_d Proc\_body \supset_d \sigma_{``x"}(Var)$.

We show next that direct inclusion cannot be expressed by the region algebra.

Theorem 5.1 *The direct inclusion and directly included operations cannot be expressed by the region algebra.*

Proof: We first consider the directly includes operation. Let $\mathcal{I} = \{A, B\}$ be a region index, with a RIG containing the edges (A, B) and (B, A). Assume there is an algebra expression e computing $B \supset_d A$. Consider the instance I with region nesting depth $4|e|$, and with the structure shown in Figure 2.

From Theorem 4.1, it follows that there exists a set of regions $S \subset I$ with nesting depth at most $2|e|$ (where $|e|$ is the number of operations in e), such that for every S-deleted version I' of I, and every region r appearing both in I and I', $r \in e(I)$ iff $r \in e(I')$. Since the region nesting in S is at most $2|e|$, for every S, I contains two B regions r and r', where r directly includes r', and both r and r' are not in S. Consider the S-deleted version I' of I obtained by deleting r'. From the theorem $r \in e(I)$ iff $r \in e(I')$. But if e computes $B \supset_d A$, then $r \notin e(I)$, while $r \in e(I')$, which yields a contradiction. The proof for $B \subset_d A$ is similar. □

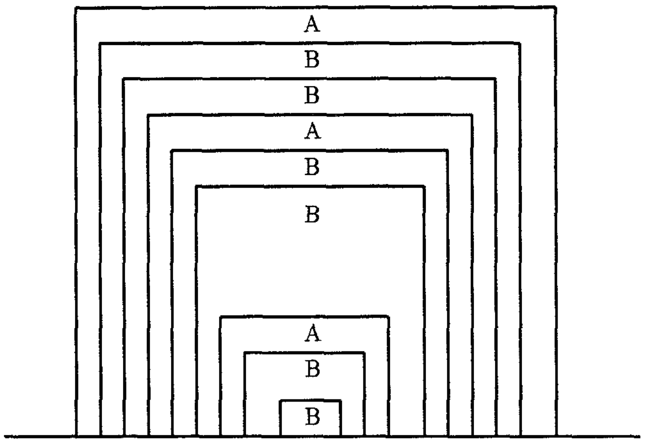

Figure 2: Counter-example for direct inclusion.

The above result is based on the fact that regions can be nested to an arbitrary depth. If the nesting depth is guaranteed to be bounded by some constant, direct inclusion is expressible. Note that files with an acyclic RIG have nesting depth bounded by the length of the longest path in the RIG.

Proposition 5.2 *Direct inclusion is expressible for files satisfying an acyclic RIG.*

Proof: We sketch the proof for $\supset_d$ ($\subset_d$ is completely analogous). If a set of regions Q is not nested then $Q \supset_d R = Q \supset (R - (R \subset (\bigcup_{T \in \mathcal{I}} T) \subset Q))$, for every region set R. For Q with nesting bounded by k, we use k algebra expressions to select regions in the i'th layer, $1 \leq i \leq k$, then we compute direct inclusion for each layer (using the above expression), and take the union of the results. □

An implementation of direct inclusion for files with unbounded nesting is presented in Section 6. We next consider a restricted class of expressions

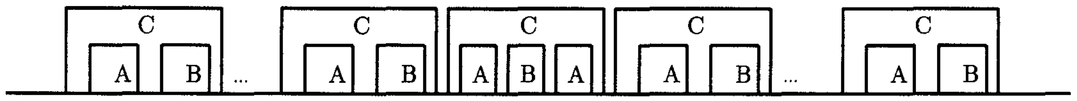

Figure 3: Counter-example for both included.

called *inclusion expressions.* These are expression that use only the $\supset$ and $\supset_d$ operations, or only $\subset$ and $\subset_d$ operations. It was shown in [CM94] that this class, although seems very restricted, is very useful for computing high level object oriented queries on files. Thus the ability to optimize such queries is crucial. We showed in the previous section that query optimization is very expensive in general. Fortunately, it turns out that inclusion expressions can be optimized in polynomial time [CM94].

5.2 Both-Included

We have showed above that the region algebra is incapable of capturing properties related to the nesting of regions. We show next that the algebra is also incapable of capturing order related properties.

Consider again the file containing the source code of a program. Assume that we want to find the procedures containing definitions of two variables, say x and y, where the definition for x precedes that of y. One may try to use the expression $Proc \supset (\sigma_{``x"}(Var) < \sigma_{``y"}(Var))$. This however does not compute the required result. Some of the procedures may be selected because they contain an x variable that precedes some definition of y, but where the y definition is in another procedure. Unfortunately, it turns out that queries as the above, involving both inclusion and order testing, cannot be expressed by the algebra.

It is important to highlight that this type of query is the most common kind of request for traditional document-based text retrieval systems. Traditional systems recognize one distinguished unit (the document) within the structure of the text being indexed. In our source code example the unit could be each procedure (this will be particularly appropriate if procedures are kept in individual files). Then, the system will support a request to retrieve variable definitions for x preceding those for y, and only consider those pairs x and y that occur in the same procedure.

Let's define the operation *both-included*, that takes three sets of regions R, S and T and returns the set

R BI $(S,T) = \{r \in R : \exists s \in S, \exists t \in T, r \supset s \wedge r \supset t \wedge s < t\}$

The expression $Proc$ BI $(\sigma_{``x"}(Var), \sigma_{``y"}(Var))$ computes exactly the above query. The result below characterizes another limitation of the region algebra.

Theorem 5.3 *The operation both-included cannot be expressed by the region algebra.*

Proof: Let $\mathcal{I} = \{A, B, C\}$ be a region index, with a RIG containing the edges (C, A) and (C, B). Assume there is an algebra expression e computing C BI (B, A). Let P be the set of patterns appearing in e and k the number of $<$ and $>$ in e. Consider the instance I, containing $4k+1$ C regions, with the structure shown in Figure 3.

Let $r_1, \ldots, r_{4k+1}$ be the regions corresponding to the C regions, where r_{2k+1} is the C region containing two A regions. Let r'_{2k+1} be the first A region included in r_{2k+1}, and let r''_{2k+1} be the second. Consider the instance $I' = reduce(I, r'_{2k+1}, r''_{2k+1})$. The regions r'_{2k+1}, r''_{2k+1} are isomorphic w.r.t. P. Thus I' contains all the regions of I except r''_{2k+1}. If we can show that I' is a k-reduced version on I, then from Theorem 4.4, it follows that $r_{2k+1} \in e(I)$ iff $r_{2k+1} \in e(I')$. But, since e computes C BI (B, A) then $r_{2k+1} \in e(I)$, while $r_{2k+1} \notin e(I')$, which yields a contradiction.

Thus all we have to show is that I' is a k-reduced version on I. Clearly, I' is a reduced version on I. Let h_k be the mapping defined by the reduce operation $reduce(I, r'_{2k+1}, r''_{2k+1})$. We have to show that I' has a $(k-1)$-reduced version I'' (with some mapping h_{k-1}), such that for every two regions $r, s \in I$, r follows s in I iff there exists $t \in h^{-1}_{k-1}(h_{k-1}(h_k(s)))$ s.t. t follows $h_k(s)$

in I'. It is easy to show that the instance $I'' = reduce(I', r_{2k+1}, r_{2k+2})$ is a $(k-1)$ reduced version of I' satisfying the above requirement. □

The preceding inexpressibility result relies on the fact that files can have arbitrary number of non overlapping regions. If the number is guaranteed to be bounded by some constant k, both-included becomes expressible. Note that for files satisfying an acyclic ROG (region order graph) this number is bounded by the length of the longest path in the ROG.

Proposition 5.4 *Both-included is expressible for files where the number of non overlapping regions is bounded by some constant k.*

The proof is similar to the case of direct inclusion covered in the previous subsection. We next show that direct inclusion and both-included are independent operators.

Proposition 5.5 *Direct inclusion cannot be expressed by the region algebra augmented with both-included. Similarly, both-included cannot be expressed by the region algebra augmented with direct inclusion.*

Proof: This follows from the fact that if only direct inclusion is added then the reduction theorem (4.4) still holds for the extended language. On the other hand, if only *both–included* is added, then the deletion theorem (4.1) still holds for the extended language. Both proofs are similar to the original ones, with an additional case in the induction step. □

Both operations can be expressed by FMFT formulas. Thus theorem 3.6 holds for the algebra augmented with direct inclusion and both-included, and queries in the extended language can be optimized.

6 Extending the Region Algebra

We show below that if the algebra is embedded in a programming language with a while construct and assignments then the operations can be supported. In particular we consider the execution cost and optimization. For example, $R_1 \supset_d R_2$ can be expressed as follows (a similar program can be used to compute $R_1 \subset_d R_2$):

```
$R_1^{layer} := R_1 - (R_1 \subset R_1)$;
$R_1^{rest} := R_1 - R_1^{layer}$;
$R_1^{result} := \emptyset$;
$All := \bigcup_{T \in \mathcal{I}} T$;
while $(R_1^{layer} \supset R_2) \neq \emptyset$ do
    $R_1^{result} := R_1^{result} \cup (R_1^{layer} \supset$
            $(R_2 - (R_2 \subset All \subset R_1^{layer})))$;
    $R_1^{layer} := R_1^{rest} - (R_1^{rest} \subset R_1^{rest})$
    $R_1^{rest} := R_1^{rest} - R_1^{layer}$;
end
return $R_1^{result}$
```

In many practical applications, one may need to compute a sequence of direct inclusions of the form $e = R_1 \circ R_2 \circ \ldots R_{n-1} \circ R_n$, where $\circ$ can be any one of $\supset_d$ or $\subset_d$, and R_i,$1 \leq i \leq n$, are region names (examples for the use of such inclusion expressions are presented in [CM94]). We consider below the case where $\circ = \supset_d$ (since $\subset_d$ can be handled in a completely analogous manner). A naive computation, that uses the above program, may be very expensive, since each direct inclusion entail loop execution. It turns out that this can be avoided, and in fact **one** loop is sufficient for computing the sequence. For an expression e of the form discussed above, and a region name T, we use the notation $\#_e^T$ to denote the number of occurrences of T in $R_2 \ldots R_{n-1}$. The expression e can be computed as follows:

```
$R_1^{layer} := R_1 - (R_1 \subset R_1)$;
$R_1^{rest} := R_1 - R_1^{layer}$;
$R_1^{result} := \emptyset$;
$All = \bigcup_{T \in \mathcal{I}} T(\subset T)^{\#_e^T}$;
while $R_1^{layer} \neq \emptyset$ do
    $R_1^{result} := R_1^{result} \cup (R_1^{layer} \supset R_2 \supset \ldots \supset R_{n-1}$
            $\supset (R_n - (R_n \subset All \subset R_1^{layer})))$;
    $R_1^{layer} := R_1^{rest} - (R_1^{rest} \subset R_1^{rest})$;
    $R_1^{rest} := R_1^{rest} - R_1^{layer}$;
end
return $R_1^{result}$
```

The number of iterations of the program is determined by the nesting depth of the input. The execution cost of each iteration is dominated by the inclusion test involving the set *All* defined in the first line of the program, and is heavily influenced by the size of the set. Can the size be reduced? This is another example where the RIG can be used for optimization. Assume we consider only instances satisfying a RIG G. It is easy to see that *All* should not necessarily contain all the regions in I. In fact, it is sufficient to consider a subset $\mathcal{I}' \subseteq \mathcal{I}$, where $\mathcal{I}'$ contains at least one region name on every path from R_i to R_{i+1}, $i = 1 \ldots n-1$, (not including the endpoints of the path). We would like to use such minimal set. Computing it is, however, expensive. We call the problem of testing if there is such an $\mathcal{I}'$ with $|\mathcal{I}'| \leq k$, for some constant k, the *minimal set* problem.

Proposition 6.1 *The minimal set problem is NP-complete.*

Proof: (Sketch) The NP algorithm guesses $\mathcal{I}'$ of size k and checks if it satisfies the requirements. The hardness proof is by reduction from the vertex cover problem known to be NP-complete. □

Note that if e contains only one operation, (i.e. $e = R_1 \circ R_2$), a minimal set can be computed in in time polynomial in the size of the RIG (using a variant of the min-cut problem [PS82]).

7 Conclusion

We conclude by discussing two possible extensions to the algebra. The region algebra studied in the previous sections can be viewed as a restricted version of the relational algebra, allowing only 1-ary relations with attributes whose domain is the domain of regions, and with a semi-join operation that uses $\subset, \supset, <, >$. The language has a very efficient implementation, and can be optimized. However, it lacks expressiveness. This is due to two factors. First, only unary relations are allowed. Second, the content of regions can be tested only in a very limited way: only a bounded number of patterns can be checked by a given expression, and joins according to the content of regions are not supported.

To extend the capabilities of the language, one may allow queries to have n-ary relations (with attributes over the region domain) as intermediate results, and support joins and not only semijoins. Note that expressions in this extended language correspond to safe FMFT formulas, (where safety of formulas is defined in the standard way [Ull88]). This is because the input can still be encoded by monadic predicates. Theorem 3.6 holds for the extended language, and thus queries can be optimized. It is easy to see that direct inclusion and both-included can be expressed by this extended language.

To enable also to explore the content of regions (and allow joins that compare the content of regions), the word index should be part of the input to the query. The price we pay for this further extension is that the input is no longer monadic (recall that the word index is viewed as a 2-ary relation). Thus emptiness testing, and optimization become undecidable.

References

[ACM93] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proc. of VLDB 93*, pages 73–84, 1993.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *FODO*, 1993.

[BGMM93] D. Barbara, H. Garcia-Molina, and S. Mehrota. The gold mailer. In *IEEE Data Eng.*, pages 92–99, 1993.

[Bur92] F. J. Burkowski. An algebra for hierarchical organized text-dominated databases. *Information procecessing and management*, 28(3):333–348, 1992.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proc. ACM Sigmod*, pages 313–324, 1994.

[CCB95] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 1995. To appear.

[CH90] K. Compton and C.W. Henson. A uniform method for proving lower bounds on the complexity of logical theories. *Annals of Pure and Applied Logic*, 48:1–79, 1990.

[CM94] M. Consens and T. Milo. Optimizing queries on files. In *Proc. ACM Sigmod*, pages 301–312, 1994.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49, 1961.

[GNOT92] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *CACM*, 35(12), December 1992.

[Gon87] G. Gonnet. Examples of PAT applied to the Oxford English Dictionary. Technical Report OED-87-02, University of Waterloo, 1987.

[GT87] G. Gonnet and F. Tompa. Mind your grammar: a new approach to modelling text. In *Proc. of VLDB 87*, pages 339–346, 1987.

[GZC89] R. H. Guting, R. Zicari, and M. C. Choy. An algebra for structured office documents. *ACM transactions on office information systems*, 7(4):123–157, April 1989.

[IK89] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83:121–139, 1989.

[NBY95] G. Navarro and R. Baeza-Yates. A language for queries on structure and contents of textual databases. In *Proc. of the 18th. SIGIR*, 1995. To appear.

[Ope93] Open Text Corporation. *PAT Reference Manual and Tutorial*, 1993.

[Pae93] A. Paepcke. An object oriented view onto public hetrogeneous text databases. In *IEEE Data Eng.*, page 484, 1993.

[PS82] C. H. Papadimitriou and K. Steiglitz. Combinatorial optimization, algorithms and complexity. *Prentice-Hall*, 1982.

[Rab69] M. Rabin. Decidability of second order theories and automata on infinite trees. *Trans. American Mathematical Society*, 141:1–35, 1969.

[SLS$^+$93] K. Shoens, A. Luniewski, P. Schwartz, J. Stamos, and J. Thomas. The Rufus system: Information organization for semi-structured data. In *Proc. of VLDB 93*, pages 97–107, 1993.

[ST92] A. Salminen and F. W. Tompa. PAT expressions: an algebra for text search. In *Papers in Computational Lexicography: COMPLEX'92*, pages 309–332, 1992.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science, Vol. B*, pages 135–191, 1990.

[Ull88] J.D. Ullman. Database and knowledge base systems. *Computer Science Press*, 1988.