Algebras for Querying Text Regions: Expressive Power and Optimization

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 increased availability of documents in digital form. Document data has a natural hierarchical structure, which may be made explicit due to the use of mark-up conventions (as 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. This algebra is part of the language in use in commercial text retrieval systems, and can form the basis for supporting SQL-like access to textual data.

By presenting a close relationship between the region algebra and the monadic first order theory of finite binary trees, we show that queries in the algebra can be optimized, in the sense that equivalence to less expensive expressions can be tested. This optimization can be difficult (co-NP-hard in the general case), but 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, and consider one such example.

1 Introduction

Recently, there has been much interest in developing database tools for manipulating structured documents such as programs, news, patents, reports, SGML documents in general and HTML World Wide Web pages [BLCL+94] in particular. Work in this area has studied high level languages for expressing queries and updates on files, and efficient execution engines for file manipulation [GNOT92, ACM93, BGMM93, SLS+93, Pae93, CM94, ACM95]. 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 a word index recording the location of the word occurrences in the text, while more advanced systems also support a region index recording location of various text 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 [Gon87, ST92, Bur92, AFS93, KM93, 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 directed towards formally characterizing the capabilities of the resulting systems, and hence being able to answer questions such as:

- What kind of structure-related information can be searched and extracted by the system?
- What queries cannot be expressed?
- Is it possible to support more complex queries without harming the performance or hindering the opportunities for optimization?

Our goal in this work is to characterize the capabilities of a suitable query interface for structured and semistructured text indexing systems. We are interested in languages that can form the basis for supporting SQL-like access to textual data [BCK+94]. We look for a language that on 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 mean not only that the language belongs to a low worst-case complexity class, but also that it can serve as a basis for 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 interested here in the expressibility of arbitrary queries, but rather in those with more obvious practical use (i.e., finding the name of the procedure in a source code repository where a given variable has been declared).

We start by studying an algebra that is the core of the PAT text retrieval system[Ope95] (currently in use, for example, to provide a widely accessible index on the World Wide Web). We chose this algebra because it includes most of the operators considered in other proposals that support indexing text regions [Bur92, NBY95, CCB95], and, as mentioned above, has been implemented in an actual system.

By showing a close relationship between this algebra and the monadic first order theory of finite binary trees, we were able to prove that queries in the algebra are optimizable, in the sense that equivalence to less expensive expressions can be tested. 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, and consider one such example.

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 finite binary trees is studied in Section 3. Section 4 presents two technical 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 discusses directions for future research.

2 Text Regions

We start by presenting a region algebra that is a modified version of 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, currently in use in commercial systems such as [Ope95]. 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. These assumptions are captured in the definition below.

Definition 2.1 A **region index schema** $\mathcal{I} = (R_1, \ldots, R_n, W)$ is a set of region names $R_1, \ldots, R_n$, together with a word index name W.

Let $reg = \{(i, j) | i, j \in \mathcal{N}, i \leq j\}$ be the domain of regions. An **instance** of a region name R_i is a finite subset of reg.

Given some pattern language P, an **instance** I of a region index schema $\mathcal{I}$ is a mapping associating an instance $R_i(I)$ to each region name R_i, and a mapping associating some boolean value $W(r, p)$ to every pair (r, p) where $r \in \bigcup_{1 \leq i \leq n} R_i(I)$ and p is a pattern expression in the pattern language P.

As a notational convenience when I is understood from the context, we use R_i for both the region name and the instance $R_i(I)$. Also, we say that region r belongs to an instance I, denoted $r \in I$, if $r \in \bigcup_{1 \leq i \leq n} R_i$. Similarly, we use $S \subseteq I$ to denote a set S of regions that belong to I.

As an example, assume we are indexing program files, and are interested in the procedures, functions, and variable definitions in them. Then, our region index schema may have the form $\mathcal{I} = (\mathsf{Proc}, \mathsf{Func}, \mathsf{Var}, W)$. When a given program file is being indexed, the Proc region name is associated with the set of procedure regions in the file, i.e. is assigned a set of pairs (i, j) describing the positions of the beginning and end of each procedure in the file. Similarly, the Func and Var region names are instantiated by the set of regions of functions and variable definitions, resp. If the pattern language supported by the system is the language of regular expressions, then the word index W should record which regular expressions are contained in each of the above indexed regions. So $W(r, p)$ is assigned true for every pair (r, p) where r is a procedure, function or variable region and p is a regular expression contained in r, and is assigned false otherwise.

The definition above is rather general; a region name R_i may be associated with an arbitrary set of regions. A given region may belong to several R_i's, and regions may overlap, with or without being included in each other. Many researchers have concentrated on files with hierarchical structure [GT87, GZC89, Bur92, ACM93, CACS94]. 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 concentrate in the rest of this paper on the class of instances that defines a hierarchy of nested regions. This is defined formally below.

We use the notation $L(r)$ (resp. $R(r)$) to denote the location of the left (resp. right) endpoint of a region r. Two regions r, s do not overlap if $R(r) < L(s)$ or $R(s) < L(r)$. A region r strictly includes the region s if $L(r) < L(s)$ and $R(r) \geq R(s)$, or $L(r) \leq L(s)$ and $R(r) > R(s)$.

Definition 2.2 We say that an instance I of a region index schema $\mathcal{I}$ is **hierarchical**, if every two region name instances R_i, R_j of I are disjoint, and every two regions $r, r' \in I$ either do not overlap, or one is strictly included in the other.

From now on we consider **only hierarchical instances**, and whenever we use the word instance we mean an hierarchical one.

Note that a region index schema can be viewed as a relational database (with one relation per region name, 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. The PAT algebra is an example of such a restricted algebra that is used by a commercial system [Ope95]. The algebra we consider here differs from the original PAT algebra in several aspects. One of them is that the latter is capable of constructing sets of regions dynamically based on text patterns. From the point of view of this work we can consider these sets as regular predefined region names in the schema. On the other hand, we consider self-nested region sets, which are not supported by PAT.

Definition 2.3 Region algebra **expressions** over a region index schema $\mathcal{I}$ are expressions generated by the grammar

$$e \to R_i \mid e \cup e \mid e \cap e \mid e - e \mid e \supset e \mid e \subset e \mid e < e \mid e > e \mid \sigma_p(e) \mid (e)$$

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 $r \supset s$, where r,s are two regions, to denote the fact that the region r strictly includes the region s. Similarly, we use $r < s$, to denote the fact that the region r precedes the region s (i.e., $R(r) < L(s)$).

Definition 2.4 The **union** ($\cup$), **intersection** ($\cap$)), and **difference** ($-$) operations 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 a region 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 obey particular inclusion relationships among regions. For instance, consider a file containing the source code of programs. Assume that each program has a header including the program name, and a body containing definitions of variables and procedures. In turn, each procedure has a 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, W), be the region index schema, where W is the word index name and the rest are region names. Consider the two expressions:

$e_1 = \mathsf{Name} \subset \mathsf{Proc\text{-}header} \subset \mathsf{Proc} \subset \mathsf{Program}$

$e_2 = \mathsf{Name} \subset \mathsf{Proc\text{-}header} \subset \mathsf{Program}$

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, hence it 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 $t \supset s$. Intuitively, 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 2.5 An instance I of a region index schema $\mathcal{I} = (R_1, \ldots, R_n, W)$ **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 a schema. We therefore have the following definition.

Definition 2.6 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 discussed previously are described by the RIG shown in Figure 1. The queries e_1 and e_2 above are equivalent with respect to that RIG.

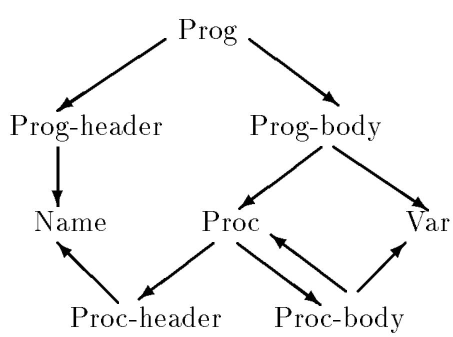

Figure 1: A RIG for source code regions.

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 on the right side.

One of the most attractive aspects of employing regions for describing the structure of text is their flexibility. Regions can be used (as in the previous example) to identify non-terminals recognized by a parser. With more generality, an arbitrary program can recognize regions in text and provide them for use in querying. In the case of HTML documents, this program can be a general SGML parser using the appropriate HTML grammar, or it can simply be a small function recognizing a few of the HTML constructs. For example, we can recognize one **HTML** region per document and within it distinguish the **Head** region (which includes the document **Title**) and **Body** region, based on their corresponding tags. Further within the body of the HTML document the regions recognized could correspond to the headings (all levels using one **Heading** region) and the **Anchor** regions (where the global hypertext links originate). The structure of the anchors can be further described by distinguishing an **Anchor-text** region (the click-able text that appears underlined in some browsers) and an **Href** region corresponding to the attribute with the same name appearing within the anchor element (this is the URL of the location that the anchor links to). Figure 2 shows the RIG satisfied by a collection of HTML documents indexed using the regions described above. It is common for HTML documents to have headings that are also links, and this can be done by either embedding an anchor in a heading, or vice versa, hence the presence in the RIG of mutual direct inclusion edges between **Heading** and **Anchor** regions.

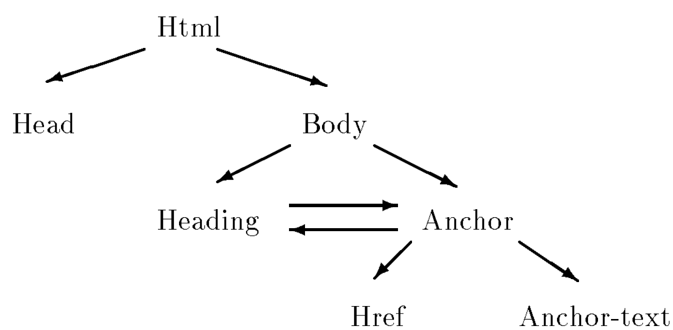

Figure 2: A RIG for HTML documents.

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 describes 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 $t < s$. As for RIGs, 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.

For example, the ROG shown in Figure 3 describes the order restrictions on the regions of the program files discussed previously. A program starts with a program header (containing the name). The header (and the name in it) is followed immediately by the program body. The body starts with a (possibly empty) sequence of variable definitions, that can be followed by a sequence of procedures. Each procedure starts with a procedure header (containing the procedure name), followed immediately by the procedure body (starting with the variable definitions, possibly followed by definitions of local procedures). So the sequence of procedures can also be viewed as an alternating sequence of procedure headers (or names) and procedure bodies.

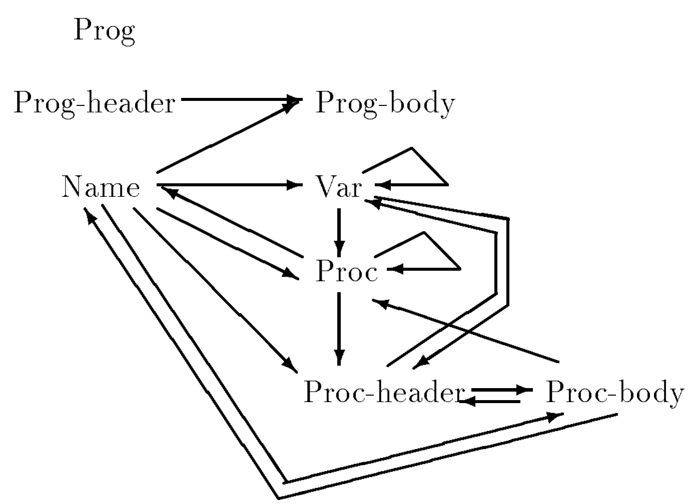

Figure 3: A ROG for source code regions.

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) [EBG96, Tho90]. This relationship helps to show that queries in algebra are optimizable, in the sense that equivalence to less expensive expressions can be tested.

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 $\{0,1\}^*$ (i.e., finite sets of binary strings)[1]. For a model t, we use the term the *words* in t to refer to the set of binary strings $\bigcup_{1 \leq i \leq n} Q_i$.

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 [EBG96, Tho90] for more details). We use the notation $\phi(t)$ to denote the result of the formula ϕ when evaluated against a model t. We say that an FMFT formula ϕ with free variables $x_1, ..., x_n$ is *unsatisfiable* if for every FMFT model t, the set $\{x_1, ..., x_n | \phi(t)\}$ is empty [EBG96]. [2]

We next study the relationship between instances of region index schema and FMFT models. Once this is understood we will explain the relationship between algebra expressions and FMFT 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 u is a prefix of v and 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, and is not its prefix. 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.

[1] 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, EBG96] 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 ($<$) and prefix order ($\supset$) can be defined in terms of $succ_0$ and $succ_1$.

[2] Note that the literature [Tho90, CH90] considered mostly infinite binary trees, hence infinite models. We focus here on **finite** ones (FMFTs). Thus (un)satisfiability is defined w.r.t to finite models only.

The relationship between models and region instances is captured by the next definition.

Definition 3.1 Let $\mathcal{I} = (R_1, \ldots, R_n, W)$ be a region index schema and $P = \{p_1, \ldots, p_k\}$ 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 Q_i, $1 \leq i \leq n$, are pairwise disjoint, each word in Q_j, $n < j \leq n+k$ appears in some $Q_{i'}$, $1 \leq i' \leq n$, and there is a bijection (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 (that does not have u as a prefix) in t, 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.

Intuitively, every Q_i, $1 \leq i \leq n$, represents a region name R_i. Items (1) and (2) above require that the representation captures the inclusion and order relationship among text regions. The inclusion is captured by the prefix relationship among words, and the order is captured by the precedence relationship among words. Item (3) establishes that the set of words in each such Q_i represents the instance of the corresponding R_i, with each distinct word representing one text region. The requirement that these Q_i's are disjoint follows from the fact we are dealing with hierarchical instances where the corresponding R_i's are disjoint (see Definition 2.2).

The Q_j, $n < j \leq n+k$, represent the part of the word index W that deals with the patterns $p_1, \ldots, p_k$. The words in each such Q_j represent the set of regions containing the pattern p_{j-n} (item (4) above). Recall from Definition 2.1, that the word index records containment information about regions in the R_i's. This is why each word in the Q_j's, $n < j \leq n+k$ is required to be in some $Q_{i'}$, $1 \leq i' \leq n$.

For example, consider the program file depicted in Figure 4, where some of the regions and words in the file are marked. Assume we have a region index schema $\mathcal{I} = (\mathsf{Prog}, \mathsf{Proc}, \mathsf{Var}, \mathsf{W})$.

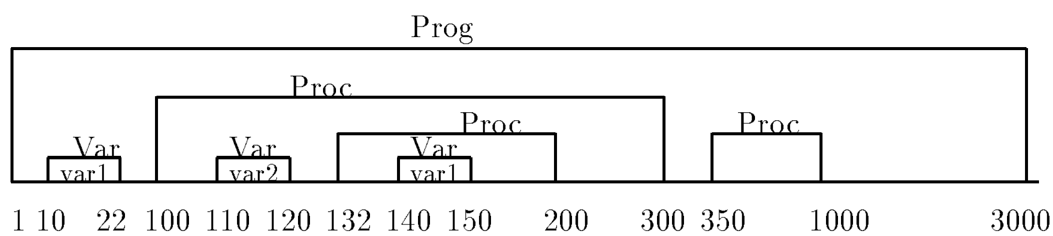

Figure 4: A program file, with some of the regions/words marked.

An instance of the schema may have the following structure.

Prog	Proc	Var	W
1-3000	100-300	10-22	(10-22,“var1”)
	350-1000	110-120	(140-150, “var1”)
	132-200	140-150	(110-120, “var2”)
	...	...	...

The hierarchical inclusion and order relations among the regions are captured in the tree of Figure 5.

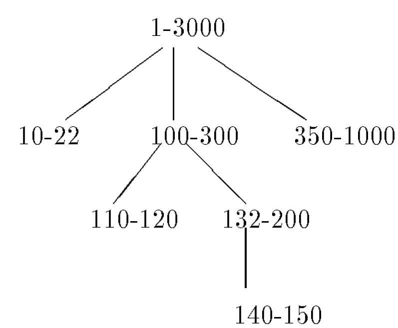

Figure 5: Regions tree.

Now, we can represent each region by a word, and record the inclusion and order relations among regions by choosing words with corresponding prefix and precedence relations. Such a representation is illustrated in the tree of Figure 6.

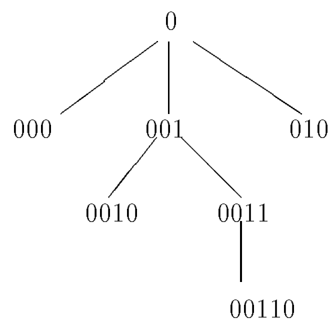

Figure 6: Words tree.

Using this representation, and assuming that we are only interested in the patterns (strings) "var1" and "var2", the above instance can be represented by the following FMFT model (where each region name and pattern is represented by a predicate with the same name).

Prog	Proc	Var	var1	var2
0	001	000	000	0010
	010	0010	00110	...
	0011	00110	...	
	...	...		

In general, every region instance has an FMFT model that represents it. Moreover, as we show below, under some restrictions, the opposite direction holds as well. This close relationship between region instances and FMFT models is formally stated in the following proposition.

Proposition 3.2 *Given a region index schema $\mathcal{I} = (R_1, \ldots, R_n, W)$, and a set of patterns P,*
(a) Every instance I of $\mathcal{I}$ has some FMFT model t that represent it w.r.t. P, and
(b) Every FMFT 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 I of $\mathcal{I}$.

Proof: We start with part (a). Given an instance I we first build a mapping

$region_I$ from words in $\{0,1\}^*$ to regions in I, satisfying requirements 1 and 2 of Definition 3.1.

For that, we first take all the regions in I and build an ordered forest whose nodes are the regions in I, and where parenthood and order relations among the nodes correspond to the direct inclusion and relative order, resp., among the regions in I. (Note that this is always possible because we are dealing in this paper with hierarchical instances.) Then, we start from the roots of the forest and, going down the forest, label all the nodes with words over $\{0,1\}^*$. We start by labeling the roots: Let i be the number of roots. We take i distinct words of length $\lceil log(i) \rceil$, in increasing order, and use them to label the roots (again in increasing order). Then, we proceed down the tree. Given a node n with label w_n, we label the children of n as follows: Let i_n be the number of children of n. We take i_n distinct word of length $\lceil log(i_n) \rceil$ in increasing order, concatenate w_n as a prefix to each of the words, and then use the resulting words to label the children of n (again in increasing order).

The mapping $region_I$ maps each word w in the labeled forest to the region r of the labeled node. It is easy to see that this mapping satisfies requirements 1 and 2 of Definition 3.1.

Now we build the Q_i's so as to satisfy requirements 3 and 4: each Q_i, $1 \leq i \leq n$ contains exactly the words u such that $region_I(u) \in R_i$, and each Q_{n+j}, $1 \leq j \leq k$ contains exactly the words u such that $W(region_I(u), p_j)$ is true.

Note that since the R_i's in I are disjoint, so are the Q_i, $1 \leq i \leq n$. Also, from the construction it is clear that all the words in Q_{n+j}, $1 \leq j \leq k$ appear in some Q_i, $1 \leq i \leq n$. Thus the FMFT model we obtained satisfies all the restrictions of Definition 3.1, and hence represents the instance I.

For part (b), assume we are given an FMFT model $t = (\{0,1\}^*, \supset, <, Q_1, \ldots, Q_{n+k})$ satisfying restrictions (i) and (ii) above. We build an instance I s.t. t represents I w.r.t. P. As before, we start by building a mapping $region_I$ satisfying requirements 1 and 2 of Definition 3.1 and mapping words in $\bigcup_{i=1\ldots n} Q_i$ to regions.

The construction of the mapping is similar to that of part (a) above. We first take all the words in $\bigcup_{i=1\ldots n} Q_i$ and build an ordered forest whose nodes represent the words, and where parenthood and order relations describe the prefix and precedence order among the words. Then, we start from the roots of the forest and going down the forest, label the nodes with regions (i,j).

[3] We start by labeling the roots: Let $count_k$ be the number of nodes in the tree rooted at the root k, and let $count_{<k}$ denote the total number of nodes in all the trees whose root precede k in the forest. The region that we associate with k is the pair $(1 + count_{<k}, 1 + count_{<k} + count_k)$.

Then, we proceed down the tree. Given a node n with label (i_n, j_n), we label the children of n as follows: Let $count_c$ be the number of nodes in tree rooted at a child node c, and let $count_{<c}$ denote the total number of the nodes in all the trees rooted at the children of n that precede c. Then the region associated with c is $(i_n + count_{<c}, i_n + count_{<c} + count_c)$.

The mapping $region_I$ maps each word w in the FMFT model to the region labeling the node of w. From the construction, it is easy to see that all the regions are either disjoint or one is strictly included in the other, and that the mapping satisfies requirements 1 and 2 of Definition 3.1. Now we build the R_i's and W so as to satisfy requirements 3 and 4: each R_i, $1 \leq i \leq n$ contains exactly the regions $region_I(u)$ s.t. $u \in Q_i$, and $W(region_I(u), p_j)$ is true iff Q_{n+j} contains the word u. Since the Q_i,$1 \leq i \leq n$ are all disjoint, so are all the R_i's; hence we obtain a hierarchical instance I that is represented by t.

□

We are now ready to consider the relationship between region algebra expressions and FMFT formulas. We are especially interested here in a specific class of formulas, called **Restricted FMFT formulas**, because, as we show, the region algebra and the restricted FMFT formulas express the same queries on regions.

Definition 3.3 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) $(\exists y)\phi_1 \wedge \phi_2 \wedge x \supset y$, and $(\exists y)\phi_1 \wedge \phi_2 \wedge y \supset x$, where ϕ_1 and ϕ_2 have one free variable each, x and y (distinct from each other).

(iii) $(\exists y)\phi_1 \wedge \phi_2 \wedge x < y \wedge \neg(x \supset y)$, and $(\exists y)\phi_1 \wedge \phi_2 \wedge y < x \wedge \neg(y \supset x)$, where ϕ_1 and ϕ_2 have one free variable each, x and y (distinct from each other).

[3] i (j) denotes the location of the right (left) end point of a region.

Proposition 3.4 *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 instance I, every model t representing I w.r.t. P, and every word w in t, $region_I(w) \in e(I)$ iff $w \in \phi(t)$.*

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

We first outline the construction to obtain a restricted formula from a region algebra expression. 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. Assume first that the only free variable in each of ϕ_1 and ϕ_2 has been renamed to the same variable name x. The translations for the expressions $e_1 \cup e_2$, $e_1 \cap e_2$, $e_1 - e_2$ is $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\phi_1 \wedge \neg\phi_2$, resp. The translation for $\sigma_{p_i}(e_1)$ is $\phi_1 \wedge Q_{n+i}(x)$.

For the rest of the operators, assume that the free variable in each of ϕ_1 and ϕ_2 has been renamed to x and y, resp. The translations for the expressions $e_1 \supset e_2$ and $e_1 \subset e_2$ are $(\exists y)\phi_1(x) \wedge \phi_2(y) \wedge x \supset y$ and $(\exists y)\phi_1(x) \wedge \phi_2(y) \wedge y \supset x$, resp. The translations for the expressions $e_1 < e_2$ and $e_1 > e_2$ are $(\exists y)\phi_1(x) \wedge \phi_2(y) \wedge y < x \wedge \neg(y \supset x)$ and $(\exists y)\phi_1(x) \wedge \phi_2(y) \wedge x < y \wedge \neg(x \supset y)$, resp.

Obtaining a region algebra expression from a restricted formula is completely analogous, except for the case of base formulas of the form $\{x|Q_i(x)\}$. Here we distinguish two cases. If $i \leq n$ then the corresponding formula is R_i. If $n < i \leq n+k$ then the corresponding formula is $\sigma_{p_{i-n}}(\bigcup_{j=1\ldots n} R_j)$.
$\Box$

The above proposition, together with Proposition 3.2, implies the following result.

Theorem 3.5 *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.*

Proof: The proof follows from Propositions 3.4 and 3.2, and from the fact that the conditions (i) and (ii) of Proposition 3.2(b) can be expressed by a restricted formula.

We first use the construction in the proof of Proposition 3.4 and translate e into a restricted formula ϕ_1. Then we use ϕ_1 to construct a restricted formula $\phi(x)$ such that for every model t, if t satisfies the two conditions of Definition 3.2(b) then the set $\{x|\phi(x)\}$ is equal to the set $\{x|\phi_1(x)\}$, and is empty otherwise. Thus ϕ is unsatisfiable iff $e(I)$ is always empty.

The formula ϕ has the form $\phi(x) = \phi_1(x) \wedge \phi_2(x)$ where ϕ_2 has the property that for every model t satisfying the two conditions, $\{x|\phi_2(x)\}$ is the set of all words in t, and is empty otherwise.

To describe the construction of the desired ϕ_2 we use three auxiliary sub-formulas:

$$
\begin{aligned}
\phi_3(x) &= \textstyle\bigvee_{i=1\ldots n} Q_i(x) \\
\phi_4(x) &= \textstyle\bigvee_{i=1\ldots n+k} Q_i(x) \\
\phi_5(x) &= \textstyle\bigvee_{1\leq i,j\leq n, i\neq j} (Q_i(x) \wedge Q_j(x)) \vee \bigvee_{n<i\leq n+k} (Q_i(x) \wedge \neg\phi_3(x)) \\
\phi_6(x) &= (\phi_4(x) \wedge \phi_5(y) \\
&\qquad \vee(\exists y)\phi_4(x) \wedge \phi_5(y) \wedge x \supset y) \\
&\qquad \vee((\exists y)\phi_4(x) \wedge \phi_5(y) \wedge x < y \wedge \neg(x \supset y)) \\
&\qquad \vee((\exists y)\phi_4(x) \wedge \phi_5(y) \wedge y \supset x) \\
&\qquad \vee((\exists y)\phi_4(x) \wedge \phi_5(y) \wedge y < x \wedge \neg(y \supset x))
\end{aligned}
$$

The formula ϕ_3 computes the set of all words in $Q_1, \ldots, Q_n$. The formula ϕ_4 computes the set of all words in t. The formula ϕ_5 computes the set of words in t violating conditions (i) or (ii) of Definition 3.2(b). Finally, ϕ_6 computes a set of words containing all the words of ϕ_5 plus all the words in t that are smaller or greater than any of these words. (For being smaller than a word y in ϕ_5 we consider the two cases of being a prefix of y, or preceding it in the lexicographical order without being a prefix. For being greater than y we consider the two cases of y being a prefix, and y being first in the lexicographical order without being a prefix.)

Note that if ϕ_5 is empty, i.e. t satisfies conditions (i) and (ii), then ϕ_6 is also empty. On the other hand, if ϕ_5 is not empty and contains some word y, than any other word x in t either equals y or is smaller or greater than y. Thus, if ϕ_5 is not empty then ϕ_6 computes the set of *all* words in t.

The desired ϕ_2 is obtained as the complement of this ϕ_6, that is, $\phi_2(x) = \phi_3(x) \wedge \neg\phi_6(x)$.

□

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 3.5 as follows.

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.*

Proof: The proof is similar to that of Theorem 3.5 and is based on the fact that the direct inclusion restrictions imposed by the RIG can be expressed by FMFT formulas. Let ϕ' be the formula constructed in the proof of Theorem 3.5. Let $G = (V, E)$ be some RIG. Let S be the set of all pairs (i, j) s.t. $(v_i, v_j) \notin E$, ϕ has the form

$$\phi = \phi'(x) \wedge \phi''$$

where

$$\begin{aligned}\phi'' = \textstyle\bigwedge_{(i,j)\in S} \quad & (\forall x, y) Q_i(x) \wedge Q_j(y) \wedge x \supset y \\ & \rightarrow ((\exists w)(\textstyle\bigvee_{k=1\ldots n} Q_k(w)) \wedge x \supset w \wedge w \supset y)\end{aligned}$$

Intuitively, the formula ϕ'' tests that whenever an edge (v_i, v_j) is not in the RIG, and a word representing an R_i region is a prefix of a word that represents an R_i region (hence the corresponding regions include one another), it is the case that the prefix is not direct (hence the inclusion is not direct). So the only direct prefix relations in the model (and hence the corresponding direct inclusions in the region instance) are those permitted by the RIG. □

Note that the above theorem uses *general* FMFT formulas and not restricted FMFT formulas. This is because (as we show later in Section 5.1) direct inclusion cannot be expressed by restricted formulas.

Satisfiability of FMFT formulas is decidable [EBG96], and thus testing if $e(I)$ is empty for every instance I is decidable.

Given an index region schema $\mathcal{I}$ and an algebra expression e over $\mathcal{I}$, we call the problem of testing if $e(I)$ is empty for every region instance I of $\mathcal{I}$, the **emptiness testing** problem. Emptiness testing can be used to optimize queries. Suppose we have a price function 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 of size smaller or equal to e). Two expressions e_1, e_2 are equivalent iff $(e_1 - e_2) \cup (e_2 - e_1)$ is empty for all instances. Thus the refined Theorem 3.6 can be used to optimize queries w.r.t. a RIG.

Testing satisfiability of FMFT formulas has non-elementary complexity [EBG96, Rab69, CH90], thus the above optimization technique is very expensive. However, in our case the algebra queries correspond to a very limited class of FMFT formulas (restricted formulas), so we may hope that emptiness testing is less expensive. The following theorem shows that, even for such restricted formulas, emptiness testing (and hence optimization) cannot be done in polynomial time (unless P=NP). The exact complexity bounds remain an open problem.

Theorem 3.7 *Emptiness testing in the region algebra is co-NP-hard.*

Proof: The proof is by reduction from the problem of checking if a 3-CNF formula is unsatisfiable, known to be Co-NP-complete. We show below that for every 3-CNF formula ϕ, there exists a region index schema $\mathcal{I}$ and an expression e, s.t. $e(I)$ is empty for all instances iff ϕ is unsatisfiable, where $\mathcal{I}$ and e are constructed in time polynomial in the size of ϕ.

Consider a 3-CNF formula $\phi = \bigwedge_i (A_{i_1} \vee A_{i_2} \vee A_{i_3})$ where $A_{i_j}, j = 1, 2, 3$ is either the variable x_{i_j} or its negation. The corresponding region index schema is $\mathcal{I} = (F, X_1, \ldots, X_n, W)$, where $x_1, \ldots, x_n$ are the variable names in ϕ. The expression e has the form $e = \bigcap_i (e_{i_1} \cup e_{i_2} \cup e_{i_3})$ where $e_{i_j} = (F \supset X_i)$ if $A_{i_j} = x_{i_j}$, and $e_{i_j} = (F - (F \supset X_i))$ if $A_{i_j} = \neg x_{i_j}$. We show below that $e(I)$ is empty for all instances iff ϕ is unsatisfiable.

First assume that there is some instance I of $\mathcal{I}$ s.t. $e(I) \neq \emptyset$. From the structure of e this implies that I has some F region containing a set S of regions, s.t. for every clause in ϕ, for at least one literal A_{i_j}, if A_{i_j} is a variable x_{i_j} then S contains some X_{i_j} region, and if A_{i_j} is a negated variable $\neg x_{i_j}$ then S contains no X_{i_j} region. It is easy to see that a variable assignment that assigns *true* to the variables corresponding to the region sets with members in S, and *false* to all the other variables, is a satisfying assignment for ϕ.

For the other direction assume that ϕ is satisfiable. Consider some satisfying assignment, and let $x_{i_1}, \ldots, x_{i_j}$ be the variables assigned *true* by this assignment. Consider a region index instance I containing a single F region f, that contains one X_{i_k} region for each of the x_{i_k}, $k = 1 \ldots j$, and contains no other regions. It is easy to see that $e(I) = \{f\}$, thus $e(I) \neq \emptyset$.

□

There are, however, some important practical cases (see Section 5.1) where emptiness testing and optimization can be done in polynomial time.

4 Properties of the Region Algebra

We now present two properties of the region algebra, that provide the technical tools for proving the expressiveness results of the following section. The observation is that, under certain conditions, algebra queries are not affected by the removal of regions from the input. This fact is used in Section 5 to show that certain queries are inexpressible in the algebra because they should have been affected by such a removal of regions.

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 region instance I, there exists a set of regions $S \subseteq I$ 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 I and I', $r \in e(I)$ iff $r \in e(I')$.

Proof: We first prove part (2). We use induction on the number of operations in e, and constructively build S. Note that in some cases we add to S more elements than are actually needed to prove (2). These extra elements will be used later when proving (1).

Basis: If $e = R_i$, then $S = \{r | r \in R_i(I) \;\wedge\; \neg\exists r' \in R_i(I), r' \supset r\}$, i.e. S is the set of outermost R_i regions. The nesting depth of S is 1, and (2) clearly holds since the regions in I' belong to the same region sets as in I.

Induction: Let e_1, e_2 be two expressions, and let $S_1, S_2 \subseteq I$ be sets of regions for which (2) holds for e_1, e_2, resp. Since $\cap$ can be expressed by $\cup$ and $-$, we have to consider only expressions constructed using $\cup, -, \sigma, \subset, \supset$, $<$ and $>$.

- If $e = e_1 \cup e_2$, then $S = S_1 \cup S_2$. The nesting depth of S is at most $2|e_1| + 2|e_1| < 2|e|$. Consider a region r that belongs to both I and I'. Note that every S-deleted version of I is also an S_i-deleted version, for $i = 1, 2$. Thus, from the induction step, $r \in e_i(I)$ iff $r \in e_i(I')$, for $i = 1, 2$. This implies that $r \in e_1(I) \cup e_2(I)$ iff $r \in e_1(I') \cup e_2(I')$.

- If $e = e_1 - e_2$ then $S = S_1 \cup S_2 \cup \{r | r \in e(I) \ \wedge \ \neg\exists r' \in e(I), r' \supset r\}$. Thus, S contains the regions in S_1 and S_2 and the outermost regions in $e(I)$. The nesting depth of S is at most $2|e_1| + 2|e_1| + 1 < 2|e|$. Again note that every S-deleted version of I is also an S_i-deleted version, for $i = 1, 2$. Thus, from the induction assumption it follows that for every region r, and every S-deleted version I' of I $r \in e_1(I)$ iff $r \in e_1(I')$, and $r \notin e_2(I)$ iff $r \notin e_2(I')$. Thus $r \in e(I)$ iff $r \in e(I')$.

- If $e = \sigma_p(e_1)$, then $S = S_1 \cup \{r | r \in e(I) \ \wedge \ \neg\exists r' \in e(I), r' \supset r\}$. The set S is at most one level more nested than S_1, thus of depth $< 2|e|$. The proof of (2) follows immediately from the fact that the word index is the same for the common regions in I' and I.

- If $e = e_1 \supset e_2$, then

  $$\begin{aligned} S = S_1 \cup S_2 \cup \{r | r \in e_1(I) \ \wedge \ \neg\exists r' \in e_1(I), r' \supset r\} \\ \cup \{s | s \in e_2(I) \ \wedge \ \neg\exists s' \in e_2(I), s \supset s'\}. \end{aligned}$$

  The nesting of regions in S is at most $2|e_1| + 2|e_2| + 2 \leq 2|e|$. Consider a region r in I. $r \in e(I)$ iff $r \in e_1(I)$ and there exists a region $s \in e_2(I)$ that is included in r. In particular consider the innermost such region (i.e. a region s that does not include another region in $e_2(I)$). As before, every S-deleted version of I is also an S_i-deleted version, for $i = 1, 2$. From the induction step, $r \in e_1(I)$ iff $r \in e_1(I')$. By the choice of S it follows that if such an s exists then $s \in S$, and thus belongs to I'. From the induction step if such an s exists and is in I', then $s \in e_2(I')$. Thus $r \in e(I)$ iff $r \in e(I')$.

- If $e = e_1 \subset e_2$, then

  $$\begin{aligned} S = S_1 \cup S_2 \cup \{r | r \in e_1(I) \ \wedge \ \neg\exists r' \in e_1(I), r \supset r'\} \\ \cup \{s | s \in e_2(I) \ \wedge \ \neg\exists s' \in e_2(I), s' \supset s\}. \end{aligned}$$

  The remainder of the proof is the same as for $e = e_1 \supset e_2$.

- If $e = e_1 < e_2$, then

  $$\begin{aligned} S = S_1 \cup S_2 \cup \{r | r \in e_1(I) \ \wedge \ \neg\exists r' \in e_1(I), (r \supset r' \vee r' < r\} \\ \cup \{s | s \in e_2(I) \ \wedge \ \neg\exists s' \in e_2(I), (s \supset s' \vee s < s'\}. \end{aligned}$$

  The nesting of regions in S is at most $2|e_1| + 2|e_2| + 2 \leq 2|e|$. Consider a region r in I. $r \in e(I)$ iff $r \in e_1(I)$ and there exists a region $s \in e_2(I)$ that follows r. In particular consider the innermost, leftmost such region (i.e. a region s that does not include another region in $e_2(I)$ and is the first such region in the file). As before, every S-deleted version of I is also an S_i-deleted version, for $i = 1, 2$. From the induction step, $r \in e_1(I)$ iff $r \in e_1(I')$. By the choice of S it follows that if such an s exists then $s \in S$, and thus belongs to I'. From the induction step if such s exists and is in I', then $s \in e_2(I')$. Thus $r \in e(I)$ iff $r \in e(I')$.

- If $e = e_1 > e_2$, then

  $$\begin{aligned} S = S_1 \cup S_2 \cup \{r | r \in e_1(I) \ \wedge \ \neg\exists r' \in e_1(I), (r \supset r' \vee r' > r\} \\ \cup\{s | s \in e_2(I) \ \wedge \ \neg\exists s' \in e_2(I), (s \supset s' \vee s > s'\}. \end{aligned}$$

  The remainder of the proof is the same as for $e = e_1 < e_2$.

This concludes the inductive proof of (2). The proof of (1) follows from the fact that the set S constructed by the above inductive process contains at least one region r in $e(I)$, if such a region exists.

□

4.2 Reduction

We next consider a more refined mechanism for removing regions from the input that preserves some containment, ordering, and word indexing properties of the original input. For that, we resort to an auxiliary notion of isomorphism between regions and define a new operation on instances. For an instance I and a region $r \in I$, we use the notation S_r to denote the smallest set containing r and all the regions in I that are included in r.

Definition 4.2 Let I be a region 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 bijective mapping τ from S_{r_2} to S_{r_1} s.t. (i) τ preserves the inclusion and precedence relationship of regions, and (ii) for every region $r \in S_{r_2}$, every region name R_i, 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 **reduce** 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_2}.

Let I be a region instance, and let I' be a region instance obtained from I by a sequence of reduce operations. Observe that 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 a bijection 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 call such a mapping h a **reduction mapping** and use it 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. We use the notation $h_{I,I'}$ to denote the reduction mapping from I to I' (defined as explained above) induced by the sequence.

An instance I' is a k-**reduced** $(k > 0)$ version of I w.r.t. P iff,
(1) it is a 0-reduced version of I w.r.t. P, and
(2) it has a (k-1)-reduced version I'' s.t. for every two regions $r, s \in I$, $r < s$ $(r > s)$ in I iff there exists $t \in h^{-1}_{I',I''}(h_{I',I''}(h_{I,I'}(s)))$ s.t. $h_{I,I'}(r) < t$ $(h_{I,I'}(r) > t)$ in I'.

To illustrate the notion of k-reduction, consider the following example. For simplicity, we assume below that the set of patterns P is empty. Assume we have an instance I describing the file in Figure 7 and containing the A, B, and C regions in the figure. We denote them below by $a_1 - a_4, b_1 - b_3$, and $c_1 - c_3$ (from left to right).

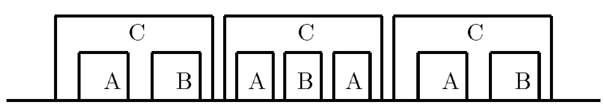

Figure 7: The file for the instance I

The regions a_2 and a_3 are isomorphic. Thus we can apply one reduction step $reduce(I, a_2, a_3)$ and obtain the instance I' described in Figure 8.

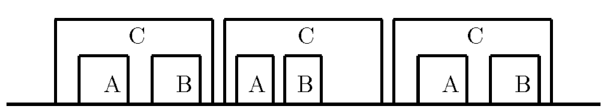

Figure 8: I', The 1-reduced version of I

Clearly, I' is a 0-reduced version of I. But I' is also a 1-reduced version of I. To see that, we have to show that item (2) in the definition above holds, i.e. we have to show that I' has a 0-reduced version I'' with the required properties. Such an I'' is presented in Figure 9.

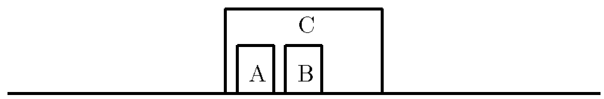

Figure 9: I'', The 0-reduced version of I'

Before we explain this, note that the reduction mapping $h_{I,I'}$ from I to I' maps all regions except a_3 to themselves (a_3 is mapped to a_2). So all order relations among regions, except those involving a_3 are preserved. What item (2) in the definition of k-reduction basically requires is that even when the reduction causes some specific order relation to be removed, the remaining instance still contains "traces" of them, in the sense that a similar relation exists between isomorphic images of the regions. For example, $a_2 < a_3$ in I, while in I' the two regions are both represented by the same region $h_{I,I'}(a_2) = h_{I,I'}(a_3)$ so one does not precede the other. But there is a "trace" for this relation: c_3 is isomorphic to c_2 and contains an A region that follows $h_{I,I'}(a_2)$ (this is a "trace" for the fact that a_3 followed a_2). Similarly, c_1 is isomorphic to c_2 and contains an A region that precedes $h_{I,I'}(a_3)$ (this is a "trace" for the fact that a_2 preceded a_3).

To see this, assume we apply to I' the following two reduction steps. First we map c_1 to c_2 and then c_3 to c_2. We obtain the 0-reduced version $I'' = reduce(reduce(I', c_2, c_1), c_2, c_3)$ described in Figure 9. The reduction mapping $h_{I',I''}$ from I' to I'' maps $c_1 - c_3$ to c_2, $b_1 - b_3$ to b_2, and a_1, a_2, a_4 to a_2. Thus we have that $h_{I,I'}(a_4) \in h^{-1}_{I',I''}(h_{I',I''}(h_{I,I'}(a_3)))$, and $h_{I,I'}(a_2) < h_{I,I'}(a_4)$. Similarly, $h_{I,I'}(a_1) \in h^{-1}_{I',I''}(h_{I',I''}(h_{I,I'}(a_2)))$, and $h_{I,I'}(a_1) < h_{I,I'}(a_3)$. Following the same lines, one can verify that I' contains similar "traces" for all the order relations in I and thus the requirements of item (2) are satisfied. 1-reduction allows the recovery of order information using one mapping. k-reduction allows for k such mappings.

The next theorem uses k-reduction to show that if enough order related information is preserved while reducing the size of the instance, the modifications to the input do not affect the result 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.

Basis: For $e = R_i$, the proof follows from the fact that the reduce operator only deletes elements and does not change the membership of the remaining elements in region sets, and that the mapping h maps regions to regions belonging to the same region set.

Induction: Assume the proposition holds for the expressions e_1, e_2. We show that it also holds for an expression e constructed from e_1, e_2 using the operations $\cup, -, \sigma, \subset, \supset, <$ and $>$ (we do not have to consider $\cap$ since it is expressible by the other operations), as follows:

- Let $e = e_1 \cup e_2$. $n \in e(I)$ iff $n \in e_i(I)$ for $i = 1$ or $i = 2$. Similarly, $h(n) \in e(I')$ iff $h(n) \in e_i(I')$ for $i = 1$ or $i = 2$. From the induction assumption, $n \in e_i(I)$ iff $h(n) \in e_i(I')$. Thus $n \in e(I)$ iff $h(n) \in e(I')$.
- Let $e = e_1 - e_2$. $n \in e(I)$ iff $n \in e_1(I)$ and $n \notin e_2(I)$. Once more, $h(n) \in e(I')$ iff $h(n) \in e_1(I')$ and $h(n) \notin e_2(I')$. Hence from the induction assumption, $n \in e_i(I)$ iff $h(n) \in e_i(I')$, and thus $n \in e(I)$ iff $h(n) \in e(I')$.
- Let $e = \sigma_p(e_1)$. The proof follows immediately from the induction assumption, and the fact that in the mapping τ used in the reduction $W(n,p)$ holds iff $W(\tau(n),p)$ holds.
- Let $e = e_1 \supset e_2$. $n \in e(I)$ iff $n \in e_1(I)$ and there exists a region $n' \in e_2(I)$ that is included in n. Similarly, $h(n) \in e(I')$ iff $h(n) \in e_1(I')$ and there exists a region $n' \in e_2(I')$ that is included in $h(n)$. From the

induction step, $n \in e_1(I)$ iff $h(n) \in e_1(I')$, and $n' \in e_2(I)$ iff $h(n') \in e_2(I')$. From the definition of the reduction operation, the mapping preserves inclusion relationships. Thus $n \supset n'$ iff $h(n) \supset h(n')$. It follows that $n \in e(I)$ iff $h(n) \in e(I')$.

- Let $e = e_1 \subset e_2$. The proof is similar to that of $\supset$. The only difference is that now n' is a region that includes n (instead of being included in n as in the previous case).

- Let $e = e_1 < e_2$. The proof is based on the observation that every k-reduced version of I is also a k'-reduced version, for every $k' \leq k$. Let k_1, k_2 be the number of $<, >$ in e_1, e_2, resp. We have that $n \in e(I)$ iff $n \in e_1(I)$ and there exists a region $s \in e_2(I)$ where $R(n) < L(s)$. Similarly, $h(n) \in e(I')$ iff $h(n) \in e_1(I')$ and there exists a region $t \in e_2(I')$ where $R(n) < L(t)$. From the induction step, $n \in e_1(I)$ iff $h(n) \in e_1(I')$. Thus all we have to show is that an $s \in e_2(I)$ following n exists, iff there exists a $t \in e_2(I')$ following $h(n)$. Since I' is a k-reduced version of I, it follows from the definition of k-reduction that I' has a k_2-reduced[4] version I'' (with some mapping h_{k_2}), s.t. $n < s$ iff there exists $t \in h_{k_2}^{-1}(h_{k_2}(h(s)))$ s.t. $h(n) < t$ in I'. From the induction step $s \in e_2(I)$ iff $h(s) \in e_2(I')$ (since e_2 has less operations). Similarly, $h(s) \in e_2(I')$ iff $h_{k_2}(h(s))) \in e_2(I'')$. But $h_{k_2}(h(s))) \in e_2(I'')$ iff for every $w \in h_{k_2}^{-1}(h_{k_2}(h(s)))$, $w \in e_2(I')$. This is in particular true for the $w = t$ above that follows $h(n)$ in I'. Thus an $s \in e_2(I)$ following n exists, iff there exists a $t \in e_2(I')$ following $h(n)$. This implies that $n \in e(I)$ iff $h(n) \in e(I')$.

- The proof for $e = e_1 > e_2$ is symmetrical to the above one.

□

We can now 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 $e(I') \neq \emptyset$.

[4]Recall that a $(k-1)$-reduction is also a k'-reduction for all $k' \leq (k-1)$.

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 direct inclusion, nor simultaneously inclusion and order relationships. We motivate the practical importance of the kinds of queries considered in their respective subsections.

5.1 Direct Inclusion

Consider a file containing the source code of a program, with the structure described in Section 2.2. 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 $\mathsf{Proc} \supset \mathsf{Proc\text{-}body} \supset \sigma_{``x"}(\mathsf{Var})$, 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 a $\mathsf{Proc\text{-}body}$ 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 $\mathsf{Proc} \supset_d \mathsf{Proc\text{-}body} \supset_d \sigma_{``x"}(\mathsf{Var})$.

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 including operation. Let $\mathcal{I} = \{A, B, W\}$ be a region index schema, with a RIG containing the edges (A, B), (B, A), and (B, B). 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 10.

From Theorem 4.1, it follows that there exists a set of regions $S \subset I$ with nesting depth of 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

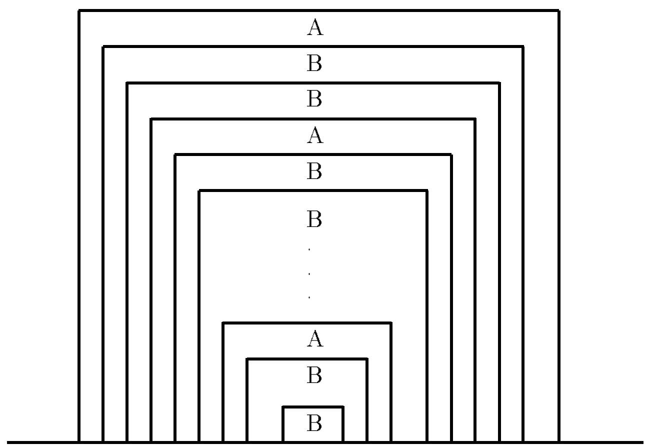

Figure 10: Counter-example for direct inclusion.

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.
□

The previous impossibility result is based on the fact that regions can be nested to an arbitrary depth. An implementation of direct inclusion for files with unbounded nesting, that makes use of a looping construct, is presented in Section 6.

If the nesting depth is guaranteed to be bounded by some constant, direct inclusion is directly expressible in the algebra. 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). Observe that if the RIG is acyclic, then the nesting depth of the regions in any region instance I satisfying the RIG is bounded by the length of the longest path in the RIG. To show that direct inclusion is expressible, we show that $Q \supset_d R$ is expressible for every two region sets $Q, R \subseteq I$. Note that Q and R may contain regions from different region names, thus may be nested. But since the nesting depth of regions in I is bounded, so is the nesting depth in Q and R.

First note that 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. This is because the sub-expression $(R - (R \subset (\bigcup_{T \in \mathcal{I}} T) \subset Q))$ discards from R regions that are not directly included in some Q, and then the full expression computes the Q regions containing such regions.

If Q is nested then the above expression fails, because some regions may be discarded unjustly from R. For example, assume we have one R region that is contained in two nested Q regions. In this case, the sub-expression $(R - (R \subset (\bigcup_{T \in \mathcal{I}} T) \subset Q))$ removes the R region because the inner Q separates the R region and the outer Q.

However, for Q with nesting bounded by k, one can use k algebra expressions to select regions in the i'th nesting layer, $1 \leq i \leq k$, then compute direct inclusion for each layer (using the above expression), and finally take the union of the results. The first (outermost) layer of Q regions is computed by the expression $L_1 = Q - (Q \subset Q)$, and the j-th layer, $j = 2 \ldots k$, is computed by $L_j = Q_j - (Q_j \subset Q_j)$, where $Q_j = Q - \bigcup_{i=1 \ldots j-1} L_i$.
$\square$

We next consider a restricted class of expressions called **inclusion expressions**. These are expressions that use only the $\supset$ and $\supset_d$ operations, or only the $\subset$ and $\subset_d$ operations. It was shown in [CM94] that this class, although seemingly very restricted, is very useful as a building block for efficiently computing high-level object-oriented queries on files, as well as supporting extended SQL language access to structured documents [BCK+94]. 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 time polynomial in the size of the expression by making use of the information provided by the RIG [CM94].

5.2 Both-Included

We have shown 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 simultaneously capturing inclusion and 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 of x precedes that of y. One may try to use the expression $\mathsf{Proc} \supset (\sigma_{``x"}\ (\mathsf{Var}) < \sigma_{``y"}(\mathsf{Var})\ )$. However, the previous expression 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 definition of y 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 supported by 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 document unit would be each procedure and the system would 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. Actually, users (and hence text retrieval systems) are not so much interested in precedence, but rather in *proximity*. However, the notion of proximity corresponds precisely to stating that elements either precede or follow each other within a certain distance of each other, and hence the ordering operations we consider are an appropriate abstraction to deal with proximity.

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

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

The expression $\mathsf{Proc}\ \ \text{BI}\ \ (\sigma_{``x"}(\mathsf{Var}),\ \sigma_{``y"}(\mathsf{Var}))$ computes exactly the example query discussed in this subsection.

Once more, we have a result that highlights an important limitation of the region algebra.

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

Proof: Let $\mathcal{I} = (A, B, C, W)$ be a region index, with a RIG containing the edges (C, A) and (C, B). Assume there is an algebra expression e computing $C\ \text{BI}\ \ (B, A)$. 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 11.

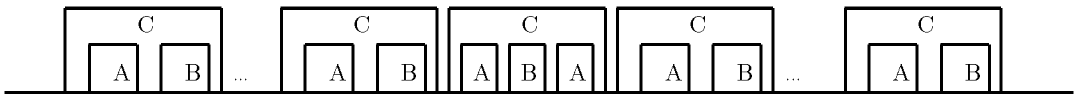

Figure 11: Counter-example for both-included.

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 $h_{I,I'}$ be the reduction mapping induced 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'', such that for every two regions $r, s \in I$, r follows (precedes) s in I iff there exists $t \in h^{-1}_{I',I''}(h_{I',I''}(h_{I,I'}(s)))$ s.t. t follows (precedes) $h_{I,I'}(s)$ 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 arguments follow the same lines as in the example of Section 4.2.

□

The preceding inexpressibility result relies on the fact that files can have an arbitrary number of disjoint 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 (which is similar to the special case for direct inclusion discussed in the previous subsection).

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

Proof: The proof is based on the observation that if R is a set of regions having one outermost region containing all the other regions, and S and T

are singleton sets, then

$$R \text{ BI } (S,T) = (R \supset (S < T)) \cap (R \supset T)$$

To compute R BI (S,T) in the general case, we first use k algebra expressions to select the k outermost regions of R. (There can be at most k such regions due to the bound on the number of disjoint regions.) For each such outermost R region we build a set containing the region and all the R regions that it includes. Then, we use k expressions to select the k innermost S regions, and correspondingly, k expressions for the k innermost T regions. (Again, there can be at most k such regions due to the bound on the number of disjoint regions.) Finally, we compute both-included for each the above subsets of R,S and T using the previously given expression, and we take the union of the results.

To be more specific, the j-th outermost R region, $j = 1 \ldots k$ is computed using the expression $O_j = R_j - (R_j < R_j)$, where $R_1 = R - (R \subset R)$, and $R_j = R_1 - \bigcup_{i=1\ldots j-1} O_i$, for $j = 2 \ldots k$.

The set containing each such outermost region and all the regions included in it are computed by $((\bigcup_{U \in \mathcal{I}} U) \subset O_j) \cup O_j$.

Finally, the q innermost S regions, $q = 1 \ldots k$, in each O_j are computed by $I_q = S_q - (S_q < S_q)$, where $S_1 = (S \subset O_j) - (S \supset (S \subset O_j))$, and $S_q = S_1 - \bigcup_{i=1\ldots q-1} I_i$, for $q = 2 \ldots k$. The q innermost T regions are computed by the same expression with S replaced by T.

□

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 (Theorem 4.4) still holds for the extended language. On the other hand, if only both-included is added, then the deletion theorem (Theorem 4.1) still holds for the extended language. The proofs of the two theorems, for the language extended with the respective operators, 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

In a previous work [CM94] we used the Pat system and its region algebra to provide high level access to semi-structured files. The idea was that the data stored in such files can be viewed as a database, or more specifically, an object oriented database [ACM93, CM94]. Queries on the database view of the file can then be formulated in OQL. To be executed, the OQL queries are translated to expressions in the region algebra and are then evaluated by the Pat system.

For example, assume we have a file containing several programs with the structure described in Section 2.2. Assume that we view the programs as database objects, and look for programs defining a procedure "proc1". In particular, we are interested in procedures that are defined directly by the program and not by other procedures. Our OQL query may have the form

$$\begin{aligned} Q \;=\; & \texttt{SELECT}\ p\ \texttt{FROM}\ Programs \\ & \texttt{WHERE}\ p.Proc.Proc{-}header.Name\ \text{contains "}proc1\text{"} \end{aligned}$$

In terms of regions, we are looking for programs that directly contain a procedure region directly containing a header region with a name containing the word *proc*1. This can be formulated as follows.

$$\mathsf{Prog} \supset_d \mathsf{Proc} \supset_d \mathsf{Proc} - \mathsf{header} \supset_d \sigma_w(\mathsf{Name})$$

However, as proved in the previous section, direct inclusion cannot be expressed by the algebra. So to support the above we had to extend the system functionalities. One possible way of supporting direct inclusion without modifying a region algebra engine (such as the Pat system) is to embed calls to the engine in a programming language. Once the algebra was embedded in a programming language having a while looping construct and assignment, the direct inclusion operation could be implemented. 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}

The program basically iterates over the nested layers of R_1 regions. In each iteration it selects the regions in one such layer (lines 1 and 7), picks those that directly include R_2 regions (using the expression presented in the proof of Proposition 5.2) and takes the union of the results (line 6). A similar approach can be used to compute both-included. In this case the iteration works from left to right on disjoint regions, and uses the expression in the proof of Proposition 5.4.

To compute the full path expression of the above OQL query, one needs to perform a sequence of direct inclusions. In general, when simulating OQL queries using the algebra, one may need to handle an expression of the form $e = R_1 \circ R_2 \circ \ldots \circ R_{n-1} \circ R_n$, where $\circ$ can be any one of $\supset_d$ or $\subset_d$. (In particular, $\subset_d$ is needed when simulating projection [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 entails a separate execution of the looping program given above. It turns out that this can be avoided, and in fact *only 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 fourth line of the program, and is heavily influenced by the size of the set. It turns out that this is another instance 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 a minimal set. Computing it is, however, expensive.

Given a region index schema $\mathcal{I}$, a RIG for $\mathcal{I}$, a direct inclusion expression $e = R_1 \supset \ldots \supset R_n$ over $\mathcal{I}$, and a constant k, we call the problem of testing if there is an $\mathcal{I}'$ with the above properties s.t. $|\mathcal{I}'| \leq k$, the **minimal set** problem.

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

Proof: The NP algorithm guesses $\mathcal{I}'$ of size k and checks that it satisfies the requirement of containing at least one region name on every path from R_i to R_{i+1}, $i = 1 \ldots n-1$, or that it removes from the RIG the chosen nodes and tests if R_{i+1} is still reachable from R_i.

The hardness proof is by reduction from the vertex cover problem known to be NP-complete. We show below that for every graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$ there exists a RIG $G' = (\mathcal{I}, E')$ and an inclusion expression e, s.t. G has a vertex cover of size $\leq k$ iff e has a set $\mathcal{I}'$ of size $\leq k+n-1$ that satisfies the above requirement. For a given graph $G = (V, E)$, we construct $\mathcal{I}$, the RIG G', and the expression e, in time polynomial in the size of G, as follows.

Let $\mathcal{I} = \{v_i, \hat{v}_i | v_i \in V\}$ and $G' = (\mathcal{I}, E')$ where

$$
\begin{aligned}
E' = \{&(\hat{v}_i, v_i), (v_i, \hat{v}_{i+1}) \mid v_i \in V\} \\
\cup\{&(\hat{v}_i, \hat{v}_i) \mid v_i \in V, i > 1\} \\
\cup\{&(v_i, v_j), (v_j, v_i) \mid (v_i, v_j) \in E\}
\end{aligned}
$$

and $e = \hat{v_1} \supset_d v_1 \supset_d \hat{v_2} \supset_d v_2 \ldots \supset_d \hat{v_n} \supset_d v_n$. Observe that according to this construction the node $\hat{v}_1$ in the RIG G' is not on any cycle in the RIG.

We first show that for every vertex cover $S \subseteq V$ of G, the region set $\mathcal{I}' = S \cup \{\hat{v}_i \mid v_i \in V, i > 1\}$ (whose size is $|S| + n - 1$) contains at least one region name on every path between two successive region sets in e. The proof is by contradiction. Assume this is not the case. Then there exists a path from $\hat{v}_i$ to v_i (or from v_i to $\hat{v}_{i+1}$) with no node in $\mathcal{I}'$. Observe that this cannot be a path from $\hat{v}_1$ to v_1 since there is only a single path with a single edge $\hat{v}_1 \rightarrow v_1$ between these two nodes. Also observe that the path cannot pass through any of the $\hat{v}_k$ nodes, $k > 1$, since they all belong to $\mathcal{I}'$. Thus the path must use at least one edge (v_i, v_j) where (v_i, v_j) or (v_j, v_i) is in E. But since S is a vertex cover, at least one of v_i, v_j is in S (and thus in $\mathcal{I}'$). A contradiction.

For the other direction we show that if $\mathcal{I}'$ contains at least one region name on every path between two successive region sets in e, then the set $S = \mathcal{I}' - \{\hat{v}_i \mid v_i \in V\}$ is a vertex cover for G with size $\leq |\mathcal{I}'| - n + 1$. For the size, observe that $\mathcal{I}'$ must contain all the nodes $\hat{v}_i$, $i > 1$ (otherwise the paths $\hat{v}_i \rightarrow \hat{v}_i \rightarrow v_i$ are not covered). Thus S's size $\leq |\mathcal{I}'| - n + 1$. Now, if S is not a vertex cover, then there is some edge $(v_i, v_j) \in E$ not covered by S. But this implies that $\mathcal{I}'$ contains no vertex (not including the endpoints) on the path $v_i \rightarrow v_j \rightarrow v_i \rightarrow \hat{v}_{i+1}$ (recall that E' is define so that it contains all edges $\{(v_i, v_j), (v_j, v_i) \mid (v_i, v_j) \in E\}$). This is a contradiction for $\mathcal{I}'$ satisfying this requirement.

$\square$

7 Future Work

We conclude by discussing two possible extensions to the algebra. The region algebra studied in the previous sections can be viewed as a special restricted variant of the relational algebra:

(1) We have three base domains, the booleans, the strings, and *reg*, the domain of regions, with the operations $\subset, \supset, <, >$ of signature $reg, reg \rightarrow bool$ for testing the inclusion and precedence relationship among regions.

(2) We have only two types of relations. 1-ary relations over the domain of regions (representing the region names and the region sets constructed as intermediate results in the query computation), and a single 2-ary relation with one string attribute and one region attribute (representing the word index).

(3) The relational algebra is restricted to allow only (i) semi-join operations (using $\subset, \supset, <, >$ as the join condition) on the 1-ary region relations, and (ii) on the 2-ary relation, selection by equality on the string attribute, followed immediately by projection on the region attribute.

This algebra is implemented in the commercial PAT system and, as shown in the previous sections, can be optimized. However, it lacks expressiveness. This is due to two factors. First, only unary region relations are allowed. Second, the textual 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 (since the number of selection operators is bounded by the length of the expression), and joins according to the content of regions are not supported.

To extend the capabilities of the language, one may extend (2) and (3) above by supporting joins (using $\subset, \supset, <, >$ as the join condition) and not only semi-joins, and allowing n-ary relations over the region domain as the result of such joins.

Note that expressions in this extended language correspond to safe FMFT formulas (where safety of formulas is defined in the standard way [Ull88, AHV95]). This is because the structure of an input region instance does not change and 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 now be expressed directly by this extended language.

To be able to fully explore the content of regions in query expressions (for example, by allowing joins that compare the contents of regions), joins with the 2-ary relation representing the word index should also be allowed. The price to be paid for this additional extension is that the encoding into FMFT formulas no longer works, so the optimization results of the previous sections cannot be used.

Acknowledgments

The authors would like to thank Stephane Grumbach, Pekka Kilpelainen, Gonzalo Navarro, and Frank Tompa for discussions and comments related to the contents of this paper.

The first author was financially supported in part by the Department of Computer Science at the University of Waterloo and by Grant CRD 147259 from the Natural Sciences and Engineering Research Council of Canada. This work was done while the second author was at the University of Toronto, and supported by the Institute for Robotics and Intelligent Systems.

References

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

[ACM95] S. Abiteboul, S. Cluet, and T. Milo. A database interface for file update. In *Proc. of SIGMOD 95*, pages 386–397, 1995.

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

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. Foundations of databases. *Addison-Wesley*, 1995.

[BCK+94] G. E. Blake, M. P. Consens, P. Kilpeläinen, P.-Å. Larson, T. Snider, and F. W. Tompa. Text / relational database management systems: Harmonizing SQL and SGML. In *Applications of Databases (Proceedings of the ADB'94 Conference)*, pages 267–280. Springer-Verlag Lecture Notes in Computer Science Nr. 819, 1994.

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

[BLCL+94] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, and A. Secret. The world-wide web. *CACM*, 37(8):76–82, August 1994.

[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.

[EBG96] E. Graedel E. Boerger and Y. Gurevich. The classical decision problem 1996. xii, (perspectives in mathematical logic). *Springer-Verlag*, 1996.

[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.

[KM93] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proc. of the 15th. SIGIR Conference*, 1993.

[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.

[Ope95] Open Text Corporation. *Open Text 5 System Integration Guide and Database Administration Guide*, 1995.

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

[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.