Nested Text-Region Algebra

Jani Jaakkola* Pekka Kilpeläinen

Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
e-mail: Pekka.Kilpelainen@cs.helsinki.fi

Abstract

So called region algebras operating on sets of text fragments have been proposed and implemented as query languages for text documents. Text documents often comprise nested regions like lists within lists or procedures within procedures. Earlier versions of region algebra do not support querying nested regions. We address this deficiency by proposing a new, unrestricted region algebra. The new algebra allows dynamic definition of nested regions. This makes it suitable for querying without any preprocessing documents, whose hierarchical structure is indicated by embedded markup. We demonstrate that this nested region algebra can be efficiently implemented, by presenting and analyzing algorithms for its operations. Especially, we show that any fixed nested region algebra expression on text of length n can be evaluated in the worst case in time $O(n^2)$, and in practice in linear time. Nested region algebra has been implemented in a publicly available Unix text search tool called sgrep.

1 Introduction

Textual documents in electronic form – office documents, large text corpora, software source code and documentation, Web pages – are ubiquitous. It is of great importance to be able to access information buried within them.

A simple yet effective way of modeling text for retrieval is to view it as contiguous fragments, e.g., as chapters, paragraphs and pages. We call these fragments *regions*. Operations for retrieving regions based on their relative ordering and containment provide a way of retrieving document components based both on their content and their surrounding context.

The PAT text search system gave an efficient implementation of a two-sorted query algebra of text positions and regions. PAT precludes overlapping regions in the evaluation of expressions by presenting them as the start positions of the regions. This causes semantic problems, since it is not possible to determine statically from a query weather it returns regions or positions [17].

*Supported by the Academy of Finland.

Burkowski overcame the semantic problems of PAT expressions in a fully compositional algebra that works purely on sets of regions [4, 5]. A further reworking of that algebra by Clarke, Cormack and Burkowski accepts overlapping regions, but nested results are precluded by restricting to the innermost regions of a set [7, 8]. Precluding nested results ensures that intermediate results are of linear size with respect to the length of the text. Also, sets of non-nesting regions can be linearly ordered either by their start or end positions, which makes efficient implementations easy.

We consider restrictions to the overlapping or nesting of regions harmful. Nested or recursive structures are common, e.g., in SGML documents [11, 10] and in programming languages. Consider the following simplified example. Assume that we want to locate the blocks of a C program, which contain an occurrence of an identifier x. (See Figure 1.)

```
{ /* 1st block */ x,y ... { /* 2nd block */ }
                          { /* 3rd block */ x ... } ... }
```

Figure 1: Locating program blocks containing "x".

Assume that we can refer to the occurrences of the opening brace by $\mathcal{I}(''\{'')$, to the occurrences of the closing brace by $\mathcal{I}(''\}'')$, and to the occurrences of the identifier by $\mathcal{I}(''\mathrm{x}'')$. Then, using the notations of the query algebra to be explained later, we would locate the wanted-for program fragments by the following expression:

$$(\mathcal{I}(''\{'')..\mathcal{I}(''\}'')) \triangleright \mathcal{I}(''\mathrm{x}'')$$

The expression is read intuitively as "Form the regions between opening and closing braces, and select those of them, which contain an occurrence of the string x". According to the non-nested semantics of the earlier region algebras the query would return only the third block in Figure 1. We feel that also the outer-most block should be returned, since it also contains an "x". We should also be able to locate the first block by the condition that it contains an occurrence of "y". This is not supported by the previous versions of region algebra.

The nested region algebra presented in this paper provides the semantics argued above by a careful redefinition of the "followed-by" ("..") operator. Nested region algebra has been implemented in a Unix text search tool called sgrep [13, 12]. The model and the tool do not set any limits on the length, overlapping or nesting of the regions. Further, the implemented tool is not restricted to any specific file formats since it does not make any assumptions about the encoding of the text structure, and it does not require any preprocessing of the text.

The rest of the paper is organized as follows. We define the data model and the operations of nested region algebra in Section 2. Then we describe and analyze the evaluation of region algebra queries in Section 3. Values of expressions can be represented as ordered lists. Most operations can be evaluated in linear time by algorithms, which traverse their operands in synchronization. In a few cases we need to sort the region lists or consider same regions for multiple times, which increases the asymptotic time of evaluating region algebra expressions on a text of length n to $O(tn)$, where $t \leq n$ is the maximum number of regions overlapping at any single position. In practical situations t is limited by a constant. In the concluding Section 4 we discuss some problems and prospects which would deserve further study.

2 Nested region algebra

We describe first the data model of nested region algebra. Then we define the operations of nested region algebra, and discuss some of their properties.

2.1 Region sets

The base of the nested region algebra is a finite universe $\mathcal{U} = \{1, \ldots, n\}$ of n points called *positions*. The positions could be, e.g., indices of indexed words [7], or start and end positions of arbitrary substrings of a file [13]. A *start position* x with $1 \leq x \leq n$ and an *end position* y with $x \leq y \leq n$ define a contiguous non-empty interval of positions $\{x, x+1, \ldots, y\}$. We call such an interval a *region* and denote it by $[x, y]$. We use a notation $r.\alpha$ for the start position x of a region $r = [x, y]$, and $r.\omega$ for its end position y. In practice, the regions could be meaningful fragments of text, like sentences, paragraphs, or sections.

The positions of the universe span the set of all regions $\mathcal{R} = \{[x, y] \mid 1 \leq x \leq y \leq n\}$. Notice that $\mathcal{R} \subseteq \mathcal{P}(\mathcal{U})$, and that $|\mathcal{R}| = n(n+1)/2$ while $|\mathcal{P}(\mathcal{U})| = 2^n$. Operators of region algebra operate on subsets of $\mathcal{R}$, which we call *region sets*. Often the regions of a region set share some common property. For example, they are often occurrences of a string pattern, or document elements of the same type or at the same depth in the document hierarchy.

The operations for manipulating region sets described in the next section are based on conditions on the relative containment and ordering of regions. Let $a = [a.\alpha, a.\omega]$ and $b = [b.\alpha, b.\omega]$ be two regions. The regions can be compared using normal set inclusion. If $b.\alpha \leq a.\alpha$ and $a.\omega \leq b.\omega$, we write $a \subseteq b$ and say that the two regions are *nested*. If a is properly contained in b, we may also write $a \subset b$. Similarly, if region a includes region b, we use notation $a \supseteq b$, and if the inclusion is proper, we may write $a \supset b$. If $a.\omega < b.\alpha$, i.e., region b begins after the end of region a, we say that region a *precedes* region b, region b *follows* region a, and the regions are *disjoint*. If two regions are neither nested or disjoint, they are *overlapping*. If a region set contains nested regions, we say that it is *nested*, and otherwise we say that it is *flat*.

We define two total orders for sets of regions. The *start position order* denoted by $<_\alpha$ is the lexicographic order of (α, ω), i.e., for any regions a and b

$$a <_\alpha b \text{ if and only if } a.\alpha < b.\alpha, \text{ or } a.\alpha = b.\alpha \text{ and } a.\omega < b.\omega \;.$$

Similarly, the *end position order* denoted by $<_\omega$ is the lexicographic order of (ω, α), i.e., for any regions a and b

$$a <_\omega b \text{ if and only if } a.\omega < b.\omega, \text{ or } a.\omega = b.\omega \text{ and } a.\alpha < b.\alpha \ .$$

Notice that for a flat region set the start position order and the end position order are identical. Any nonempty set of regions $A \subseteq \mathcal{R}$ has both a *minimum* and a *maximum* element with respect to each of the two orders:

$$\begin{aligned}
\min_\alpha A &= \{a \in A \mid \neg\exists a' \in A : a' <_\alpha a\}, \\
\max_\alpha A &= \{a \in A \mid \neg\exists a' \in A : a <_\alpha a'\}, \\
\min_\omega A &= \{a \in A \mid \neg\exists a' \in A : a' <_\omega a\}, \text{ and} \\
\max_\omega A &= \{a \in A \mid \neg\exists a' \in A : a <_\omega a'\} \ .
\end{aligned}$$

2.2 Nested region algebra expressions

The expressions of nested region algebra have the following syntax:

$$\begin{array}{lll}
E & \rightarrow (E) & \\
 & \mid E \lhd E \mid E \not\lhd E & \text{\% Containment} \\
 & \mid E \rhd E \mid E \not\rhd E & \text{\% expressions, Sect. 2.2.1} \\
 & \mid E \cup E \mid E \cap E \mid E \setminus E & \text{\% Set expressions, Ex. 2.1} \\
 & \mid E..E & \text{\% ``Followed-by'', Sect. 2.2.2} \\
 & \mid E \ \mathit{quote} \ E & \text{\% Sect. 2.2.2} \\
 & \mid \mathit{hull}(E) & \text{\% Sect. 2.2.3} \\
 & \mid E \ominus E & \text{\% Sect. 2.2.4} \\
 & \mid \mathcal{I}(p), \text{ where } p \text{ is a pattern} & \\
 & \mid \{[\alpha_1, \omega_1], \ldots, [\alpha_k, \omega_k]\}, & \\
 & \quad 1 \leq \alpha_i \leq \omega_i \leq n & \text{\% Constant region set}
\end{array}$$

We assume that there exists an indexing function $\mathcal{I}(p)$, which for a given text pattern p returns the set of the regions, which are occurrences of p in the text being queried. Different implementations can provide different pattern languages. Constant region sets allow referring to fixed sets of regions with fixed start and end positions.

Next we present the operations of nested region algebra. Each operator takes one or two region sets as its operands and produces a new region set as its result. This means that arbitrarily complex queries can be composed of the operators.

In what follows, let $A \subseteq \mathcal{R}$ and $B \subseteq \mathcal{R}$ be regions sets if not mentioned otherwise.

2.2.1 Containment operators

All variants of region algebra provide similar operators for choosing regions of set A on the basis of their containment relation with regions of set B. The containment operator are

1. $A \lhd B$ ("A in B"), which selects the regions of A that are included in some region of set B,
2. $A \not\lhd B$ ("A not in B"), which selects the regions of A that are not included in any region of set B,
3. $A \rhd B$ ("A containing B"), which selects the regions of A that contain some region of set B, and
4. $A \not\rhd B$ ("A not containing B"), which selects the regions of A that do not contain any region of set B.

The formal definitions these operators are given below.

$$\begin{aligned}
A \lhd B &= \{a \in A \mid \exists b \in B : a \subset b\} \\
A \not\lhd B &= \{a \in A \mid \neg\exists b \in B : a \subset b\} \\
A \rhd B &= \{a \in A \mid \exists b \in B : a \supset b\} \\
A \not\rhd B &= \{a \in A \mid \neg\exists b \in B : a \supset b\}
\end{aligned}$$

Example 2.1 *The ordinary set theoretic operators $A \cup B$, $A \cap B$, and $A \setminus B$ for the union, intersection and difference of regions sets are sometimes useful, but except for union they are not necessary for expressing text retrieval queries.*

Assume that D is a set of regions representing documents of interest. Then we can express standard Boolean retrieval operations as follows:

1. *Fetch documents that mention "cat" OR "dog":*

$$D \rhd (\mathcal{I}(''cat'') \cup \mathcal{I}(''dog'')), \ or \ (D \rhd \mathcal{I}(''cat'')) \cup (D \rhd \mathcal{I}(''dog''))$$

2. *Fetch documents that mention both "cat" AND "dog":*

$$(D \rhd \mathcal{I}(''cat'')) \rhd \mathcal{I}(''dog''), \ or \ (D \rhd \mathcal{I}(''cat'')) \cap (D \rhd \mathcal{I}(''dog''))$$

3. *Fetch documents that mention "cat" but do NOT mention "dog":*

$$(D \rhd \mathcal{I}(''cat'')) \not\rhd \mathcal{I}(''dog''), \ or \ (D \rhd \mathcal{I}(''cat'')) \setminus (D \rhd \mathcal{I}(''dog''))$$

Also structure-based queries can be expressed using containment operators. Assume that region set T contains the regions of document titles . Then the following query would retrieve the titles of the documents that mention cats in the body of the document but not in the title:

$$T \lhd (D \rhd (\mathcal{I}(''cat'') \not\lhd T))$$

□

Arbitrarily complex search conditions can be formed by combining containment operators in the manner shown above.

2.2.2 Producing regions from their boundaries

Text fragments are often determined by their bounding regions, which are normally indicated my some visible markup like quotation marks, parentheses, or multi-character tags. We classify such fragments into *quotation-like* and *nested fragments*. Quotation-like fragments are disjoint, and can thus be delimited at the beginning and at the end by the same mark. Examples of quotation-like fragments are string constants in many programming languages. Nested fragments, on the other hand, can contain each other. If nested fragments are visually marked-up, their beginning and their end have to use different markup. Familiar examples of nested fragments are nested parentheses and nested document elements.

We present operators for producing both nested and quotation-like fragments from their bounding regions. We start by discussing nested fragments. Consider the regions delimited by parentheses in the below string:

$$(1\ (2(3))\ (4))$$

A human reader recognizes four substrings surrounded by a balanced pair of parentheses: "(1 (2 (3)) (4))", "(2 (3))", "(3)", and "(4)". We derive the semantics of an operator for forming such nested region sets by formalizing this principle of forming regions by matching delimiters together "from inside out". The operator is denoted by $A..B$, and read "A followed by B". For example, we should get the above substrings as the value of the expression $\mathcal{I}(''('')..\mathcal{I}('')'')$.

We define that the value of $A..B$ is

1. $\emptyset$ if $\neg\exists a \in A, b \in B : a.\omega < b.\alpha$;
2. else $\{[a.\alpha, b.\omega]\} \cup ((A \setminus \{a\})..(B \setminus \{b\}))$, where $a \in A$ and $b \in B$ are such regions that $a = \max_\omega\{a' \in A \mid a'.\omega < b.\alpha\}$ and $b = \min_\alpha\{b' \in B \mid b.\alpha > a.\omega\}$.

The definition means that result regions span from the beginning of a region of A to the end of the closest following unmatched region in B.

Example 2.2 *Figure 2 is a depiction of the value $A..B$ when $A = \{a_1, a_2, a_3, a_4\} = \{[1,2],[3,6],[4,6],[6,7]\}$ and $B = \{b_1, b_2, b_3\} = \{[5,8],[7,9],[7,10]\}$. The regions of each set are shown from up to down in the start position order. Region $a_1 = [1,2]$ is the closest preceding region for $b_1 = [5,8]$, so they produce the result region $r_1 = [1,8]$. Region $a_2 = [3,6]$ precedes region $b_2 = [7,9]$, but so does region $a_3 = [4,6]$, which is greater in the end position order. b_2. For this reason regions a_3 and b_2 are chosen to produce the result region $r_3 = [4,9]$. Regions a_2 and $b_3 = [7,10]$ are closest together among the remaining regions. They produce the result region $r_2 = [3,10]$. Finally, region $a_4 = [6,7]$ does not produce any result region since it does not precede any region of B.* □

Notice that each input region may either participate in the production of at most one result region or it may fail to produce any result region. Therefore it holds in general that $|A..B| \leq \min\{|A|, |B|\}$.

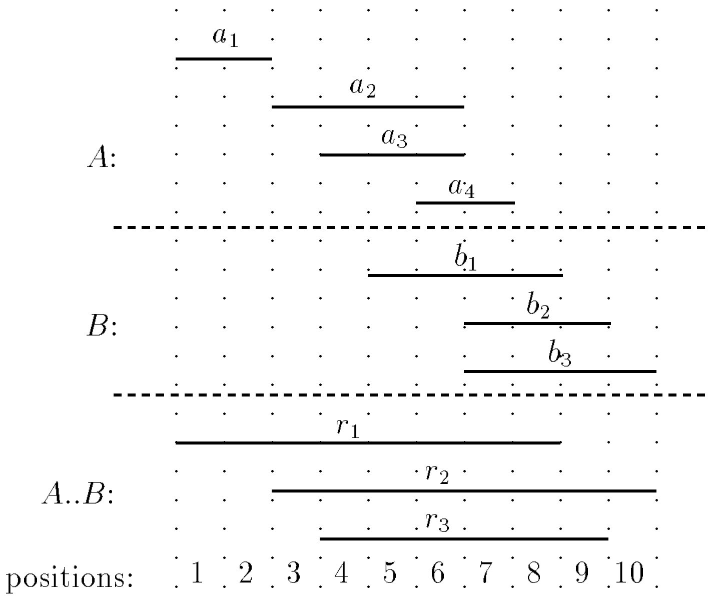

Figure 2: An example of "A followed by B".

Example 2.3 *In XML [3], the start of any non-empty document element is denoted by a* start-tag *of the form* <GI ...>*, which contains the so called* generic identifier GI *of the element, possibly followed by additional* attributes*. The end of an element is denoted by an* end-tag *of the form* </GI>*. Empty elements having no content are denoted by an empty element tag of the form* <GI .../ >*. The region set of all non-empty elements in well-formed XML documents can be expressed by the following expression:*[1]

$$((\mathcal{I}("<") \not\triangleleft \mathcal{I}("</"))..(\mathcal{I}(">") \not\triangleleft \mathcal{I}("/>"))..(\mathcal{I}("</")..\mathcal{I}(">"))$$

The sgrep search tool supports shorthand notations for expressions called macros*. Given suitable definitions for macros STAG and ETAG for the expressions retrieving start tags and end tags of non-empty elements, the above query would be expressed as*

$$STAG..ETAG\ .$$

□

The .. operator is not suitable for retrieving quotation-like fragments which begin and end by the same marker. Consider the following fragment of the LaTeX [14] source of this article:

```
regions $a$ and $b$
```

[1] We deliberately simplify the syntax of XML documents here. A complete query should also exclude left angle brackets appearing in *markup declarations*, *comments*, *processing instructions*, or *CDATA sections*.

Let us see how we would retrieve the regions surrounded by dollar signs, which LaTeX uses to delimit in-line formulas. The value of $\mathcal{I}(''\$'')..\mathcal{I}(''\$'')$ contains three regions: `$a$`, `$ and $`, and `$b$`. This is because the "followed-by" operator simply matches each region with the closest following region, when the operand sets are equal. For this reason we introduce another combination operator A *quote* B, which produces disjoint regions by combining the first region a of A with the first region b of B that follows a, and by continuing similarly with the regions of A and B that follow b. This means, e.g., that the value of $\mathcal{I}(''\$'')$ *quote* $\mathcal{I}(''\$'')$ above consists of the intended two regions, `$a$` and `$b$`. The formal definition of A *quote* B is

1. $\emptyset$ if $\neg\exists a \in A, b \in B : a.\omega < b.\alpha$;
2. else $\{[a.\alpha, b.\omega]\} \cup (\{a' \in A \mid a'.\alpha > b.\omega\}$ *quote* $\{b' \in B \mid b'.\alpha > b.\omega\})$, where $a = \min_\alpha A$ and $b = \min_\alpha\{b' \in B \mid b'.\alpha > a.\omega\}$.

Notice that *quote* always produces a flat region set whose regions are disjoint.

Example 2.4 *Since the tags of XML documents do not nest within each other, the macros alluded to in Example 2.3 could be defined as follows using the quote operator:*

$$
\begin{aligned}
TAGC &= (\mathcal{I}(''>'') \not\triangleleft \mathcal{I}(''/>''))\\
STAG &= (((\mathcal{I}(''<'') \not\triangleleft \mathcal{I}(''</'')) \textit{ quote } TAGC)\\
ETAG &= (\mathcal{I}(''</'') \textit{ quote } TAGC)
\end{aligned}
$$

□

2.2.3 The hull of a region set

It is convenient to have a simple representation for the positions covered by a region set, which may contain nesting and overlapping regions. Such a presentation is provided by the *hull* of a region set. The *hull of* A is the minimal set of regions that covers the positions of the regions in set A. Function $hull(A)$ produces the hull of set A. We define it as a function $hull : \mathcal{P}(\mathcal{U}) \rightarrow \mathcal{R}$ with

$$
\begin{aligned}
hull(A) = \{[x,y] \in \mathcal{R} \quad | \quad & [x,y] \subseteq \bigcup_{a \in A} a \ \wedge \\
& x-1 \notin \bigcup_{a \in A} a \ \wedge \ y+1 \notin \bigcup_{a \in A} a\}\ .
\end{aligned}
$$

That is, $hull(A)$ consists of the maximal regions which comprise positions covered by members of A. Therefore it is the minimal region set that covers the same positions as A. Notice that $hull(A)$ is always a flat region set, and that any two regions of $hull(A)$ are disjoint.

The hull is utilized in the implementation of some of the other operators. Also, sgrep returns by default the hull of the result as a minimal representation of the regions retrieved by the query.

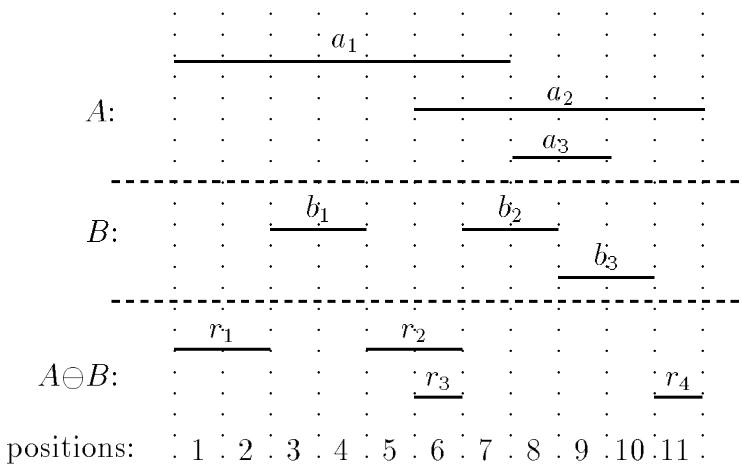

Figure 3: The result of $A \ominus B$.

2.2.4 Extracting regions

The containment expressions allow us to retrieve fragments of text that satisfy some condition on their content or on their context. It is sometimes useful to be able exclude from the result some smaller fragments that satisfy some other condition. This can be done using operator $A \ominus B$ ("A extracting B"), whose value consists of the result of removing from the regions of A any overlap with any region of B. The operator is defined by

$$A \ominus B = \bigcup_{a \in A} \mathit{hull}(\{a \setminus \bigcup_{b \in B} b\}) \ .$$

Notice that each region of A may produce several regions of $A \ominus B$. For each $a \in A$, the difference $a \setminus \bigcup_{b \in B} b$ is a subset of $\mathcal{U}$ and $\mathit{hull}(\{a \setminus \bigcup_{b \in B} b\})$ is the set of regions which result by cutting off from a any overlap with the regions of B.

Example 2.5 *Figure 3 is a depiction of the value of* $A \ominus B$*, when* $A = \{a_1, a_2, a_3\} = \{[1,7], [6,11], [8,9]\}$ *and* $B = \{b_1, b_2, b_3\} = \{[3,4], [7,8], [9,10]\}$*. The result consists of regions* $r_1 = [1,2]$*,* $r_2 = [5,6]$*,* $r_3 = [6,6]$*, and* $r_4 = [11,11]$*.* □

Each region of $A \ominus B$ may inherit either of its end points from a region a of A, or it may equal a "gap" $g = [x, y]$, where $\{x-1, y+1\} \subseteq \bigcup_{b \in B} b$ and $g \cap \bigcup_{b \in B} b = \emptyset$. Since B can have at most $|B| - 1$ such gaps, there can be at most $2|A| + |B| - 1$ regions in $A \ominus B$. This bound is also tight, as can be seen in Figure 4.

Example 2.6 *The textual content of XML files, i.e., the content without embedded markup tags can be retrieved using the following query:*

$$(\{[1,1]\}..\{[n,n]\}) \ominus (\mathcal{I}(''<'')..\mathcal{I}(''>''))$$

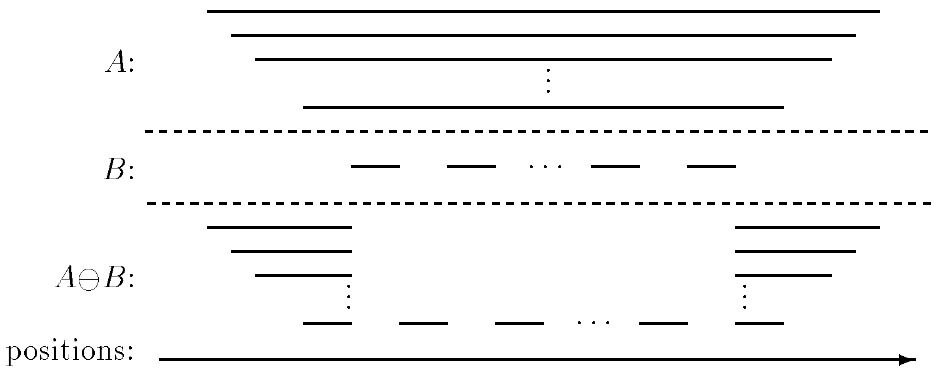

Figure 4: $A \ominus B$ can contain $2|A| + |B| - 1$ regions.

□

2.3 Properties of expressions

We have found the operators of nested region algebra useful for retrieval and simple manipulation of structured text documents [13]. We saw in Example 2.1 that containment operators together with union are sufficient for expressing standard Boolean queries. We will finish this section by stating some simple algebraic properties of region expressions.

It is obvious that the containment operators are not commutative, since their operands have a different role: the regions of left-hand operand are selected on the basis of their relation to the regions in the right-hand operand. It is also obvious that "followed-by" and *quote* are not commutative either, since they treat regions of the left-hand operand as region start-delimiters and regions of the right-hand operand as region end-delimiters.

The operands are also not associative in general. The following inclusions can be easily derived from the definition of the containment operators and from the transitivity of region containment:

$$\begin{aligned} A \lhd (B \lhd C) &\subseteq (A \lhd B) \lhd C & (1) \\ A \rhd (B \rhd C) &\subseteq (A \rhd B) \rhd C & (2) \\ (A \lhd B) \rhd C &\subseteq A \lhd (B \rhd C) & (3) \\ (A \rhd B) \lhd C &\subseteq A \rhd (B \lhd C) & (4) \end{aligned}$$

The following equivalences between containment and set expressions are also straightforward to check:

$$\begin{aligned} A op\ (B \cup C) &\equiv (A op\ B) \cup (A op\ C) \text{ for } op \in \{\lhd, \rhd\} & (5) \\ A \not\lhd B &\equiv A \setminus (A \lhd B) & (6) \\ A \not\rhd B &\equiv A \setminus (A \rhd B) & (7) \end{aligned}$$

$$(A\,op_1 B)\,op_2 C \quad \equiv \quad (A\,op_1 B) \cap (A\,op_2 C) \text{ for } op_1, op_2 \in \{\triangleleft, \not\triangleleft, \triangleright, \not\triangleright\} \qquad (8)$$

Equation 8 implies that the order of evaluating containment conditions does not matter, i.e.,

$$(A\,op_1 B)\,op_2 C \equiv (A\,op_2 C)\,op_1 B \ ,$$

when $op_1, op_2 \in \{\triangleleft, \not\triangleleft, \triangleright, \not\triangleright\}$.

3 Evaluating nested region expressions

In this section we show that expressions of nested region algebra can be evaluated efficiently, by presenting and analyzing algorithms for each its operations.

We assume that the expressions are presented as expression trees, whose leaves stand for either constant region sets or calls of the indexing function $\mathcal{I}(p)$ for some text pattern p. Internal nodes of expression trees are labeled by operators of the algebra. An expression tree can be evaluated in an arbitrary bottom-up order; each internal node can be evaluated once its subtrees have been evaluated.

The evaluation of region algebra expressions is based on representing region sets by ordered lists. Region sets have to be considered in the evaluation of expressions both in the start position order and in the end position order. To avoid sorting, each region set is presented by a pair of lists (S, E), where S is a *start position list* which is kept in increasing start position order, and E is an *end position list* which is kept in increasing end position order. We call such pairs of lists *(S,E) lists*. For the algorithms we assume that a given region in an (S,E) list can be updated and replaced through either S or E in constant time. Figure 5 shows a possible data structure for implementing (S,E) lists.

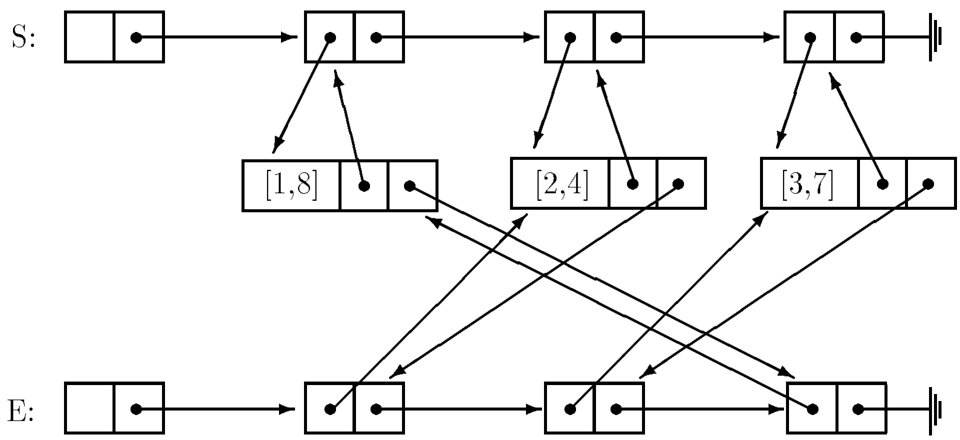

Figure 5: A possible data structure for accessing region set $\{[1,8],[2,4],[3,7]\}$ in both start position and end position order.

The evaluation is initiated by attaching (S,E) lists to the leaves of the expression tree. An (S,E) list for a constant region set $\{[\alpha_1, \omega_1], \ldots, [\alpha_k, \omega_k]\}$ can obviously be constructed in constant time $O(k)$. If $\mathcal{I}(p)$ is evaluated by performing a lookup from an index which has been constructed off-line, it can be implemented, applying standard techniques like balanced trees or binary search, to return an (S,E) list in time $O(\log n + m)$, where n is the length of the indexed text and m is the number of occurrences of p. $\mathcal{I}(p)$ can also be implemented by performing a pattern matching scan of the text, returning a set of non-nesting occurrences in order, say, from left to right. It is well known that such pattern matching can be performed for a wide class of patterns, including regular expressions, in time $O(|p|n)$. (See, e.g., [1] and [6]).

The union of two region sets can be implemented simply by merging their S and E lists. An algorithm for implementing the intersection of region sets represented by (S, E) lists L and R can be based on a synchronized traversal of, say, the start position lists $L.S$ and $R.S$. A region that appears in both $L.S$ and $R.S$ is marked accepted, and the result (S,E) list is formed by collecting the accepted regions in order through both $L.S$ and $L.E$. An algorithm for set difference is similar, once we switch the roles of accepted and rejected regions in $L.S$. So, all set operations can clearly be implemented in time which is linear in the size of the input region sets.

Containment operations are rather similar to set operations, but slightly more complicated to evaluate. We discuss algorithms for them in the next section. The algorithms assume constant-time implementations of primitive list operations, like adding elements and accessing the first element, the successor of an element, or the last element in a list.

3.1 Evaluating containment operators

Figure 6 presents algorithm *In* for evaluating $A \lhd B$. Region sets A and B are passed to the algorithm via its formal parameters L and R, which are (S,E) lists. The algorithm is simplified by noticing that $A \lhd B \equiv A \lhd (B \not\lhd B)$, i.e., it is sufficient to consider the outermost regions of B, which form a flat region set (line 1). A list containing the regions of $B \not\lhd B$ can be computed by function *Outer* in linear time by a straightforward scan of the regions of B in start position order. During the scan, we know that a region is included in some other region of B, and shall thus be discarded, if and only if its end position does not exceed the largest end position of its preceding regions.[2]

The main body of algorithm *In* consists of a loop (lines 4–17), which goes in start position order through the regions in both A, stored in (S,E) list L, and $(B \not\lhd B)$, stored in list R'. At each iteration a region a of A and a region b of B are compared, and one of them is replaced by its successor, until either list runs out (line 4). So, clearly the loop terminates in time $O(|A| + |B|)$.

The body of the loop comprises a case-analysis of the relationship between regions a and b. If a is contained in b, the record corresponding to a is marked

[2] The sgrep implementation of nested region algebra provides $outer(A) = A \not\lhd A$ and $inner(A) = A \not\rhd A$ as language primitives.

Algorithm: In(L,R).

Input: (S,E) lists L and R representing region sets A and B.

Output: An (S,E) list representing $A \triangleleft B$.

```
R':= Outer(R.S);
a:= First(L.S); % current region of A
b:= First(R'); % current region of B
while a ≠ null and b ≠ null do
    if a ⊂ b then
        Mark a accepted;
        a:= Next(a, L.S);
    elsif a.α ≤ b.α then
        % a.α < b'.α for all b' ∈ outer(B) s.t. b <_α b'
        a:= Next(a, L.S);
    elsif b ≠ Last(R') then
        b':= Next(b,R'); % b'.ω > b.ω
        if b'.α ≤ a.α then
            b:= b';
        else
            a:= Next(a, L.S);
        fi;
    else % b = Last(R')
        if b.ω > a.α then
            a:= Next(a, L.S);
        else
            b:= Next(b, R'); % = null
        fi;
    fi;
od;
Collect Result.S from accepted regions in L.S;
Collect Result.E from accepted regions in L.E;
return Result;
```

Figure 6: An algorithm for evaluating $A \triangleleft B$.

Algorithm: Inner(L).

Input: List L $= [b_1, \ldots, b_m]$ of regions of a set B in increasing $<_\alpha$ order.

Output: A list of the regions of $B \not\triangleright B$ in increasing $<_\alpha$ order.

if $m \leq 1$ **then return** L;
elsif $b_1 \subset b_2$ **then return** Inner($[b_1, b_3, b_4, \ldots, b_m]$);
else
    R:= Inner($[b_2, \ldots, b_m]$);
    % Let R $= [r_1, \ldots, r_k]$; now $b_1.\alpha < b_2.\alpha \leq r_1.\alpha$
    if $b_1 \supset r_1$ **then**
        return R;
    else
        return $[b_1, r_1, \ldots, r_k]$;
    fi;
fi;

Figure 7: An algorithm for computing the innermost regions of B.

accepted (lines 5 and 6). If $a \not\subset b$ and a does not start after the start of region b, it cannot be included in any of the remaining regions of B (lines 8–9). Otherwise, region a begins after the start of region b. If b has a successor b' (lines 10–11), which could include a (line 12) – and thereby also any of the successors of a that might be included in current b – we replace b by b' (line 13). If region a cannot be included in b', it cannot be included in any later region of B either (line 14). Finally, when b is the last region of *outer*(B), there are two possibilities remaining. Either region b ends after the start of a, in which case some of a's possible successors could be included in b (lines 15–16). Otherwise the last region (i.e., b) of B precedes any of the remaining regions of A, and the loop can be terminated (line 17).

At the end, the result (S,E) list is formed by collecting the marked regions through both L.S and L.E in their original order (lines 18–20), which can be done in time $O(|A|)$.

Algorithm *Containing* for evaluating $A \triangleright B$ presented in Figure 8 is in principle similar to algorithm *In*. This time the evaluation is simplified by noticing that $A \triangleright B \equiv A \triangleright (B \not\triangleright B)$. Figure 7 presents algorithm *Inner*, which computes a list containing the value of $(B \not\triangleright B)$ in time $O(|B|)$, assuming that the lists are implemented in a manner, which allows them to be passed as parameters by references, and the head element of a list to be replaced in constant time. It is rather easy to see that algorithm *Containing* computes the value of $A \triangleright B$ in time $O(|A| + |B|)$.

Notice that the algorithms for $A \triangleleft B$ and $A \triangleright B$ can straightforwardly be modified to evaluate their complementary operations $A \not\triangleleft B$ and $A \not\triangleright B$, by

Algorithm: Containing(L,R).

Input: (S,E) lists L and R representing region sets A and B.

Output: An (S,E) list representing $A \triangleright B$.

```
R':= Inner(R.S);
a:= First(L.S); % current region of A
b:= First(R'); % current region of B
while a ≠ null and b ≠ null do
    if a ⊃ b then
        Mark a accepted;
        a:= Next(a, L.S);
    elsif a.ω ≤ b.ω then
        % a ⊅ b' for all b' ∈ (B ⊅ B) s.t. b <_α b'
        a:= Next(a, L.S);
    else % b.α < a.α and b.ω < a.ω,
        % thus a' ⊅ b for all a' ∈ A s.t. a <_α a'
        b:= Next(b,R');
    fi;
od;
Collect Result.S from accepted regions in L.S;
Collect Result.E from accepted regions in L.E;
return Result;
```

Figure 8: An algorithm for evaluating $A \triangleright B$.

swapping the cases where the current region a is either accepted to the result or rejected from the result. So, we have seen that an (S,E) list representation of the value of containment operations can be computed in linear time, given the (S,E) lists for their operand sets.

3.2 Computing combinations of region sets

Figure 9 presents algorithm *FBY* for computing the value of $A..B$. Again, the algorithm manipulates region sets A and B as (S,E) lists L and R, respectively. The algorithm performs the matching of regions by storing the regions of A that precede the first un-matched region b of B in *end position order* in a stack (lines 1, 3, and 5–7). The closest unmatched region r of A for b is then found from the top of the stack (line 9). The algorithm utilizes the existing order in the lists L.S and R.E, by replacing by the result region [r.α, b.ω] both region r in L.S (line 10) and current region b in R.E (line 12). If no matching start region was found for b, but A contains regions still unconsidered (line 14), regions of B are skipped until the first one following the current region a of A (line 15).

It is easy to see that the main loop can be executed in time $O(|A| + |B|)$.

The algorithm forms the result (S,E) list by scanning accepted regions from lists L.S and R.E in their original order, using procedures *StartCollect* and *EndCollect* (lines 16–17). The newly created result regions in L.S are in increasing start position order, unless they have a common start position. Result regions with a common start position may form in L.S, though, contiguous sublists of regions, whose end positions updated on line 10 violate the start position order. Procedure StartCollect sorts such sublists in start position order, and removes any duplicates. Similarly, any sublists of result regions in R.E not in end position order are sorted by procedure EndCollect (line 17).

Assume that *StartCollect* (or *EndCollect*) needs to sort l sublists having lengths $k_1, \ldots, k_l$. This requires time $O(\sum_{i=1}^{l} k_i \log k_i) = O(m \log k)$, where $m = |A..B| = \min\{|A|, |B|\}$, and $k = \max\{k_1, \ldots, k_l\}$. Sorting is only needed for sublists of regions with either a common start or end position. Large sets of regions with a common start or end position can be formed by $A \ominus B$, when set A contains a deep chain of nested regions. (See Figure 4 on page 10.) A slightly looser upper bound for the time to collect the result regions in algorithm *FBY* is $O(m \log t)$, where the *thickness* of $A..B$ defined by $t = \max_{x \in \mathcal{U}} |\{r \in A..B \mid x \in r\}|$ is a measure of maximum overlap among the regions in $A..B$. If the regions created by the expressions correspond to meaningful elements of structured documents, t is limited by a constant. For example, the maximum nesting that we have seen in the SGML document collections that we have encountered is of 16 elements. Therefore we argue that in practise algorithm FBY works in linear time, as opposed to its theoretical $O(n \log n)$ worst case behavior.

The operation A *quote* B is straightforward to evaluate in linear time by traversing both sets A and B in start position order. For region $a = \min_{\alpha} A$ the first $b \in B$ which follows a is located, and the process continues similarly after locating the first $a \in A$ that follows this b. The result regions $[a.\alpha, b.\omega]$ do

Algorithm: FBY(L,R).

Input: (S,E) lists L and R representing region sets A and B.

Output: An (S,E) list representing $A..B$.

```
a:= First(L.E); % current region of A
b:= First(R.S); % current region of B
Empty(Stack);
until (a = null and IsEmpty(Stack) or b = null) do repeat
    while a ≠ null and a.ω < b.α do
        Push a on Stack;
        a:= Next(a, L.E);
    od;
    % a = null or a.ω ≥ b.α
    if not IsEmpty(Stack) then
        r:= Pop(Stack);
        r.ω:= b.ω;
        Mark r accepted;
        Replace b in R.E by r;
        b:= Next(b, R.S);
    elsif a ≠ null then % b.α ≤ a.ω
        while b ≠ null and b.α ≤ a.ω do b:= Next(b, R.S);
    fi;
endrepeat;
Result.S:= StartCollect(L.S);
Result.E:= EndCollect(R.E);
return Result;
```

Figure 9: An algorithm for evaluating $A..B$.

Algorithm: Hull(L).

Input: (S,E) list L representing a region set A.

Output: An (S,E) list representation for the hull of A.

```
1.    Result.S:= Result.E:= null;
2.    L':= Outer(L.S);
3.    a:= First(L');
4.    while a ≠ null do
5.        e:= a.ω;
6.        b:= Next(a,L');
7.        while b ≠ null and b.α ≤ e+1 do
                % b.ω > e
8.            e:= b.ω;
9.            b:= Next(b,L');
          od;
10.       Append(Result.S, [a.α, e]);
11.       Append(Result.E, [a.α, e]);
12.       a:= b;
      od;
13.   return Result;
```

Figure 10: An algorithm for computing $hull(A)$.

not overlap and can thus be added in both the start position list and the end position list of the result in the order they are created.

3.3 Computing the hull

Figure 10 presents algorithm *Hull* for computing the hull of a region set A. The algorithm is simplified by noticing that $hull(A) = hull(A \not\triangleleft A)$. Therefore it suffices to consider a list L' consisting of the outermost regions of A (line 2), which form a flat region set. The algorithm consists of two nested loops. The outer loop (lines 4-12) is executed once for each *block*, which is a sublists of L' consisting of overlapping or adjacent regions. For each block, a new region is created in the result (lines 10–11). After locating the initial region a of a block, the inner loop (lines 7–9) locates the subsequent regions b of the block. Since the regions in L' are in both in increasing start position order and in increasing end position order, it is sufficient to keep track of the end position of the last region as the current end position e of the block (line 8).

It is easy to see that algorithm *Hull* considers each region of A once, and requires therefore time $O(|A|)$.

3.4 Computing extraction

Figure 11 presents algorithm *Extract* for computing the value of $A \ominus B$. Region sets A and B are represented similarly to earlier algorithms as (S,E) lists L and R, respectively, which are scanned in increasing start position order. This time the evaluation is simplified by noticing that $A \ominus B = A \ominus hull(B)$ (line 2).

The region traversal policy of algorithm *Extract* differs from the previous ones, since now we may need to consider for each region $a \in A$ several suffixes that remain after cutting off its overlap with regions $b \in hull(B)$. The algorithm stores these suffixes in a list Sufs (line 17) in (not necessarily strictly) increasing order of their start positions, which we call *relaxed start position order*. The algorithm considers regions of A and their suffixes in relaxed start position order, by choosing among the unconsidered regions in lists L.S and Sufs the next one, whose start position is smaller (lines 9 and 18). Ties are broken by choosing the region whose end position is smaller. If there are duplicates, the region in L.S is chosen.

Producing an (S,E) list representation of $A \ominus B$ would at first seem to require sorting of the result regions. For example, the order of the result regions in Figure 4 (on page 10) would seem to differ quite a lot from the order of the input regions. A closer examination reveals that the order of the result regions is determined by the order of start and end positions of the input regions. We take advantage of this as follows. There are three ways how result regions of $A \ominus B$ can be created. First, they can be original regions of A, which do not intersect any region of B. These regions are selected to the result, similarly to earlier algorithms, by marking them accepted and then scanning them through (S,E) list L. Clearly the mutual start position and end position orders between these regions are the same as in the original list L. Second, a result region can be a fragment of a region a of A, which precedes the overlapping part of a that has been cut off by a region b of $hull(B)$ (line 13). Since the algorithm considers both regions of A and their suffixes, and regions of $hull(B)$ in increasing order of their start positions, these new "fragment regions" are created both in increasing start position order and in increasing end position order. These regions are collected in list NewRegs. Notice that NewRegs may contain duplicates, even though it is otherwise in order.

Third, a result region of $A \ominus B$ may be a *suffix* of a region a of A, which inherits its end position from a but whose start overlaps with some region of B. Such suffix regions inherit their mutual end position order from the original end position order of A. To see this, consider two regions r_1 and r_2 in $A \ominus B$, such that $r_1 = [b_1.\omega + 1, a_1.\omega]$ and $r_2 = [b_2.\omega + 1, a_2.\omega]$ for some $a_1, a_2 \in A$ and $b_1, b_2 \in hull(B)$. If $a_1.\omega \neq a_2.\omega$, then r_1 and r_2 are clearly in the same end position order as a_1 and a_2. On the other hand, if $a_1.\omega = a_2.\omega$, then $b_1 = b_2$ and thus $r_1 = r_2$ by the definition of $A \ominus B$. This means that while there may be duplicates among the result regions which are suffixes of original regions of A, they are otherwise mutually in their original end position order. For this reason the algorithm creates such new regions in the L.E list at the place of the corresponding original region a (lines 16 and 7). Regions r inserted in list Sufs are implemented as pointers to the actual record for the region, which is held in list L.E. This way later updates to these regions, when accessed through list Sufs, are reflected to the copy held in list L.E.

At the end the algorithm forms the result by first collecting in list NewSufs regions of L.E marked "accepted suffixes" (line 19). Since these regions are not nested, the end position order of L.E is also a start position order for NewSufs. Also the regions in NewRegs are in both start position order and in end position order. These lists are merged with the accepted regions of L.S in start position order (line 20), and with the accepted regions of L.E in end position order (line 21), to form the result (S,E) list. Some care has to be taken to ensure that the S and E list of the result end up sharing common regions, as depicted in Figure 5. Also notice that any duplicates possibly introduced in lists NewRegs and NewSufs need to be removed during the merging of the lists.

Obviously the work done by algorithm *Extract* is proportional to the number of regions scanned in the loop on lines 4–18. The loop considers each region of A and B at most once, but it may append a new region to list Sufs for each pair of regions $(a, b) \in A \times B$. (Consider for example the situation of Figure 4.) Therefore the algorithm requires in the worst case quadratic time $O(|A||B|)$. On the other hand, for each region $b \in hull(B)$ the algorithm creates at most t regions in list Sufs, where t is the thickness of region set A. This gives a worst-case time bound of $O(|A| + t|B|)$. As we argued in Section 3.2, the thickness of meaningful region sets on existing document collections is limited by a constant. Therefore we conclude that also algorithm *Extract* works in practical situations in linear time.

4 Discussion

We saw in Section 3 that any fixed nested region algebra expression E can be evaluated on text of length n in time $O(tn)$, where $t \leq n$ is the maximum thickness of the region sets arising in the evaluation of E. We argued that with sensible queries on real document collections t is limited by a constant, and therefore nested region algebra queries can be evaluated in practise in linear time.

Nested region algebra has been implemented in a publicly available search tool called sgrep. The sgrep implementation has been well received as a practical text search tool, e.g., to implement searches on XML files [15]. The algorithms and data structures used in sgrep differ slightly from those of Section 3. Especially, it would be rather space-consuming to implement large region sets as (S, E) lists, which require six pointers for each region. The data structure used in sgrep is a dynamic chain of fixed-size blocks, each capable of storing 128 regions. The region lists are maintained in start position order only. The sgrep implementations of algorithms *FBY* and *Extract* sort regions in those rare situations when it is needed.

Navarro and Baeza-Yates [16] have presented a related *proximal nodes model* for querying document databases. The design of the proximal nodes model was

Algorithm: Extract(L, R).

Input: (S,E) list L and R representing region sets A and B.

Output: The (S,E) list representation of $A \ominus B$.

```
EmptyList(Sufs); EmptyList(NewRegs);
R':= Hull(R);
a:= First(L.S); b:= First(R'.S);
while a ≠ null do
    if b = null or a.ω < b.α then
        if a is marked "suffix" then
            Mark a "accepted suffix";
        else
            Mark a "accepted";
        fi;
        a:= Next(a, L.S, Sufs);
    elsif b.ω < a.α then
        b:= Next(b, R'.S);
    else % a ∩ b ≠ ∅
        if a.α < b.α then
            Append([a.α, b.α-1], NewRegs);
        if b.ω < a.ω then
            r:= [b.ω+1, a.ω]; Mark r "suffix";
            Replace(a, L.E, r);
            Append(r, Sufs);
        fi;
        a:= Next(a, L.S, Sufs);
    fi;
od;
NewSufs:= regions marked "accepted suffix" in L.E;
Result.S:= merge of NewRegs, NewSufs, and regions marked "accepted" in L.S
        in start position order;
Result.E:= merge of NewRegs, NewSufs, and regions marked "accepted" in L.E
        in end position order;
return Result;
```

Figure 11: An algorithm for computing $A \ominus B$.

driven by similar efficiency concerns than the design of nested region algebra. Similarly to nested region algebra, proximal node expressions can be evaluated by traversing operand sets in synchronization, which leads to linear evaluation times. The proximal nodes model is based on multiple explicitly stored hierarchies. Nested region algebra is more primitive and therefore more widely applicable in the sense that it does not assume any fixed hierarchies. This aspect has been proven useful by the sgrep implementation, which allows complex structure-based queries be expressed and evaluated by dynamically recognizing hierarchies implicit in marked-up documents.

Some queries are inherently difficult to express in set-based region algebras. Consens and Milo have considered a static version of nested region algebra, which contains no operations for creating new regions from existing ones. They have shown that in static region algebra one cannot express relations of *direct inclusion* (e.g., finding which of nested procedures contain a defining occurrence of a variable) and *"both included"* (e.g., locating procedures, which contain a definition of variables x and y, in this order) [9]. Direct inclusion may be impossible to express in our dynamic region algebra, too, even though the proof technique of Consens and Milo does not apply to the dynamic case. On the other hand, "both included" can be expressed in dynamic nested region algebra. For example, assume that the bodies of procedures are surrounded by braces and that a variable definition starts with word "var" and ends with a semicolon. Then the following query would locate procedures, which contain a definition of variable "x" followed by a definition of variable "y".

$$\begin{array}{l}(\mathcal{I}(''\{'')..\mathcal{I}(''\}'')) \rhd \\ (\ ((\mathcal{I}(''\mathrm{var}'')..\mathcal{I}('';'')) \rhd \mathcal{I}(''\mathrm{x}''))..((\mathcal{I}(''\mathrm{var}'')..\mathcal{I}('';'')) \rhd \mathcal{I}(''\mathrm{y}''))\ )\end{array}$$

On the other hand, the above example illustrates also the difficulty of expressing direct inclusion. The query extracts procedures which contain the required variable definitions, but it appears to be very difficult to restrict it just to those procedures, which would contain the definitions *directly.*

The proximal nodes model offers more complex primitives based on relations between explicit nodes, and therefore it offers more powerful query primitives than nested region algebra. With this respect it is remarkable that the "both included" relation cannot be expressed in the proximal nodes model [16].

The proximal nodes model of Navarro and Baeza-Yates does not allow overlapping hierarchies in results of expressions. Our work is in this respect an extension to previous query models: We have shown that both nesting and overlaps can be supported without essentially compromising algorithmic efficiency. It might be worthwhile to study how far we could extend the nested region algebra, say, towards the more powerful proximal nodes model, while simultaneously preserving the model efficiently evaluateable.

Allen has presented an exhaustive list of 13 relations between temporal intervals a and b of time [2]. The list contains relations like "a starts at the same time but ends earlier than b", "a starts later but ends at the same time as b", "a precedes b", and "a starts when b ends". Allen's relations include also the plain containment, which is considered as the central primitive in region algebras. It is not clear, which others of Allen's relations would be useful as primitives of region selection queries. In any case, adding them as alternative selection operators to region algebra would be independent of its other features. Such alternative selection operators could easily be implemented as simple variations of the presented algorithms for containment expressions.

References

[1] A. V. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science – Volume A: Algorithms and Complexity*, chapter 5. Elsevier/MIT Press, 1994.

[2] J.A. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[3] T. Bray, J. Paoli, and C.M. Sperberg-McQueen, editors. *Extensible Markup Language (XML) 1.0.* W3C Recommendation 10-February-1998. The latest version is available at `http://www.w3.org/TR/REC-xml`.

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

[5] F. J. Burkowski. Retrieval activities in a database consisting of heterogeneous collections of structured text. In N. Belkin, P. Ingwersen, and A. M. Pejtersen, editors, *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–125. ACM Press, June 1992.

[6] C. L. A. Clarke and G. V. Cormack. On the use of regular expressions for searching text. *ACM Transactions on Programming Languages and Systems*, 19(3):413–426, May 1997.

[7] 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*, 38(1):43–56, 1995.

[8] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. Schema-independent retrieval from heterogeneous structured text. In *The Fourth Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, April 1995. Available as a technical report from `ftp://cs-archive.uwaterloo.ca/cs-archive/CS-94-39`.

[9] M. P. Consens and T. Milo. Algebras for querying text regions. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 11–22, California, 1995.

[10] C. F. Goldfarb. *The SGML Handbook.* Claredon Press, Oxford, 1990.

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

[12] J. Jaakkola. *Sgrep home page*. University of Helsinki, Department of Computer Science, April 1996. `http://www.cs.helsinki.fi/~jjaakkol/sgrep.html`.

[13] J. Jaakkola and P. Kilpeläinen. Using sgrep for querying structured text files. In J. Saarela, editor, *Proceedings of SGML Finland 1996*. SGML Users' Group Finland, October 1996. Available as Dept. of Computer Science Report C-1996-83, University of Helsinki, 1996.

[14] L. Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Englewood Cliffs, New Jersey, 1986.

[15] M. Leventhal, D. Lewis, and M. Fuchs. *Designing XML Internet Applications*. Prentice Hall Inc., Englewood Cliffs, New Jersey, 1998.

[16] G. Navarro and R. Baeza-Yates. Proximal nodes: A model to query document databases by content and structure. *ACM Transactions on Information Systems*, 15(4):400–435, October 1997.

[17] A. Salminen and F. Wm. Tompa. PAT expressions: an algebra for text search. In *COMPLEX'92*, pages 309–332, 1992. Also technical report number OED–92–02, UW Centre for the New Oxford English Dictionary and Text Research.