Vojkan Mihajlović
Djoerd Hiemstra
Henk Ernst Blok
Peter M. G. Apers

Utilizing Structural Knowledge for Information Retrieval in XML Databases

Centre for Telematics and Information Technology (CTIT)
Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS)
University of Twente

Contents

Utilizing Structural Knowledge for Information Retrieval in XML Databases

Vojkan Mihajlović Djoerd Hiemstra Henk Ernst Blok Peter M. G. Apers
CTIT, University of Twente
P.O. Box 217, 7500AE Enschede, The Netherlands
{v.mihajlovic, d.hiemstra, h.e.blok, p.m.g.apers}@utwente.nl

Abstract

In this paper we address the problem of immediate translation of eXtensible Mark-up Language (XML) information retrieval (IR) queries to relational database expressions and stress the benefits of using an intermediate XML-specific algebra over relational algebra. We show how adding an XML-specific algebra at the logical level of a DBMS enables a level of abstraction from both query languages for information retrieval in XML and the underlying physical storage and manipulation. We picked a region algebra as a basis for defining the structure aware (SA) view on XML in which we can distinguish among different XML entities, such as element nodes, text nodes, words, and determine their containment relation. Region algebras are already well established in semi-structured document processing as shown in an extensive overview of region algebra approaches in this paper. Furthermore, we propose a variant of region algebra that can support ranking operators in an elegant way while staying algebraic. As relevance scores are computed for regions in our region algebra we named it score region algebra (SRA). The benefits of introducing score region algebra are explained on a set of query examples. Besides abstracting from the query language used and the physical implementation, SRA enables a certain degree of abstraction from the retrieval model used and the opportunity to use the query optimization at the logical level of a database. Various retrieval models can be instantiated at the physical level based on the abstract specification of SRA operators. We also discuss numerous region algebra operator properties that provide a firm ground for query rewriting and optimization at the SA level, which is an important premise for the existence of such a logical view on XML.

1 Introduction

Despite the numerous existing systems developed for XML querying, the problem of expressing as well as executing information retrieval like (IR-like) queries over XML collections is still an open issue [2, 17]. An IR-like query (an example is given in Figure 2 in Section 3) does not specify hard conditions on XML elements, but queries the collection on elements 'about' a certain topic. For instance, an XML element that is relevant to a query for elements about "relational databases" might not contain the phrase "relational databases", or even both words "relational" and "databases". IR-like queries should result in a ranked list of XML elements, in decreasing order of some score value the system assigns to each element. The score value has to reflect the probability of relevance of the element to the IR-like query.

A promising approach to executing XPath [4] and XQuery [5] is the use of relational database technology [18, 19, 39], which can be easily extended to IR-like querying of XML [13, 25]. However, there are still some open questions, such as: what would be the most effective way to support IR-like querying in XPath and XQuery using relational database technology? The semantics of XPath and XQuery [10, 12] give rules for navigation through XML structure, but not the rules that specify how score values for XML elements should propagate and relate to one another. Similarly,

the semantics of relational algebra introduce rules for manipulating relational tables that describe XML data, but again the rules for score computation and propagation cannot be derived from the relations present in the relational database.

We follow a three-level database approach for developing an XML-IR system [22, 38], consisting of conceptual, logical, and physical level. However, we specify the logical level in such a way that it is capable of capturing the structure and semantics of the specific data representation such as XML and enable ranked retrieval. We named it therefore *structure aware (SA)* logical level. The benefits of the usage of a multiple level database management system is that we are able to provide data independence between the storage and data representation at the physical level, structure aware algebra at the logical level that captures the properties of data representation stored into database, and the query language used at the conceptual level. Furthermore, by specifying the algebra in a right way we provide a certain level of abstraction from the information retrieval model used for ranked retrieval of XML elements.

The algebra we propose is based on so-called region algebras, for example, [6, 8, 23, 26]. We extended the region algebra approaches to model XML more precisely and to support ranked retrieval of XML fragments. To enable modeling of XML documents we introduced new attributes in the region data model. We included the name of the region (i.e., the name of XML nodes, content words, etc.), the information about the type of a region, and the information representing the relevance score of a given region. The relevance score reflects the importance of the retrieved region (XML element) to a query specified by a user. We named the algebra *score region algebra (SRA)*.

The advantages of SRA over other region algebra approaches is that it is closed in the domain (of SRA regions) and that it supports orthogonal definition of retrieval model with respect to the algebra operators. Unlike many approaches for ranked retrieval in XML, the algebra we define assumes the ranking is a part of the algebra and not a side effect of performing some operations on regions (like in [26]) or a separate IR module (like in many IR approaches for XML retrieval). Therefore, we follow the approach taken by Fuhr et al. [15, 16], although we base our algebra on containment model rather than path model, and do not make any restrictions on defining the retrieval model.

Score region algebra is sufficiently simple to study algebraic properties in depth and it is sufficiently powerful to express IR-like queries as those proposed in the NEXI query language used for the evaluation of XML retrieval in INEX [37]. NEXI stands for "narrowed extended XPath". It only uses the descendant axis step from XPath, and it extends XPath with a special *about*-function that provides IR-like search (see [37] for details). However, the score region algebra can be easily extended to support other XPath axis steps with extra parent information [28]. The basic idea behind the algebra is to support as much as possible for the full-text search requirements [7] (although we consider only the core of the algebra in this paper) and it is driven by the wish to integrate XML databases and information retrieval as discussed in [2, 14].

Introducing an intermediate level with score region algebra in it allows for the usage of algebraic properties for query rewriting and optimization, specific to the data representation used, i.e., XML in our case, using all the available information in such data representations. The optimization should be achieved not only for the regular XPath/XQuery queries but for the IR-like queries as well. Therefore, by defining the algebra in such a way we have the opportunity to use the optimization methods not just for basic region algebra operators, but for the scoring region algebra operators as well. This allows for the introduction of more powerful optimization techniques concerned with speeding-up the execution of operations that compute score values for ranked retrieval. In this paper we study the usefulness of algebraic properties for query optimization and for developing and understanding IR-like extensions.

The paper is organized as follows. We provide an exhaustive overview of a number of region algebra approaches and give comparison of their features in Section 2. In Section 3 we explain how relational technology is used to process XPath (NEXI) queries. We give the translation of queries into relational algebra and discuss why we need a different intermediate level. Section 4 introduces our region algebra and discuss basic region algebra operator properties. In Section 5, we define scoring operators in SRA and their properties, and discuss the opportunities for query

optimization in our region algebra extended for ranked retrieval. Section 6 introduces a possible instantiation of a retrieval model at the physical level using SRA expressions at the logical level. We conclude the paper with a discussion and our plans for future research.

2 Region Algebra

We start this section by specifying some of the terms we are using in this paper. We continue by explaining characteristics and drawbacks of previous region algebra approaches, and conclude with the comparison of these region algebra approaches.

2.1 Region Algebra Basics

The term data model is used used to define a collection of entities (data structures) and their contained fields, along with the operations or functions that manipulate them[1]. The collection of entities in most of the region algebra approaches represents a set of regions, specified by starting and end position pairs for each region, denoted as s and e, where $e \geq s$. Region sets are organized in different concepts (such as concordance list [6] or generalized concordance list [8]) which bring some constrains on how these sets can be formed or what can be the result of the application of region algebra operators to operands.

To ease the comprehension of region algebra approaches in this section, we first stress some differences that exists in numerous region algebra approaches discussed in the following subsections. The differences are based on the formation of the initial data set on which the algebraic operators are executed, and on the specific domain on which these operators are executed.

The initial data set, following the chosen data model, is defined based on an indexing process formally specified (or described) in each region algebra paper. Firstly, the initial data set can be precomputed, i.e., the initial data set is established before any of the algebra operations is performed. In that case we say that the data set is *predefined*, and the algebra has to provide a fetch operator which selects a region or a number of regions form the data set, using the *token* as a parameter in the fetch operator (e.g., $fetch(token)$). Secondly, if the data set is not predefined it is *dynamically created* during the query execution. The data set is created using an explicitly defined indexing operator (e.g., $I(pattern)$), where the search string (or a character) is specified as a parameter of an indexing operator (i.e., *pattern*).

The initial data set in some cases defines the final (complete) data set, i.e., the domain on which all operators can be executed. The application of region algebra operators results in a set of regions which are either identified by some predefined entity (i.e., predefined region set), or consists of regions which are present in those entities (i.e., a subset of regions in a region set). As the creation of new regions is not allowed we named this data model as *static data model*. If the creation of new region sets (new regions) is allowed in the algebra, the operators are not closed in predefined domain of the initial data set. The application of algebraic operators on region sets can create new regions that are not in the predefined data model (if any). Thus, we named this data model as *dynamic data model*.

To summarize, we can distinguish between two data models if we consider the data set creation process: (1) static data models, and (2) dynamic data models. Dynamic data models can be either with the predefined (initial) data set or implicitly defined, i.e., defined during run time using an "indexing" operator. However, static data models must be predefined as the data model does not allow the creation of new entities in the algebra.

2.2 Region Algebra Approaches

Region algebra approaches we address in this paper are: PAT expressions, concordance lists, generalized concordance lists, proximal nodes, Consens/Milo region algebra, nested region algebra,

[1]Based on the definition of data model given in the W3C glossary: http://www.w3.org/TR/2002/PR-DOM-Level-2-HTML-20021108/glossary.html.

text constraints, and ranked region algebra. The basic idea behind region algebra approaches is the representation of text documents as a set of 'extents', where each extent is defined by its starting and end position. The application of the idea of text extents to XML documents is straightforward. If we regard each XML document instance as a linearized string or a set of *tokens* (including the document text itself), each component can then be considered as a text region or a contiguous subset of the entire linearized string. This property, as well as some other present in region algebra approaches give us the reason for studying region algebra approaches for their application to ranked retrieval in XML.

2.2.1 PAT expressions

The earliest work on region algebras is presented in [34]. The PAT system [34] was developed to combine traditional search techniques with a new search facilities, such as lexical search, proximity search, contextual search, and Boolean search, as well as for handling structured text documents. In PAT, the initial region set in the data model is not predefined. It can be defined by modeling text as *character string*, where each character is an elementary unit, or as an *indexed string*, where each token bounded with delimiters is considered to be an elementary unit. The result of the evaluation of any PAT expression can be either a *match point* set (character or pattern positions in a string) or a *region set* defined by a starting and end match points.

Among a number of operators PAT includes operators for expressing containment relations between regions: including, not including, within; as well as set operators for forming set union, set difference, and set intersection. Due to the distinction between match points and regions in the data model, expressions can be formed in PAT that produce unexpected results (especially if the first operand is a region set and the second one is a match point set), which can be seen in the original paper [34]. Thus, in the later region algebra approaches the distinction between match points and regions has never been incorporated into the data model.

2.2.2 Concordance lists

The aim of the region algebra introduced by Burkowski [6] is to preserve the data independence in text dominated databases with hierarchical structure. Burkowski developed the containment model, which can serve as a foundation for structured text databases, as the relational model serves as the foundation for relational databases.

The algebra is defined on a text collection represented as finite sequence of words, where Burkowski identified tree basic entities: words, text elements, and contiguous extents. A *text element* is defined as a sequence of *contiguous words* that have a semantic meaning (e.g., identified with mark-up). Each sub sequence of consecutive words represents a *contiguous extent* defined by the position of the starting word in the sequence and the position of the end word in the sequence. A data model is defined on a set of contiguous extents, statically formed (predefined), which are used to denote the positions of words and text elements, specified using mark-up or another kind of document annotation.

In the algebra, two contiguous extents can be either nested or disjoint. Burkowski's model also introduces a *concordance list* as being a named list[2] of starting and end position pairs that specify disjoint contiguous extents. Concordance lists are used for managing positions and nesting properties of the static contiguous extents predefined in the data model.

Algebraic expressions can be specified using the so-called retrieval command string (RCS). RCS is defined as a sequence of filter operations that specify which contiguous extents should be selected or rejected based upon containment tests: select narrow (SN), select wide (SW), reject narrow (RN), and reject wide (RW). Those operations are performed on concordance lists, where the left operand represents a concordance list which is filtered by a set of concordance lists that represent the second operand. For example, a search on "chapters" that contain terms "high" or "level"

[2]Although the author uses the "names" for the concordance lists, those names are not identified as a part of the data model. They are used only for fetching the proper concordance list (set of contiguous extents).

can be expressed as: `CHAPTER SW {``high'',``level''}`. Therefore, in Burkowski's model the interpretation of the "," in the query is a Boolean OR, or a concordance list union operator in the algebra. The author does not introduce a Boolean AND (concordance list intersection) operator as the search on "chapters" that contain terms "high" and "level" can be expressed like: `CHAPTER SW {``high''} SW {``level''}`.

Furthermore, RCS syntax contains some additional operators for supporting relevance ranking, such as cardinality, sublist, and length operators. Relevance ranking is done on-the-fly and is based on a type of *tf.idf* model (for details see [6]). Additionally, a ranking vector was introduced to store the ranking values for contiguous extents in a concordance list. The ranking vector can be used for ranked retrieval of contiguous extents. However, the binding between those vectors and contiguous extents is not clearly defined in the paper. Note that the introduced ranking operators, as well as ranking vector definition, violate the contiguous extent data model, since the result of the mentioned operators (stored in ranking vector) is not a contiguous extent, but a real or natural number.

2.2.3 Generalized concordance lists

To enable expressions on overlapped contiguous extents, as well as Boolean operators and operators for proximity search, Clarke et al. [8] loosen some constrains of Burkowski's model. The data model is defined based on five alphabets: index alphabet, markup alphabet, database alphabet, text alphabet, and stop-list alphabet. The index alphabet is a union of two disjunctive alphabets, markup and database alphabet, while the database alphabet is defined as a union of two disjunctive alphabets, text and stop-list alphabet.

In an indexing process the authors made a decision to index only text alphabet and markup alphabet. To be able to distinguish between the two indexed alphabets, authors made a distinction between the result domains of indexing functions for two alphabets. The result domain of the application of indexing function on the text alphabet is the domain of positive natural numbers. Each number represents relative position of the word in a document consisting of only words that are in the text alphabet (preserving their relative order from the original document). The result domain of the application of indexing function on the markup alphabet is the domain of positive rational numbers and the number is assigned in such a way that the mark-up alphabet symbols, denoting the beginning and the end of a structured element that begins and ends on the same word from text alphabet, can be clearly identified. For example, `<speaker>witch</speaker>` is indexed as: `<speaker>(n), witch(n), </speaker>(n+1/2)`. For more details see [8].

The algebra is defined on a set of extents specified using starting and end positions, called *generalized concordance list* (GC-list). The initial GC-list is obtained after indexing, and it defines extent bounds. A GC-list is introduced to model the non-nested extents, while the overlapping of extents in a GC-list is not allowed in the model. The algebra preserves the four basic operators from the original model of Burkowski, which are in the paper termed contained in, containing, not contained in, not containing, and explicitly introduces two combination operators, "both of"and "one of", similar to Boolean AND and OR operators, and an ordering operator "followed by". The result GC-list of the application of "one of" operator is a set of non-nested extents that contain either extents from the first or extents from the second operand, while the result GC-list of the application of "both of" operator is a set of non-nested extents that contain at least one extent from both operands (i.e., GC-lists). The "followed by" operator is defined as a concatenation of the closest extents in two operands.

Although the model is similar to the model presented in [6], there is an elementary difference between them. The difference is in the formation of concordance lists, since in Burkowski's approach the data model consists of (1) "real" regions in which the starting position is different from the end position (text elements) and (2) "fake" regions where those two positions are the same (words). In the former case the predefined data model consists only of "fake" regions. This decision made the introduction of ordering operator a necessity, as this was the only way to define the "real" regions in the algebra. As a consequence, this made the creation of arbitrary extents possible in the algebra.

2.2.4 Proximal nodes

A proximal node region algebra was introduced by Navarro and Baeza-Yates [3]. Algebra is defined on a set of nodes that represents either symbols (i.e. words or characters) or structured elements organized in a set of independent hierarchies. The data model is formed of a *view* (tree structure of a document), specified using a set of *constructors* (node types in the three structure) and associated *segments* for each constructor (a pair of numbers representing contiguous portion of underlying text). Additionally, the data model has a specific *text view* composed of text constructors that have a flat structure.

The algebra supports operators for expressing inclusion (including direct inclusion, and positional inclusion), positional operators (i.e before, after), and set manipulation operators. Although the model is very expressive, it can not support overlap in result sets, as well as the combination of nodes from different views. In the paper [3] authors also explained how the operators can be efficiently implemented.

2.2.5 Consens/Milo region algebra

Consens and Milo [9] studied characteristics of region algebra approaches in the scope of capturing the important structural properties of structured text documents (i.e. nesting and ordering) and investigate the difficulty level in query optimization process. Consens and Milo used modified PAT algebra [34] as a basic model for their approach. They simplified the PAT approach by keeping only the core functionality of the algebra to be able to compare it with the approaches previously described in this section. Such an algebra was used as a basic model for their studies.

Consens and Milo created the relationship between the region algebra and the first order monadic theory of finite binary trees which they used for proving some of the properties in the algebra. Authors especially outlined two properties:

- region algebra approaches are incapable of expressing direct inclusion for nested regions, and
- region algebra approaches are incapable of expressing both included expression, i.e., the containment relation between a region, and two regions which have to be in a specified order.

As these two properties are useful for many applications, authors proposed the extension of the algebra as a programming language embedding, where these two properties can be supported.

2.2.6 Nested region algebra

The application of region algebra for search in nested text regions was presented by Jaakkola and Kilpelainen [23, 24]. The data model is defined on a set of regions where each region represents a contiguous non-empty interval of positions. The data set is dynamically created as authors assume that there exist an indexing function $I(p)$ that returns a region set consisting of all the regions that bound the text pattern p occurring in a queried text. They assume that the numbering is formed on a character basis, instead of a word basis.

Operators are defined based on conditions of relative containment and ordering (start and end position orders) of regions. In the operator set all containment operators from the model presented in [8] are preserved, except that the Boolean AND and OR operators are now replaced with the set operators which have similar functionality. Furthermore, additional operators are introduced for expressing set difference (Boolean NOT operator) and for coping with nested properties of data model: (1) binary operator *quote* for producing disjoint "followed by" regions, (2) unary operator *hull* for producing minimal set of regions that covers the regions in an original region set, and (3) binary operator *extracting* for removing nested regions.

Jaakkola and Kilpelainen showed that the nested region algebra queries can be evaluated in practice in linear time. They have proven that their nested region algebra is capable of expressing "both included" relation (although the expression is somewhat complex [24]). The "direct inclusion" relation, however, cannot be expressed in their algebra.

2.2.7 Text constraints

Another region algebra approach for arbitrary text regions is described in [30, 31], where Miller et al. based their algebra on the language for specifying text structure, called text constraints (TC). Similarly to a region definition in [24], authors defined the data model on a character basis. The indexing is again performed dynamically using three different primitive expressions: (1) literals for matching all occurrences of a string in a data model, (2) regular expressions for matching regions that satisfy regular expression specification, and (3) identifiers that refer to predefined named regions (e.g., *tag* in HTML).

The operator set consists of the same operators as defined in [24], except the hull and extracting operators. The operator set also introduces new operators, such as concatenation operator which is used for defining adjacent regions. Furthermore, Miller [30] showed that many more operators can be defined using the *iterator* operator in combination with the basic region operators. The operator specification is based on a subset of Allen's 13 interval relations [1], namely before, after, in, contains, overlaps-start, overlaps-end[3]. The algebra allows the usage of unary relational operators, which are actually binary operators with the implicit operand that represents the union of all regions in the data set.

It is interesting to note that in the implementation Miller et al. [30, 31] used the *region interval* formed from dense region sets. A region interval defines a set of regions whose start and end positions are given by intervals rather than with positions. Authors transformed the region interval space into two dimensional plane and used a variant of R-tree to evaluate the region algebra expressions [31]. Furthermore, authors showed that with the usage of tandem tree traversal the complexity of the relational expressions will be linear in the worst case. In [30] an extensive research was conducted, both empirical and mathematical, for defining the complexity of region algebra expression evaluation.

2.2.8 Ranked region algebra

An attempt for adapting region algebra to ranked retrieval was presented in [26, 27]. The algebra is based on the region algebra presented in [8] (i.e., nesting is prohibited for regions). However, the approach cannot be considered as a fully algebraic approach since the ranking is not defined in the scope of algebraic operators but rather as a side effect of the application of algebraic operators (similarly to approach presented in [6]).

In this approach, query is decomposed into a series of subqueries, each representing a subtree of a query tree. Therefore, each subquery represents the region algebra operation evaluated on two region sets that are obtained after the execution of child subqueries in a query tree. The algebra operators are defined as in [8].

To enable ranked retrieval Masuda et al. [26, 27] extended the definition of operators in a way that each operator produces scores for the result regions as a side effect. For score computation they considered each extent to be a keyword, and treated it like keyword in traditional IR systems. Using traditional IR *tf.idf* approach authors applied term frequency and inverse document frequency computations for extents (i.e. keywords) obtained as a result of the application of subquery on the data model. The computed scores are used to define a *document vector* and a *query vector*. The document vector represents the vector of term frequencies computed for the results of each subquery, while the query vector represents inverse document frequency kind of measure for each subquery (for details see [27]). The final score is computed as a cosine measure between the document vector and the query vector.

In order to incorporate the structural information in the retrieval model authors proposed several mappings (i.e., coefficients) that can be used to define the query vector. Besides the simple *idf* measure, two other coefficients are proposed. The *structure coefficient* takes into account the rareness of the structures. The *interpolated coefficient* is the *idf* score of the query itself, combined with the weighted average *idf* scores of its subqueries.

[3]Miller has shown that the set of relations is complete and that the other seven possible relations can be defined as a union or intersection of the basic six relations.

2.3 Comparison of Different Region Algebra Data Models and Properties

Here we emphasize several differences in region algebra approaches. Although region algebra approaches share the same nucleus, as we already saw, region algebra can be formalized differently considering the original data model and the creation of data set on which algebra operators are defined. Table 1 presents comparison of different region algebra approaches considering the features of region algebras discussed below.

Table 1: Comparison of different region algebra approaches.

Approach	model base	predefined	dynamic	nesting	overlap	set ops.	rank ops.
PAT	both	no	yes	no	no	all	no
concordance list	string	yes	no	no	no	union	yes
GC-list	string	yes	yes	no	yes	no	no
proximal nodes	both	yes	yes	yes	no	all	no
Consens/Milo	string	yes	no	yes	yes	all	no
nested regions	character	no	yes	yes	yes	all	no
text constrains	character	no	yes	yes	yes	all	no
ranked algebra	string	yes	yes	no	yes	no	yes

First distinction is based on the content modeling aspect in the region algebra data model (*model base* column in Table 1). One way is to view document text as a character string and index each character in a string with its relative position with respect to the beginning of a document. Although this approach is beneficial for regular expression and substring search, it yields many problems with respect to large document collections ($> 100MB$), especially if we consider the way that each data unit and its *id* must be defined in a database data model. In that case each character, along with its index (*id*), should be stored in the database, resulting in an enormous storage overhead. The other solution would be to index the characters dynamically in a database using the existing database operators which would result in a very inefficient query execution.

Therefore, in many approaches the text is viewed as an indexed string (the name is taken from [34]), where string consists of indexed elements (i.e. tokens or words) and delimiters (e.g., white space, line feed, etc.). Although this approach introduces some problems in distinguishing between elements and delimiters (e.g., if "." is considered as a delimiter then the string "27.01" would be divided in two elements, "27" and "01" and search for the term "27.01" would fail), the indexed string manipulation and storage from a database point of view are much more seamless.

Furthermore, we can distinguish between two approaches in generating the data set for the region algebra (as we stated in the Section 2.1): (1) region algebras where the data set on which the algebraic operators can be performed is predefined (static region algebra approach) and (2) region algebras where regions are dynamically created during algebra expression evaluation (dynamic region algebra approach). In the former case the fetching operator ($fetch(token)$) should only select a specified token that is defined (indexed) in the data set. In the latter case an indexing operator exists ($I(pattern)$) that creates a region (specifies its starting and end positions) based on a pattern specification and its relative position in the document. In case of the character based approach where the pattern is actually a sequence of characters, the region is created dynamically, using the begin and end position of the characters of a pattern matched in a document. This distinction is depicted in the column *dynamic* in Table 1.

The region algebra approaches also differ in the way how tagging information (e.g., mark-up) is treated in the indexing process. In case of character based indexing, mark-up regions are dynamically created by recognizing the character sequences used to denote the begin and the end delimiters of a tagged data sequence, e.g., "<" and ">" for the start tag in XML. To avoid the repetition of complex expressions for defining opening and closing tags Jaakkola and Kilpelainen [24] used macros that can be considered as complex operators for finding regions that define tags. In the indexed string approach, meta data has to be somehow distinguished from the document

content. Therefore, Clarke et al. [8] used real numbers to index mark-up and Burkowski [6] used "naming" for concordance lists that is predefined (defined during the database load). This distinction is listed in the *predefined* column in Table 1.

Table 1 also points out the expressiveness of different region algebra approaches by specifying what kind of operators they support besides the basic containment operators. Column *nesting* specifies whether a particular region algebra approach supports operators that manipulate nested regions, while *overlap* column depicts whether region algebra approaches allow overlapping between regions. The last two columns, *set ops.* and *rank ops.* distinguish between approaches that support or do not support set operators, namely union, intersection, and difference, and approaches that support ranking of regions in the algebra respectively.

3 XML and Relational Databases

In this section we explain the formation of the initial XML data set and discuss some issues on the indexing of XML documents. The relational storage of such documents is also discussed, along with the relational algebra expressions for two NEXI query examples. The section is concluded with the discussion on relational approaches for XML retrieval and a rationale is given for the introduction of the new structure aware (SA) logical level in a DBMS.

3.1 Representing XML in Relational Databases

Most of the database approaches to XML choose to index XML documents before storing them into relational tables. The rationale for this is the structural organization of XML documents and the benefits that can be achieved when querying such an indexed relational representation of XML documents. Furthermore, the structural information can also be used on higher levels in a DBMS for optimization and rewriting. For an illustration we refer to [20] where the authors used the pre-post and stretched pre-post indexing scheme for the relational storage of XML documents. In our approach we used a variant of the stretched pre-post indexing[4] scheme that also indexes each word in XML text nodes. Note that the indexing also produces the initial data set for the data model that we define in Section 4.

```
<article lang=‘‘en’’ date=‘‘10/02/04’’>
   <title>Region algebra</title>
   <bdy>
      <sec>
         <p>Structured documents ...</p>
         <p>Text search ...</p>
      </sec>
      ...
   </bdy>
   ...
</article>
```

Figure 1: Example XML document.

The data set creation, i.e., the formation of the initial data set from XML documents (in a plain text format) can be explained through the usage of a two step indexing process[5]. The indexing process is explained using an example XML document given in Figure 1. In the first

[4]Note that the term indexing differs from the concept of indexing as defined in the traditional database systems. It denotes the method used for creation of the initial data set stored in a database.

[5]Although XML documents are actually graphs we simplify the XML structure and treat these entities as if they were organized as a hierarchical (tree-like) structure.

step each token in the XML document (denoted with $\mathcal{D}$) is indexed regarding its relative position with respect to its beginning and its type: $I_1 : \mathcal{D} \rightarrow \mathcal{X}$. As a result we obtain a set of elements: $x \in \mathcal{X}$, uniquely identified by their position in the XML document. Each element has the form of $x = \{position, token, token\_type\}$ as shown in Table 2.

Table 2: Intermediate index structure ($\mathcal{X}$) obtained after initial indexing (I_1) of XML document depicted in Figure 1.

position	**token**	**token_type**
0	`<article>`	start tag
1	`lang`	attribute name
2	`''en''`	attribute value
3	`date`	attribute name
4	`''10/02/04''`	attribute value
5	`<title>`	start tag
6	`region`	term
7	`algebra`	term
8	`</title>`	end tag
9	`<bdy>`	start tag
10	`<sec>`	start tag
11	`<p>`	start tag
12	`structured`	term
13	`documents`	term
...	...	...
54	`</p>`	end tag
...	...	...
576	`</sec>`	end tag
...	...	...
9876	`</bdy>`	end tag
...	...	...
10034	`</article>`	end tag

The second step produces regions that can be considered as the initial data set. These regions are produced by pairing corresponding tokens that represent opening and closing tags, attribute names and values, etc., and by removing mark-up delimiters from the tokens: $I_2 : \mathcal{X} \rightarrow \mathcal{C}$. This results in a data set like the one presented in Table 3. Thus, the initial data set construction can be defined as a composition of two indexing procedures: $I = I_1 \circ I_2$. Although the indexing is a two step process it can be implemented as a single walk through an XML document using the SAX parser and stack structures (see e.g., [20]).

An indexed XML collection (document), however, is not stored in one relational table since this table would be huge and in most cases (on most platforms) hard to process. In many relational approaches to XML different fragmentations of this basic table are used. The fragmentation can be horizontal, based only on type of XML nodes (like in [20] and [25]), vertical based on a name and/or type of XML elements, e.g., [13], or based on paths to XML nodes in an XML tree structure (like in [32]). For illustrative purpose we use horizontal fragmentation of XML data as presented in [25]. Consequently, different relational tables are defined for the XML element nodes and attribute nodes and the word table is defined for the words in the XML text nodes. This is depicted in Table 4. In further discussion we will not consider the attribute table since it is not important for the issues that we are discussing in this paper.

3.2 From XML Queries to Relational Algebra

The two example queries given in Figure 2 will be used as our leading examples in the following sections. As query language we use the NEXI query language which has officially been adopted

Table 3: Data model for XML document presented in Figure 1 obtained after the composition of initial indexing (I_1) and final indexing (I_2).

start	end	name	type
0	10034	`article`	node
1	2	`lang`	attr_name
2	2	`en`	attr_value
3	4	`date`	attr_name
4	4	`10/02/04`	attr_value
5	8	`title`	node
6	7	-	text
6	6	`region`	term
7	7	`algebra`	term
9	9876	`bdy`	node
10	576	`sec`	node
11	54	`p`	node
12	53	-	text
12	12	`structured`	term
13	13	`documents`	term
...	...	...	...

Table 4: Relational data model for storing XML document presented in Figure 1.

Node table $\mathcal{N}$

start	end	name	type
0	10034	`article`	node
5	8	`title`	node
6	8	-	text
9	9876	`bdy`	node
10	576	`sec`	node
11	54	`p`	node
12	53	-	text
...	...	...	...

Word table $\mathcal{W}$

start	name
6	`region`
7	`algebra`
12	`structured`
13	`documents`
...	...

Attribute table $\mathcal{A}$

start	owner	name	type
1	0	`lang`	name
2	0	`en`	value
3	0	`date`	name
4	0	`10/02/04`	value
...	...	...	...

for INEX 2004[6]. Its detailed description can be found in [37]. For now we consider that the *about* condition inside queries is strict (corresponds to a Boolean search, i.e., *about* behaves the same as XPath *contains* expression). Later on in this paper we elaborate more on the use of the *about* clause for ranking.

For the chosen storage model, composed of $\mathcal{N}$ and $\mathcal{W}$, we can directly transform any NEXI expression into relational algebra expression. For NEXI example query 1 depicted in Figure 2, possible relational algebra expressions could be specified as given in Figure 3. We disregard the type attribute in the relational expressions for brevity.

Note that there is a frequent usage of a group of expressions consisting of join and projection

[6]http://inex.is.informatik.uni-duisburg.de:2004/.

Figure 2: NEXI queries.

Example NEXI query 1:

```
//article//bdy[about(.//sec, structured) and about(.//sec, documents)]
```

Example NEXI query 2:

```
//article//bdy[about(., region) and about(., algebra)][about(.//sec, XML)]
              //p[about(., information) and about(., retrieval)]
```

operations that simulate the XPath descendant/ancestor step. This group of expressions actually represents the bottleneck for XPath query processing, since its naive execution in relational DBMS is extremely slow. A number of techniques have been proposed to speed up the execution of XPath descendant and ancestor steps, such as multi-predicate merge join [39], staircase join [20], containment join [25], etc. Using such abstract join operators, denoted with $\bowtie_{\sqsupset}$ (for expression types R_7, R_8 and R_{10} in Figure 3) and $\bowtie_{\sqsubset}$ (for expression type R_6 in Figure 3), the query plan for NEXI query example 2 can be expressed as shown in Figure 4[7].

Figure 3: Relational query plan for example query 1 given in Figure 2.

$$
\begin{aligned}
R_1 &= \sigma_{n=\text{“article”}}(\mathcal{N}) \\
R_2 &= \sigma_{n=\text{“bdy”}}(\mathcal{N}) \\
R_3 &= \sigma_{n=\text{“sec”}}(\mathcal{N}) \\
R_4 &= \sigma_{n=\text{“structured”}}(\mathcal{W}) \\
R_5 &= \sigma_{n=\text{“documents”}}(\mathcal{W}) \\
R_6 &= \pi_{start_2,end_2,name_2}(R_2 \bowtie_{start_2>start_1,end_2<end_1} R_1) \\
R_7 &= \pi_{start_3,end_3,name_3}(R_3 \bowtie_{start_3<start_4,end_3>end_4} R_4) \\
R_8 &= \pi_{start_3,end_3,name_3}(R_3 \bowtie_{start_3<start_5,end_3>end_5} R_5) \\
R_9 &= R_7 \cap R_8 \\
R_{10} &= \pi_{start_6,end_6,name_6}(R_6 \bowtie_{start_6<start_9,end_6>end_9} R_9)
\end{aligned}
$$

Figure 4: Relational query plan for example query 2 given in Figure 2.

$$
\begin{aligned}
R_1 &= \sigma_{name=\text{“article”}}(\mathcal{N}) \\
R_2 &= \sigma_{name=\text{“bdy”}}(\mathcal{N}) \\
R_3 &= \sigma_{name=\text{“sec”}}(\mathcal{N}) \\
R_4 &= \sigma_{name=\text{“p”}}(\mathcal{N}) \\
R_5 &= \sigma_{name=\text{“region”}}(\mathcal{W}) \\
R_6 &= \sigma_{name=\text{“algebra”}}(\mathcal{W}) \\
R_7 &= \sigma_{name=\text{“XML”}}(\mathcal{W}) \\
R_8 &= \sigma_{name=\text{“information”}}(\mathcal{W}) \\
R_9 &= \sigma_{name=\text{“retrieval”}}(\mathcal{W}) \\
R_{10} &= R_2 \bowtie_{\sqsubset} R_1 \\
R_{11} &= ((R_{10} \bowtie_{\sqsupset} R_5) \cap (R_{10} \bowtie_{\sqsupset} R_6)) \bowtie_{\sqsupset} (R_3 \bowtie_{\sqsupset} R_7) \\
R_{12} &= R_4 \bowtie_{\sqsubset} R_{11} \\
R_{13} &= (R_{12} \bowtie_{\sqsupset} R_8) \cap (R_{12} \bowtie_{\sqsupset} R_9)
\end{aligned}
$$

[7]This abstract join operator can be considered as a step towards (higher level) structure aware algebra operator, see Section 4.

3.3 Do We Need an Algebra in Between?

There might be a number of reasons to define an intermediate algebra that differs from relational algebra. First we give two most important points that reflects the drawbacks of using relational algebra for XML IR.

1. As we saw in the previous sections, to be able to express XPath+IR (NEXI) queries in relational databases we need new operators for the efficient execution of XPath+IR subexpressions, such as descendant and ancestor steps, vague containment conditions, etc. The exact technique how we implement the subexpression is defined at the physical level, and it does not have to be unique, i.e., we can have multiple variants of relational expressions for the same XPath+IR subexpressions. Therefore, the execution times for distinct implementations can differ regarding the relational storage of the XML data, parameters of the relational tables, and index structures used for accelerating the execution of relational expressions.

2. Another important issue concerning immediate translation of XPath+IR expressions into relational algebra is that the relational algebra expressions are highly dependent on the relational schema chosen for the storage of XML data. If we change the relational storage, the relational algebra expressions for each query have to be rewritten according to the chosen relational schema. This is especially the case for large collection of XML data, since usually huge relational tables, that have more than a million entries, are typically broken into a number of smaller ones using one of the fragmentation methods mentioned in Section 3.1.

A new logical level that is able to reason about containment relation among XML entities, i.e., element nodes, text nodes, words, etc., and provide the framework for XML elements relevance score computation, would be more appropriate for XML IR for number of reasons:

1. Having a structure aware level with the algebra defined in it would provide the right level of abstraction considering different XPath+IR queries formed at the conceptual level and the storage structure and access algorithms chosen at the physical level. In such a way we provide the needed *data independence* at the SA logical level.

2. Furthermore, the reasoning that can be done on the SA level can be useful for query rewriting and *optimization*. Using knowledge about the size of the operands and the cost of the execution of different operators at the physical level we are able to generate different logical query plans, achieving faster execution times and lower memory usage when executing.

3. A final but important reason for defining an algebra at the logical level is to allow the expression of IR-like queries (*about* in NEXI), i.e., score computation and ranking of XML elements. Therefore, the algebra should provide a specific level of *IR understanding* that is based on the retrieval model used for score computation. Thus, we should be able to implement different retrieval models at the SA level, and compare their performance with respect to their effectiveness and efficiency, without altering the logical algebra.

These features cannot be clearly specified in the relational algebra, as relational algebra is not capable of capturing ranked retrieval, and therefore blurs the opportunity for query rewriting and optimization with score computations in ranked retrieval. On the contrary, algebra at the SA level should provide clear rules for query rewriting and optimizations, where not only plain but scoring operators should adhere certain operator properties that can be used for query optimization. The exact way of how we use region algebra operator properties on the SA level and how we extend the region algebra to support ranked retrieval is explained in the following two sections.

4 Our Region Algebra

For defining the intermediate logical level we have chosen the region algebra approach, because it is already well established in the area of structured document retrieval (see Section 2), and because

Table 5: Basic score region algebra operators.

Operator	**Operator definition**
$\sigma_{n=name,t=type}(R)$	$\{r \mid r \in R \wedge n = name \wedge t = type\}$
$R_1 \sqsupset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \exists r_2 \in R_2 \wedge s_1 < s_2 \wedge e_1 > e_2\}$
$R_1 \not\sqsupset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \nexists r_2 \in R_2 \wedge s_1 < s_2 \wedge e_1 > e_2\}$
$R_1 \sqsubset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \exists r_2 \in R_2 \wedge s_1 > s_2 \wedge e_1 < e_2\}$
$R_1 \not\sqsubset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \nexists r_2 \in R_2 \wedge s_1 > s_2 \wedge e_1 < e_2\}$
$R_1 \sqcap R_2$	$\{r \mid r \in R_1 \wedge r \in R_2\}$
$R_1 \sqcup R_2$	$\{r \mid r \in R_1 \vee r \in R_2\}$

of the numerous useful properties of region algebra operators that we discuss in the remainder of the paper.

Since we focus on XML documents, we decided to use the rich information set of the XML data model [10] as a base for the definition of our region algebra data model. Furthermore, given the expressive data model we are able to distinguish between different information items in XML (see [10]) and to further extend the algebra with new operators and concepts for ranked retrieval. As we enriched the region definition with a score concept (introduced in Section 5), we named the algebra *score region algebra*, or *SRA* in short.

4.1 Plain region algebra

As already stated, with the specification of the region algebra data model we provide a uniform platform for defining region algebra operators. Here, we discuss the basic XML region algebra data model which can be defined using four region attributes, based on the indexed data set described in the previous section (for more details see [25]).

Our XML data model consists of element, text, comment, and processing instruction nodes, as well as attribute information and text node content. Thus we disregard some of the entities in the complete XML data model as defined in [10] since they are not relevant to the topic of this paper. We can exert that our algebra can be extended to support other XML entities such as namespaces, IDREFs, etc., without affecting the core data model. Thus, in our XML data model we simplify the XML structure and treat these entities as they are organized in a hierarchical (tree-like) structure.

Definition 1 *The plain region algebra data model is defined on the domain $\mathcal{R}$ which represents a set of region tuples. A region tuple r ($r \in \mathcal{R}$), $r = (s, e, n, t)$, is defined by these four attributes: region start attribute - s, region end attribute - e, region name attribute - n, and region type attribute - t. The region start and end attributes must satisfy ordering constraints ($e \geq s$).*

The semantics of region start and region end attributes are the same as in other region algebra approaches: they denote the bounds of a region. The region name attributes are used to denote node names, content words, attribute names, attribute values, etc. To be able to distinguish different node types in XML, the type information is needed.

Next, we define the basic region algebra operators. The definition of region algebra operators is based on the operators specified in the previous region algebra approaches, extended to support a specific XML structure. Table 5 defines the following seven basic region algebra operators: selection (σ), containing ($\sqsupset$), not containing ($\not\sqsupset$), contained by ($\sqsubset$), not contained by ($\not\sqsubset$), region set intersection ($\sqcap$), and region set union ($\sqcup$). We use R_i ($i = 1, 2, ...$) to denote the region sets, their corresponding non-capitals to denote regions in these region sets (r_i), and corresponding indexed non-capitals to denote region attributes (s_i, e_i, n_i, t_i).

As can be noticed in Table 5, we introduce a selection operator (σ) instead of an interval operator used in [6, 8, 23, 26] (usually denoted with $I(token)$). The interval operator is actually not a real region algebra operator but rather specifies the indexing function applied to a specified

Figure 5: Region algebra query plan for example query 1 given in Figure 2.

$$
\begin{aligned}
\textsc{article} &= \sigma_{n=\text{“article”},t=node}(C) \\
\textsc{bdy} &= \sigma_{n=\text{“bdy”},t=node}(C) \\
\textsc{sec} &= \sigma_{n=\text{“sec”},t=node}(C) \\
\textsc{structured} &= \sigma_{n=\text{“structured”},t=term}(C) \\
\textsc{documents} &= \sigma_{n=\text{“documents”},t=term}(C) \\
R_1 &= (\textsc{sec} \sqsupset \textsc{structured}) \sqcap (\textsc{sec} \sqsupset \textsc{documents}) \\
R_2 &= (\textsc{bdy} \sqsubset \textsc{article}) \sqsupset R_1
\end{aligned}
$$

Figure 6: Region algebra query plan for example query 2 given in Figure 2.

$$
\begin{aligned}
\textsc{article} &= \sigma_{n=\text{“article”},t=node}(C) \\
\textsc{bdy} &= \sigma_{n=\text{“bdy”},t=node}(C) \\
\textsc{sec} &= \sigma_{n=\text{“sec”},t=node}(C) \\
\textsc{p} &= \sigma_{n=\text{“p”},t=node}(C) \\
\textsc{region} &= \sigma_{n=\text{“region”},t=term}(C) \\
\textsc{algebra} &= \sigma_{n=\text{“algebra”},t=term}(C) \\
\textsc{xml} &= \sigma_{n=\text{“XML”},t=term}(C) \\
\textsc{information} &= \sigma_{n=\text{“information”},t=term}(C) \\
\textsc{retrieval} &= \sigma_{n=\text{“retrieval”},t=term}(C)
\end{aligned}
$$

$$
\begin{aligned}
R_1 &= \textsc{bdy} \sqsubset \textsc{article} \\
R_2 &= ((R_1 \sqsupset \textsc{region}) \sqcap (R_1 \sqsupset \textsc{algebra})) \sqsupset (\textsc{sec} \sqsupset \textsc{xml}) \\
R_3 &= (\textsc{p} \sqsubset R_2) \\
R_4 &= (R_3 \sqsupset \textsc{information}) \sqcap (R_3 \sqsupset \textsc{retrieval})
\end{aligned}
$$

token that returns the region set of the occurrences of *token* in a document. The selection operator is a unary region algebra operator that operates on a region set and produces a region set as a result. It is defined to enable the selection of regions formed during the initial data set creation (explained in Section 3.1), based on name and type region attributes.

Following the query examples given in Figure 2, we give the same query execution plans defined using the region algebra operators instead of the relational ones given in Figures 3 and 4. The region algebra query plans for query examples 1 and 2 are given in Figure 5 and Figure 6. We use C to denote the initial data set of regions . We can note a great resemblance between the previous relational query plans and region algebra query plans. This exerts the simplicity of transforming region algebra expressions into relational expressions. However, any change in the relational storage will result in the change of the query plan only at the physical level, while the query plan on the SA level remains the same.

Note that in order to model XML properly, we could enrich the definition of a region with the additional information of XML references, parent, or level information, etc. For some details on extensions on region algebra approaches we refer to papers [25], [28], and [29].

4.2 Region Algebra Operator Properties

In this section we discuss the properties of the algebraic operators and their use for query rewriting and optimization. Some of the properties are illustrated using the examples given in Figure 5 and Figure 6. Many properties are mentioned in the papers about region algebra by Clarke et al. [8] and Jaakkola and Kilpelainen [23], but none of these papers discuss their usage. Our study on region algebra shows that there are only a few operators that have the basic operator properties such as: identity, inverse, commutativity, associativity, and distributivity. However, there is a number of region algebra specific properties that can be considered as special cases of distributivity and associativity.

Besides the unary selection operator (σ), we can distinguish two classes of binary region algebra operators. The first class consists of containment operators: $\sqsupset$, $\not\sqsupset$, $\sqsubset$, $\not\sqsubset$, while the second class consists of the standard set operators: $\sqcap$ and $\sqcup$. Below we discuss some of the containment and set operator properties in more detail.

4.2.1 Operators $\sqsupset$ and $\not\sqsupset$

Operators $\sqsupset$ and $\not\sqsupset$ select a subset of regions in the first operand with respect to the regions in the second operand.

Identity: For the operator $\sqsupset$ right identity can not be specified since regions that represent leaf nodes in XML tree do not contain other regions and the result of the application of operator $\sqsupset$ will always be an empty set. However, the operator $\not\sqsupset$, which is an inverse of the operator $\sqsupset$, has two right identities, that is a collection root node - *Root* and an empty set - $\emptyset$:

$$R \not\sqsupset Root = R, \tag{1}$$

$$R \not\sqsupset \emptyset = R. \tag{2}$$

Inverse: Based on the above mentioned property of $\sqsupset$ and $\not\sqsupset$ operators we can conclude that the first operator has no inverse element, while the second one has a left inverse which is actually the collection root node in case the left operator is not the root node itself:

$$Root \not\sqsupset R = \emptyset,\ R \neq Root \tag{3}$$

Commutativity: Based on the definition of $\sqsupset$ and $\not\sqsupset$ operators these operators do not follow the commutativity law (see Example 1 in Appendix A).

Associativity: Similarly, based on the left operand selective nature of $\sqsupset$ and $\not\sqsupset$ they are not associative, which can be seen in Example 2 given in Appendix A.

Distributivity: It can be proven (see Examples 3 - 6 in Appendix A) that for most of the combinations the distributive law does not hold. It holds only in one case that can easily be proven based on the definition of operators $\sqsupset$ and $\sqcup$:

$$R_1 \sqsupset (R_2 \sqcup R_3) = (R_1 \sqsupset R_2) \sqcup (R_1 \sqsupset R_3) \tag{4}$$

4.2.2 Operators $\sqsubset$ and $\not\sqsubset$

Similarly as for the previous two operators, operators $\sqsubset$ and $\not\sqsubset$ are defined as containment operators that select a subset of regions in a first operand.

Identity: For the operator $\sqsubset$ right identity can not be specified in the general case since the collection root region is not contained in any other region and the result of the application of operator $\sqsubset$ will always be an empty set. However, if we exclude the collection root region from the data node set we can define the identity which is the same root node. The operator $\not\sqsubset$, which is an inverse of operator $\sqsubset$, has one right identity:

$$R \sqsubset Root = R,\ R \neq Root, \tag{5}$$

$$R \not\sqsubset \emptyset = R. \tag{6}$$

Inverse: Operator $\sqsubset$ do not have the inverse element since there is no element that can contain the collection root node. Operator $\not\sqsubset$ do have the left inverse element:

$$\emptyset \not\sqsubset R = \emptyset. \tag{7}$$

Commutativity: Based on the definition of $\sqsubset$ and $\not\sqsubset$ operators they do not follow the commutativity law (see Example 7 in Appendix A).

Associativity: Operators $\sqsubset$ and $\not\sqsubset$ are not associative as can be seen in Example 8 in Appendix A.

Distributivity: This property holds only in one case, similar as for the containing operators above (see Examples 9 - 12 in Appendix A):

$$R_1 \sqsubset (R_2 \sqcup R_3) = (R_1 \sqsubset R_2) \sqcup (R_1 \sqsubset R_3). \tag{8}$$

4.2.3 Operator $\sqcap$

Operator $\sqcap$, based on its definition, can be considered as a binary operator equal to a set intersection operator. Therefore, the properties of the $\sqcap$ operator are similar to the properties of a set intersection operator.

Identity: In case of the $\sqcap$ operator the identity element is the whole node set collection in a database - C. Furthermore, it is a two-sided identity, since it can be used as a left or right identity:

$$R \sqcap C = R, \tag{9}$$

$$C \sqcap R = R. \tag{10}$$

Inverse: Since there are no two SRA regions which can form the complete collection data set using the operator $\sqcap$ (except two C regions), it does not have the inverse.

Commutativity: Based on the definition of the $\sqcap$ operator, we can prove that the operator is commutative:

$$R_1 \sqcap R_2 = R_2 \sqcap R_1. \tag{11}$$

Associativity: Similarly to a set intersection operator, the $\sqcap$ operator is associative:

$$(R_1 \sqcap R_2) \sqcap R_3 = R_1 \sqcap (R_2 \sqcap R_3). \tag{12}$$

Distributivity As operator $\sqcap$, applied on two region sets, as a result produces the set of regions that are in both operand sets, the distributivity with operators $\sqsupset$ and $\sqsubset$ does not hold (see Proof in Appendix A). Since the definition of $\sqcap$ and $\sqcup$ is the same as the definition of the set intersection and set union operators the distributivity of $\sqcap$ over $\sqcup$ holds:

$$R_1 \sqcap (R_2 \sqcup R_3) = (R_1 \sqcap R_2) \sqcup (R_1 \sqcap R_3). \tag{13}$$

4.2.4 Operator $\sqcup$

Based on the definition of operator $\sqcup$, it can be considered as a binary operator similar to the set union operator. Therefore, the properties of the $\sqcup$ operator are similar to the properties of the set union operator.

Identity: In case of the $\sqcup$ operator the identity element is an empty set element - $\emptyset$. The identity is two-sided, since it can be used as a left or right identity:

$$R \sqcup \emptyset = R, \tag{14}$$

$$\emptyset \sqcup R = R. \tag{15}$$

Inverse: Since there are no two SRA regions which can form the empty set using the operator $\sqcup$ (except two $\emptyset$ regions), it does not have the inverse element.

Commutativity: Based on the definition of $\sqcup$ operator, the operation is commutative:

$$R_1 \sqcup R_2 = R_2 \sqcup R_1. \tag{16}$$

Associativity: Similarly as set union operator, operator $\sqcup$ is associative:

$$(R_1 \sqcup R_2) \sqcup R_3 = R_1 \sqcup (R_2 \sqcup R_3). \tag{17}$$

Distributivity Since operator $\sqcup$ as a result produces the set of regions that are either in one or in both operand sets, the distributivity with operators $\sqsupset$ and $\sqsubset$ does not hold (see Proof in Appendix A). Since the definition of $\sqcup$ and $\sqcap$ is the same as the definition of the set union and set intersection operators the distributivity of $\sqcup$ over $\sqcap$ holds:

$$R_1 \sqcup (R_2 \sqcap R_3) = (R_1 \sqcup R_2) \sqcap (R_1 \sqcup R_3). \tag{18}$$

4.2.5 Special cases of associativity and distributivity

There are several interesting properties of the region algebra operators which can be useful for query rewriting and optimization at the SA level of a database. Here we mention the special case of containment operator associativity (property 19), containment operator normalization (property 20), and a special case of set-containment operator distributivity (property 21). The first two properties are mentioned in papers [8] and [23]. We know of no publication on region algebras that mentions the third property. Furthermore, we explain these properties on the two query examples given in Figure 2, whose query plans are given in Figure 5 and Figure 6.

For operators $op1 \in \{\sqsupset, \not\sqsupset, \sqsubset, \not\sqsubset\}$ and $op2 \in \{\sqsupset, \not\sqsupset, \sqsubset, \not\sqsubset\}$ property 19 and property 20 hold:

$$(R_1\ op1\ R_2)\ op2\ R_3 = (R_1\ op2\ R_3)\ op1\ R_2, \tag{19}$$

$$(R_1\ op1\ R_2)\ op2\ R_3 = (R_1\ op1\ R_2) \sqcap (R_1\ op2\ R_3). \tag{20}$$

For operators $op1 \in \{\sqcap, \sqcup\}$ and $op2 \in \{\sqsupset, \not\sqsupset, \sqsubset, \not\sqsubset\}$ property (21) is true:

$$(R_1\ op1\ R_2)\ op2\ R_3 = (R_1\ op2\ R_3)\ op1\ (R_2\ op2\ R_3). \tag{21}$$

These properties can be easily proven based on the definition of the operators.

To illustrate property 19 and property 20 we use the region algebra expression specified for the example query 1, given in Figure 5:

$$(\textsc{bdy} \sqsubset \textsc{article}) \sqsupset ((\textsc{sec} \sqsupset \textsc{structured}) \sqcap (\textsc{sec} \sqsupset \textsc{documents}))$$

The expression may be read as follows: "Retrieve bdy-elements *contained-by* article-elements *containing* the *intersection* of sec-elements *containing* the term 'structured' and sec-elements *containing* the term 'documents'." Using the property 19 we can rewrite this expression into:

$$(\textsc{bdy} \sqsupset ((\textsc{sec} \sqsupset \textsc{structured}) \sqcap (\textsc{sec} \sqsupset \textsc{documents}))) \sqsubset \textsc{article}$$

Furthermore, using the property 20 this expression can be rewritten into:

$$(\textsc{bdy} \sqsupset ((\textsc{sec} \sqsupset \textsc{structured}) \sqsupset \textsc{documents})) \sqsubset \textsc{article}$$

and using again the property 19 to:

$$(\textsc{bdy} \sqsupset ((\textsc{sec} \sqsupset \textsc{documents}) \sqsupset \textsc{structured})) \sqsubset \textsc{article}$$

Using properties 19 and 20 we are able to choose the most appropriate query plan assuming that we have the information on which subexpressions are more selective. This reasoning can be applied for choosing which subexpressions will be more selective for SEC $\sqsupset$ DOCUMENTS or SEC $\sqsupset$ STRUCTURED, or similarly for BDY $\sqsupset$ ((SEC $\sqsupset$ DOCUMENTS) $\sqsupset$ STRUCTURED) and BDY $\sqsubset$ ARTICLE expressions. For example, since usually all regions from the BDY region set are contained in the ARTICLE region set, the BDY $\sqsubset$ ARTICLE expression should be pushed up in the query plan as it is not a selective expression. Also the formulations of the query with the $\sqcap$ operator can be useful for parallel execution of two containment subqueries.

Property 21 is explained on a part of example query 2, denoted with R_2 in Figure 6. Using the expression:

$$((R_1 \sqsupset \text{REGION}) \sqcap (R_1 \sqsupset \text{ALGEBRA})) \sqsupset (\text{SEC} \sqsupset \text{XML})$$

where $R_1 = \text{BDY} \sqsubset \text{ARTICLE}$, and property 21 for operators $\sqcap$ and $\sqsupset$, this expression can be rewritten into:

$$((R_1 \sqsupset \text{REGION}) \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqcap ((R_1 \sqsupset \text{ALGEBRA}) \sqsupset (\text{SEC} \sqsupset \text{XML}))$$

We would obtain a similar region algebra expression for the `or` expression of example NEXI query 1 in Figure 2 instead of the `and` expression, where $\sqcap$ operator will be replaced with the $\sqcup$ operator. This provides an opportunity for, e.g., parallelization.

Furthermore, using property 19 next expression could be obtained from the previous one:

$$((R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset \text{REGION}) \sqcap ((R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset \text{ALGEBRA})$$

and after the application of property 20 the final expression would be:

$$((R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset \text{REGION}) \sqsupset \text{ALGEBRA}.$$

Therefore, instead of six operands and five operators we have a reduction to five operands and four operators, where the selection of the R_1 regions, that contain sections that contain term "XML" is pushed down to the first subexpression (assuming it is highly selective).

A similar expression can be obtained for the `or` combination in the *about*, where the distributivity property (13) could be applied as the last step:

$$(R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset (\text{REGION} \sqcup \text{ALGEBRA}).$$

5 Understanding IR

In this section some issues about the impact of introducing relevance ranking (i.e., score computation) in region algebra are discussed. We start with the score region algebra data model and continue with the definition of score region algebra operators. This section concludes with the SRA operator properties and with the emphasis of the role of score region algebra in a database retrieval model.

Table 6: Region algebra operators for score manipulation.

Operator	Operator definition
$R_1 \sqsupset_p R_2$	$\{r \mid r_1 \in R_1 \wedge (s,e,n,t) := (s_1,e_1,n_1,t_1) \wedge p := f_{\sqsupset}(r_1,R_2)\}$
$R_1 \not\sqsupset_p R_2$	$\{r \mid r_1 \in R_1 \wedge (s,e,n,t) := (s_1,e_1,n_1,t_1) \wedge p := f_{\not\sqsupset}(r_1,R_2)\}$
$R_1 \sqsubset_p R_2$	$\{r \mid r_1 \in R_1 \wedge (s,e,n,t) := (s_1,e_1,n_1,t_1) \wedge p := f_{\sqsubset}(r_1,R_2)\}$
$R_1 \not\sqsubset_p R_2$	$\{r \mid r_1 \in R_1 \wedge (s,e,n,t) := (s_1,e_1,n_1,t_1) \wedge p := f_{\not\sqsubset}(r_1,R_2)\}$
$R_1 \sqcap_p R_2$	$\{r \mid r_1 \in R_1 \wedge r_2 \in R_2 \wedge (s_1,e_1,n_1,t_1) = (s_2,e_2,n_2,t_1) \wedge (s,e,n,t) := (s_1,e_1,n_1,t_1) \wedge p := p_1 \otimes p_2\}$
$R_1 \sqcup_p R_2$	$\{r \mid r_1 \in R_1 \wedge r_2 \in R_2 \wedge ((s,e,n,t) := (s_1,e_1,n_1,t_1) \vee (s,e,n,t) := (s_2,e_2,n_2,t_2)) \wedge p := p_1 \oplus p_2\}$

5.1 SRA Data Model

Relevance ranking cannot be explicitly expressed in the standard relational algebra. For example, to store score information an additional attribute for each entry in the relational table must be introduced. This attribute stores the ranking score values for particular XML elements during the query execution. Furthermore, a number of operators have to be defined in the relational algebra which combined should express the score computation, i.e. instead of a join operator in Figure 4 we would use a combination of relational score operators. However, the introduction of score operators in region algebra is easier and more elegant than in relational algebra since the retrieval model specification is done on the right level of abstraction (SA level) and without considering the issues of how these operators are implemented on the lower (physical) level. Furthermore, as we will see in this section, the exact retrieval model does not have to be specified completely as the structure aware framework of our score region algebra can be kept abstract with respect to the exact retrieval model used.

To be able to model XML properly for IR-like search we enriched the definition of a region with the score information which represents the relevance of each region given the query specified by a user.

Definition 2 *The SRA data model is defined on the domain $\mathcal{R}$ which represents a set of region tuples. A region tuple r ($r \in \mathcal{R}$), $r = (s,e,n,t,p)$, is defined by these five attributes: region start attribute - s, region end attribute - e , region name attribute - n, region type attribute - t, and region score attribute - p. Region start and end attributes must satisfy ordering constraints ($e_i \geq s_i$).*

In the SRA the result of the application of operators cannot introduce new regions, that is, regions with different region bounds (i.e., s and e values) than originally specified in the database during the creation of the initial data set (database load). Furthermore, the operators cannot change the name and type (n and t) attributes of regions identified in the database. However, algebra operators are allowed to change the value of the region score attribute (p) for each region in the initial data set[8].

5.2 Scoring operators in Region Algebra

For explaining the SRA operators we again use the example query expressions given in Figure 2, except that we treat the *about* clause as a vague constraint instead of the strict interpretation as in previous sections. Thus, paths and terms in the about clause do not have to be strictly matched, and the vague match is defined by the retrieval model.

To enable score computation and region ranking based on computed scores we introduce new region algebra operators. For each binary region algebra operator defined in Table 5 the probabilistic counterpart is defined. To distinguish between plain region algebra operators and score

[8]These restrictions were introduced to simplify the model. However, to be able to model full XPath/XQuery languages some of these restrictions may be loosen.

region algebra operators we use the index $_p$ for score operators. The score operators are depicted in Table 6. Note that the operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$ produce all regions from the first operand (R_1) as a result, i.e., the region start, end, type and name attribute values are copied from the left operand region set to the result region set, while the score attribute (p) of the result set gets its value based on the containment relation among regions in the left and regions in the right operand as well as their score values. The definition of operators $\sqcap_p$ and $\sqcup_p$ are similar to the definition of basic set intersection and set union operators, i.e., the result region start, end, type, and name are obtained the same way as for plain region algebra operators, except that the result score value for regions is defined based on the score values of regions in the left and right operand region sets.

In the definition of score operators we introduce four complex scoring functions, $f_{\sqsupset}$, $f_{\not\sqsupset}$, $f_{\sqsubset}$, and $f_{\not\sqsubset}$, as well as two abstract operators, $\otimes$ and $\oplus$, which define the retrieval model. Using the abstract definition of operators we leave their exact implementation for the physical level (for more details on this issue see [25, 29]). However, for the $\oplus$ operator we assume that there exists a default score value (denoted with d), and in case when the region r_1 is not present in the region set R_2 the score is computed as $p = p1 \oplus d$ and in case when the region r_2 is not present in the region set R_1 the score is computed as $p = d \oplus p_2$.

The functions $f_{\sqsupset}$, $f_{\not\sqsupset}$, $f_{\sqsubset}$, and $f_{\not\sqsubset}$, applied to a region r_1 and a region set R_2, result in the numeric value that takes into account the score values of region $r_2 \in R_2$ and the probabilistic value that reflects the structural relation between the region r_1 and the region set R_2. For containing operator, taking into account the structural organization of XML documents which is a tree structure, usually many regions from the region set R_2 are contained in the region r_1 (e.g., section elements inside the article element). Although for the contained by operator there is a small chance that the region r_1 is contained by a set of regions present in R_2, it can happen that there are nested XML elements with the same name (e.g., section elements inside other section elements denoting subsections). Therefore, one region can be contained in multiple regions with the same name. Similar reasoning can be applied for not containing and not contained by operators.

Following the previous discussion, and isolating the score values from both operands we can define complex functions as follows:

$$f_{\sqsupset}(r, R) = p \cdot \sum_{\bar{r} \in R \sqsubset R'} (g_{\sqsupset}(\bar{r}, r) \cdot \bar{p}) \tag{22}$$

$$f_{\not\sqsupset}(r, R) = p \cdot \sum_{\bar{r} \in R \sqsubset R'} (g_{\not\sqsupset}(\bar{r}, r) \cdot \bar{p}) \tag{23}$$

$$f_{\sqsubset}(r, R) = p \cdot \sum_{\bar{r} \in R' \sqsupset R} (g_{\sqsubset}(\bar{r}, r) \cdot \bar{p}) \tag{24}$$

$$f_{\not\sqsubset}(r, R) = p \cdot \sum_{\bar{r} \in R' \sqsupset R} (g_{\not\sqsubset}(\bar{r}, r) \cdot \bar{p}) \tag{25}$$

We assume that R' is the region set containing a single region r, and $g_{\sqsupset}(\bar{r}, r)$, $g_{\not\sqsupset}(\bar{r}, r)$, $g_{\sqsubset}(\bar{r}, r)$, and $g_{\not\sqsubset}(\bar{r}, r)$ are abstract functions used to define the score propagation based on the structural relation between the region r and regions in the region set R. In the straightforward implementation functions $g_{\sqsupset}(\bar{r}, r)$ and $g_{\sqsubset}(\bar{r}, r)$ can be constant functions equal to e.g., 1. However, if we base the retrieval model on the term frequency, the first two functions can be defined as $g_{\sqsupset}(\bar{r}, r) = \frac{size(\bar{r})}{size(r)}$ and $g_{\not\sqsupset}(\bar{r}, r) = 1 - \frac{size(\bar{r})}{size(r)}$. Similarly, other two functions can be defined as, e.g., $g_{\sqsubset}(\bar{r}, r) = \frac{size(\bar{r})}{\sum_{\bar{r}} size(\bar{r})}$ and $g_{\not\sqsubset}(\bar{r}, r) = 1 - \frac{size(\bar{r})}{\sum_{\bar{r}} size(\bar{r})}$. Since the exact retrieval model is not the main issue in this paper and since we want to abstract as much as possible from the real implementation of retrieval model in the SRA, here we will not elaborate more on it. Later in this paper in Section 6 we give some hints how the retrieval model can be defined in the framework of SRA.

The abstract operator $\otimes$ specifies how scores are combined in an `and` expression, while the operator $\oplus$ defines the score combination in an `or` expression inside the NEXI predicate. In the remainder of the paper we take the simple approach where $\otimes$ is a product of two score values, while $\oplus$ is the sum of scores, as it shows good behavior for retrieval. Other implementations are

possible e.g., minimum and maximum. For more detail about possible implementations of these operators and on their comparison we refer to paper [25].

To illustrate the elegance of expressing score computation in region algebra we show how we can express NEXI query 1 in score region algebra:

$$(\text{BDY} \sqsupset_p ((\text{SEC} \sqsupset_p \text{STRUCTURED}) \sqcap_p (\text{SEC} \sqsupset_p \text{DOCUMENTS}))) \sqsubset \text{ARTICLE}$$

which very much resembles the original query plan for example query 1 given in Figure 5. The last operator in this query plan is SRA operator which is equal to the plain region algebra operator as it should only perform the selection on nodes from the left operand based on the regions in the right operand (ARTICLE), without any manipulation on score values of the left operand (i.e., just copying them). Therefore, the definition of this operand is the same as it is given in Table 5, except that in this case each region has additional fifth attribute which is a region score attribute. Thus, assuming that the data model is as defined for SRA, operators given in Table 5 are also valid in SRA, except that binary operators are used only for node selection based on containment relation and set intersection and union, and that unary select operator selects the specified regions based on name and type attribute, assuming the default value for score is already assigned to regions (e.g., 1).

5.3 Properties of Score Operators

Considering the properties of score operators we can exert that some of the properties follow ones defined for the region algebra without scores, some of them hold only if some conditions are satisfied (conditional properties which depend on the underlying retrieval model), and some of them are no longer valid.

5.3.1 Operators $\sqcap_p$ and $\sqcup_p$

The $\sqcap_p$ operator defines the Boolean-like AND combination of scores obtained for two regions with the same region bounds (i.e., s and e values). It preserves the identity and inverse element properties from the $\sqcap$ operator (properties 9 and 10), but only in case the default score value for all regions in the initial region set is the value which is the identity element for abstract operator $\otimes$, i.e., 1 for the multiplication:

$$R \sqcap_p C = R, \text{ i.e., } p \cdot 1 = p, \forall r \in R, \tag{26}$$

$$C \sqcap_p R = R, \text{ i.e., } 1 \cdot p = p, \forall r \in R. \tag{27}$$

Furthermore, the $\sqcap_p$ operator is commutative or associative (properties 11 and 12) if the operator $\otimes$ is commutative or associative, respectively, which is the case for multiplication:

$$R_1 \sqcap_p R_2 = R_2 \sqcap_p R_1, \tag{28}$$

i.e., $p_1 \cdot p_2 = p_2 \cdot p_1, \forall r_1, r_2 \in R_1 \sqcap R_2$, and

$$(R_1 \sqcap_p R_2) \sqcap_p R_3 = R_1 \sqcap_p (R_2 \sqcap_p R_3), \tag{29}$$

i.e., $(p_1 \cdot p_2) \cdot p_3 = p_1 \cdot (p_2 \cdot p_3), \forall r_1, r_2, r_3 \in (R_1 \sqcap R_2) \sqcap R_3$.

An extension of the set union operator is given by the $\sqcup_p$ operator. It defines the Boolean-like OR combination of scores for two regions. Similarly to the $\sqcap_p$ operator, the $\sqcup_p$ operator preserves the identity and inverse element properties from the $\sqcup$ operator (properties 14 and 15) but only in case the default value (d) taken for the $\sqcup_p$ operator is the value which is the identity element for the abstract operator $\oplus$, i.e., 0 for the summation in our case:

$$R \sqcup_p \emptyset = R \text{ i.e., } p + 0 = p, \forall r \in R, \tag{30}$$

$$\emptyset \sqcap_p R = R, \text{ i.e., } 0 + p = p, \forall r \in R. \tag{31}$$

As in the $\sqcap_p$ operator case, commutativity and associativity properties depend on the definition of the $\oplus$ operator. In other words, the $\sqcup_p$ operator is commutative or associative (properties 16 and 17) if the operator $\oplus$ is commutative or associative respectively, which is true for the summation.

$$R_1 \sqcup_p R_2 = R_2 \sqcup_p R_1, \tag{32}$$

i.e., $p_1 + p_2 = p_2 + p_1, \forall r_1, r_2 \in R_1 \sqcup R_2$, and

$$(R_1 \sqcup_p R_2) \sqcup_p R_3 = R_1 \sqcup_p (R_2 \sqcup_p R_3), \tag{33}$$

i.e., $(p_1 + p_2) + p_3 = p_1 + (p_2 + p_3), \forall r_1, r_2, r_3 \in (R_1 \sqcup R_2) \sqcup R_3$.

Furthermore, the $\sqcap_p$ operator distributes over the $\sqcup_p$ operator, since '$\cdot$' distributes over '+' (property 13). Vice versa is not the case (property 18).

$$R_1 \sqcap_p (R_2 \sqcup_p R_3) = (R_1 \sqcap_p R_2) \sqcup_p (R_1 \sqcap_p R_3) \tag{34}$$

Following the reasoning above and the fact that each region can equally likely be the right answer to a user query, we consider that the default value for region score in the initial data set C is 1, from now on, and that the default value for score d of a region not present in the region set for operator $\sqcup_p$ is 0. Thus, in our case properties 30 and 31 do not hold.

5.3.2 Operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$

The operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$ are modified versions of operators $\sqsupset$, $\not\sqsupset$, $\sqsubset$, and $\not\sqsubset$, that do not change the bounds of the regions and name and type attributes of the left operand but change the score attribute p according to the containment relation on the operands and their score values, defined by complex functions $f_{\sqsupset}$, $f_{\not\sqsupset}$, $f_{\sqsubset}$, and $f_{\not\sqsubset}$. Since the instantiation of these operators is not unique, i.e., they can implement distinct retrieval models, the exact operator properties can be determined only (after their instantiation) on the physical level.

Note that even if we define the additional operators as Boolean operators that return 0 or 1, indicating that the left operand contains or does not contain regions from the right operand, the properties of the operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$ will not strictly follow the properties of the operators $\sqsupset$, $\not\sqsupset$, $\sqsubset$, and $\not\sqsubset$. The same holds if the operators are defined based only on a number of regions in the second operand that are (not) contained in or (not) contained by the first operand.

Therefore, the operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$ do not have identity and inverse elements and are not commutative and associative. The reason for the lack of algebraic properties for these operators can be found in their purpose. These operators define the basic retrieval model (e.g., language model) of a DB IR system aimed at finding the score of an XML node (element node) region that contains a term region (region containing only one term). In most of the cases, these operators include foreground and background statistics (e.g., as defined in language model approach [21]). Furthermore, the definition of the operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$, along with the definition of other operators for score manipulation define the domain of region score attribute (e.g. what are the restrictions that the score manipulation operators must follow).

Based on the definition of the operators using the region frequency it can be proven that the operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$ do not distribute over the operator $\sqcup_p$ in general case (properties 4 and 8).

5.3.3 Additional SRA operator properties

There are some additional (conditional) properties of score operators which can be of interest for the optimization. If we assume that the functions $f_{\sqsupset}(r, R)$, $f_{\not\sqsupset}(r, R)$, $f_{\sqsubset}(r, R)$ and $f_{\not\sqsubset}(r, R)$ are defined as in Equations 22 - 25 and that abstract functions (g) are not dependent on the score value of region $\bar{r}$ (i.e., $g_{\sqsupset}(\bar{r}, r) = g_{\sqsupset}(\bar{s}, \bar{t}, \bar{n}, r)$, $g_{\not\sqsupset}(\bar{r}, r) = g_{\not\sqsupset}(\bar{s}, \bar{t}, \bar{n}, r)$, $g_{\sqsubset}(\bar{r}, r) = g_{\sqsubset}(\bar{s}, \bar{t}, \bar{n}, r)$, and $g_{\not\sqsubset}(\bar{r}, r) = g_{\not\sqsubset}(\bar{s}, \bar{t}, \bar{n}, r)$), property 19 holds for $op1_p \in \{\sqsupset_p, \not\sqsupset_p, \sqsubset_p, \not\sqsubset_p\}$ and $op2_p \in \{\sqsupset_p, \not\sqsupset_p, \sqsubset_p, \not\sqsubset_p\}$:

$$(R_1\ op1_p\ R_2)\ op2_p\ R_3 = (R_1\ op2_p\ R_3)\ op1_p\ R_2. \tag{35}$$

In other words the score for each region in the result region set, denoted with p, is computed as:

$$\begin{aligned} p &= (p_1 \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2)) \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3) \\ &= (p_1 \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3)) \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2), \end{aligned}$$

We use $g(\bar{r}, r)$ to denote one of the functions $g_{\sqsupset}(\bar{r}, r)$, $g_{\not\supset}(\bar{r}, r)$, $g_{\sqsubset}(\bar{r}, r)$, or $g_{\not\sqsubset}(\bar{r}, r)$, and $\bar{r} \in R \sqsupset R'$, $\bar{r} \in R \not\supset R'$, $\bar{r} \in R' \sqsubset R$, or $\bar{r} \in R' \not\sqsubset R$ based on the type of operators $op1_p$ and $op2_p$.

Furthermore, if the score value for all regions in the first operand R_1 is equal to 1 (default value for all regions), and we assume that the regions in each operand, R_2 and R_3, have the same score value, denoted with p_2 and p_3, property 20 holds:

$$(R_1\ op1_p\ R_2)\ op2_p\ R_3 = (R_1\ op1_p\ R_2) \sqcap_p (R_1\ op2_p\ R_3), \tag{36}$$

i.e., for every region in the result set we obtain score p:

$$\begin{aligned} p &= (1 \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2) \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3) \\ &= (1 \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2) \cdot (1 \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3)). \end{aligned}$$

If we consider the expression R_4 in the NEXI query 2 we can come up with two query plans shown below.

$$((\text{P} \sqsubset_p R_2) \sqsupset_p \text{INFORMATION}) \sqcap_p ((\text{P} \sqsubset_p R_2) \sqsupset_p \text{RETRIEVAL})$$

and

$$((\text{P} \sqsupset_p \text{INFORMATION}) \sqcap_p (\text{P} \sqsupset_p \text{RETRIEVAL})) \sqsubset_p R_2$$

Although they are almost the same we could not apply property 36 to the first query plan since the scores of regions in the result of the expression $\text{P} \sqsubset_p R_2$ are not equal to 1 in general case. For the second query plan the score value for all regions in P is 1 and the property can be applied. Thus, at the end we can come up with the query plan as shown below:

$$((\text{P} \sqsupset_p \text{INFORMATION}) \sqsupset_p \text{RETRIEVAL}) \sqsubset_p R_2$$

The scoring version of property 21 for score operators $\sqcap_p$ and $\sqcup_p$ does not hold in general. For example, next equation is not true:

$$(R_1 \sqcap_p R_2) \sqsupset_p R_3 = (R_1 \sqsupset_p R_3) \sqcap_p (R_2 \sqsupset_p R_3).$$

It will be true only if $f_{\sqsupset}(r_{1,2}, R_3)$[9]$= 1$ which is not true in general case:

[9]We use $r_{1,2}$ to denote the regions with the same bounds in region sets R_1 and R_2.

$$
\begin{aligned}
p &= (p_1 \cdot p_2) \cdot \sum_{\bar{r}_3} (g(\bar{r}_3, r_{1,2}) \cdot \bar{p}_3) \\
&\neq (p_1 \cdot \sum_{\bar{r}_2} (g(\bar{r}_2, r_{1,2})) \cdot \bar{p}_2) \cdot (p_2 \cdot \sum_{\bar{r}_3} (g(\bar{r}_3, r_{1,2}) \cdot \bar{p}_3)).
\end{aligned}
$$

Similarly, for $op_p = \{\sqsupset_p, \not\supset_p, \sqsubset_p, \not\sqsubset_p\}$ we have:

$$(R_1 \sqcup_p R_2)\ op_p\ R_3 = (R_1\ op_p\ R_3) \sqcup_p (R_2\ op_p\ R_3).$$

i.e.[10],

$$
\begin{aligned}
p &= (p_1 + p_2) \cdot \sum_{\bar{r}_3} (g(\bar{r}_3, r_{1|2}) \cdot \bar{p}_3) \\
&\neq (p_1 \cdot \sum_{\bar{r}_2} (g(\bar{r}_2, r_1)) \cdot \bar{p}_2) + (p_2 \cdot \sum_{\bar{r}_3} (g(\bar{r}_3, r_2) \cdot \bar{p}_3)).
\end{aligned}
$$

5.4 Region algebra and SRA operators

Here we give some issues concerning the combination of score and plain region algebra operators. More elaborate discussion on issues about combining SRA and region algebra operators is left for future research. We can only exert that, as we illustrated before, these issues are highly dependent on the specification of the retrieval model, in other words, on the exact definition of complex functions f (and g).

For the score manipulation operators the distributivity of plain operators (operators that do not change the score values of regions in the left operand) over scoring operators does not hold. This can easily be proven if we consider for example two sides of the inequality given in equation below:

$$R_1\ op1\ (R_2\ op2\ R_3) \neq (R_1\ op1\ R_2)\ op2\ (R_1\ op1\ R_3),$$

where $op1 = \{\sqsupset, \not\supset, \sqsubset, \not\sqsubset\}$ and $op2 \in \{\sqsupset_p, \not\supset_p, \sqsubset_p, \not\sqsubset_p\}$. The left side, $R_1\ op1\ (R_2\ op2\ R_3)$, will always produce the set of regions which are actually a subset of regions in R_1 with the same score values. In that case the following equation is true (when disregarding the score values):

$$R_1\ op1\ (R_2\ op2\ R_3) = R_1\ op1\ R_2$$

The right side $((R_1\ op1\ R_2)\ op2\ (R_1\ op1\ R_3))$ consists of two subexpressions each producing a subset of regions in a region set R_1. To be equal to the left side of the expression the right side of the expression should produce the same regions with the same score values as in the expression $R_1\ op1\ R_2$ which is not the case in general (e.g., it is trivial if $R_2 = R_3$). Similarly if $op1 \in \{\sqsupset_p, \not\supset_p, \sqsubset_p, \not\sqsubset_p\}$ and $op2 = \{\sqsupset, \not\supset, \sqsubset, \not\sqsubset\}$, the left side of the expression will result in regions from region set R_1 with modified score values, while the right side will produce a subset of R_1 (with potentially different score values than on the left side).

In case of $op2 \in \{\sqcap_p, \sqcup_p\}$ the left side will produce regions that contain (or do not contain) regions with the same region bounds as regions in R_2 and R_3, while the right side will produce regions which are the same in the first and in the second subexpression of the right expression. In general these two regions are not equal. For example, if $R_2 \sqcap R_3 = \emptyset$ the score values of regions on the left side of the equation will be 0, and if there exist a region $r_1 \in R_1$ that satisfies both subexpressions on the right side, $R_1\ op1\ R_2$ and $R_1\ op1\ R_3$, this particular region will have the score that is different from 0.

Similar reasoning can be applied to the other combination of region algebra and SRA operators.

[10] We use $r_{1|2}$ to denote the regions which are present in one of the region sets, R_1 or R_2.

6 A Possible Instantiation of Retrieval Functions

Here we illustrate a possible physical instantiation of information retrieval functions specified in the SRA framework based on our INEX 2003 and 2004 approaches [25, 29]. The complex scoring functions defined at the SA logical level, $f_{\sqsupset}(r, R)$ and $f_{\not\sqsupset}(r, R)$, implement the about function specified in NEXI, where '-' in front of the term denote that the element containing that term should be penalized.

The context region in which we perform frequency counts is denoted by ctx. We will make a distinction between the set of term regions, denoted with $\mathcal{W}$, and the set of XML element node regions, denoted with $\mathcal{N}$ (similar as in Section 3). Therefore, selection of terms will be specified as $\sigma_{n=term\_name}(\mathcal{W})$ and selection of element nodes as $\sigma_{n=element\_name}(\mathcal{N})$. We use T to denote arbitrary term region set, e.g., for term "XML" we would have $T = \sigma_{n=\text{"XML"}}(\mathcal{W})$.

We first introduce five auxiliary functions at the physical level, to compute the term frequency - $tf(ctx, T)$, the 'surrounding document' term frequency - $tf'(ctx, T)$, the collection frequency - $cf(T)$, the document frequency - $df(T)$, and the length prior - $lp(ctx)$. The surrounding document term frequency is used in the model as it shows good behavior for retrieval (see [36]). Variable λ represents the smoothing parameter for the inclusion of background statistics and μ is the mean value (i.e., logarithm of the desired size for the element) and ρ is the variance (in our case set to 1) for the log-normal prior.

These auxiliary functions can be implemented using two physical operators: the size operator $size(ctx)$ returns the size of a selected region ctx (i.e., the number of terms in a region ctx), while count operator $|T|$ returns the number of regions (terms) in a (term) region set T.

Function $tf(ctx, T)$ computes the term frequency of term regions in T given the context element node ctx. It is computed as:

$$tf(ctx, T) = \frac{|T \ltimes_{\sqsubset} ctx|}{size(ctx)},$$

where $\ltimes_{\sqsubset}$ can be considered as the physical equivalent of the $\sqsubset$ operator on logical level.

The function $cf(T)$ computes the collection frequency of a (term) region set T as follows:

$$cf(T) = \frac{|T|}{size(Root)},$$

where *Root* represents the region that is not contained by any other region in the collection (i.e., the root region of the entire XML collection).

The document frequency function results in a document frequency for a *term* present in the region set T. We used *article* XML elements as a representation of XML documents[11]. The document frequency function is computed as:

$$df(T) = \frac{|\sigma_{n=\text{"article"}}(\mathcal{N}) \ltimes_{\sqsupset} T|}{|\sigma_{n=\text{"article"}}(\mathcal{N})|}.$$

To define the length prior of the region ctx we can use the size of the element, $lp(ctx) = size(ctx)$, the standard element prior, $lp(ctx) = \log(size(ctx))$, or the log-normal distribution,

$$lp(ctx) = \frac{e^{-((\log(size(ctx)) - \mu)^2 / 2\rho^2)}}{size(ctx)\rho\sqrt{2\pi}}.$$

Finally, the function $tf'(ctx, T)$ computes the term frequency in a surrounding document:

$$tf'(ctx, T) = \frac{|T \ltimes_{\sqsubset} (\sigma_{n=\text{"}article''}(\mathcal{N}) \ltimes_{\sqsupset} ctx)|}{size(\sigma_{n=\text{"}article''}(\mathcal{N}) \ltimes_{\sqsupset} ctx)}.$$

[11] This is due to the organization of the INEX XML collection consisting of IEEE articles that most resemble the notion of a document in traditional information retrieval.

The complex scoring functions (g) defined at the logical level can now be implemented through the combination function h, which is defined in terms of these auxiliary functions and mentioned parameters:

$$g_{\sqsupset,\lambda,\alpha,\mu} = h_{\sqsupset,\lambda,\alpha,\mu}(tf(ctx, T), tf'(ctx, T), cf(T), df(T), lp(ctx)),$$
$$g_{\not\sqsupset,\lambda,\alpha,\mu} = h_{\not\sqsupset,\lambda,\alpha,\mu}(tf(ctx, T), tf'(ctx, T), cf(T), df(T), lp(ctx)).$$

We can now implement different retrieval models, as ones based on the statistical language model approach [21], at the physical level, by instantiating combination function h with the right auxiliary functions. We implemented different variants of retrieval model, using different background statistics and different length priors, as well as using a "fuzzy" and "probabilistic" combination of scores as can be seen in [25]. Therefore, the instantiation of the combination functions determines the actual retrieval model used. Note that some retrieval models may require extension of the logical and/or physical level with new auxiliary functions, e.g., to support other frequency measures.

In [25, 29] we also defined variants of the functions $f_{\sqsupset}(r, R)$ and $f_{\sqsubset}(r, R)$, denoted with $f_{\blacktriangleright}(r, R)$ and $f_{\blacktriangleleft}(r, R)$, that are based on region containment operators and the operators *size* and *count*. These functions specify the propagation of scores among regions based on their containment relation. Therefore, a variant of $f_{\sqsupset}(r, R)$ is defined based on the expression $R \sqsubset r$, while a variant of $f_{\sqsubset}(r, R)$ is based on $R \sqsupset r$. For example, the complex scoring function $f_{\sqsupset}(r, R)$ can be defined as a product of scores in region r and weighted sum of scores for regions R contained in r, normalized by the region size. The function $f_{\sqsubset}(r, R)$ can be defined as sum of a product of scores for regions R containing region r and the score of region r:

$$f_{\sqsupset}(r_1, R_2) = p_1 \cdot \frac{\sum_{r_i \in R_2 \bowtie_{\sqsubset} r_1} (size(r_i) * p_i)}{\sum_{r_i \in R_2 \bowtie_{\sqsubset} R} size(r_i)},$$

$$f_{\sqsubset}(r_1, R_2) = p_1 \cdot \sum_{r_i \in R_2 \bowtie_{\sqsupset} r_1} p_i.$$

For more details on the implementation of SA algebra on physical level we refer to [25, 29].

7 Conclusions and Future Work

In this paper we point out that looking from the database point of view and from the XML querying point of view there is something missing in modeling XML information retrieval. Our premise is that the missing part is the structure aware (SA) view on XML and we try to justify it throughout the paper. We address the problem of translating and executing IR-like queries over XML documents stored in relational databases. We stress the usefulness of the intermediate SA logical level, for which we choose region algebra based on its successful history in the area of search in semi-structured documents as shown in the paper.

Region algebra provides a number of properties that can be used for *query optimization* at the structure aware logical level of a database. Furthermore, the score region algebra (SRA) we introduced can support score operators for ranked retrieval as an integral part of the algebra and not as a side effect, enabling query optimization not only for plain (non-score) but for scoring region algebra operators as well. The expressiveness considering ranked retrieval in our region algebra is far richer than in other region algebra approaches that support ranked retrieval, such as [6] and [26].

An important property of our region algebra is that expressing query plans using the operators, σ, $\sqsupset$, $\sqsubset$, $\not\sqsupset$, $\not\sqsubset$, $\sqcap$, $\sqcup$, $\sqsupset_p$, $\sqsubset_p$, $\not\sqsupset_p$, $\not\sqsubset_p$, $\sqcap_p$, and $\sqcup_p$, defined in Table 5 and 6, preserves *data independence* between the conceptual level, the structure aware logical level, and the underlying physical level of a database. Furthermore, with the abstract specification these operators enable the separation between the structural query processing and the underlying probabilistic model used for ranked retrieval: a design property termed content independence in [11]. Data and

content independence also allow extensions on conceptual level that can be supported on logical and physical level, enabling the application of three-level architecture with the region algebra at the SA level for other XML query languages (e.g., XPath/XQuery full-text search extension [2]) and richer representation of XML data models (e.g., including ID/IDREFs [10]).

An important aspect of introducing SA level in XML DBMS is that we would like to experimentally evaluate the benefits of intermediate level, both in terms of effectiveness and efficiency of such a system. We will work on more advanced retrieval models that incorporate the structure and semantics of XML and can be defined in the framework of score region algebra. Furthermore, capturing our ad hoc approach for score manipulation in region algebra in a well-defined framework (e.g., probability theory), would provide a more elegant way for defining score operators and for the exact specification of SRA operator properties.

We are also planning to further explore the potential usage of region algebra operator properties. This will involve further study on the influence of the definition of score operators (score functions and abstract operators) on score operator properties. Our future research is also concerned with the effects of modifying or changing the retrieval model used, by using different retrieval models as a base (e.g., tf.idf [35], BM25 [33]), by adding different background statistics (i.e., document frequency, element frequency), or by adapting the model for phrase search (see [29]), etc.

References

[1] J. Allen. Time Intervals. In *Communication of the ACM*, volume 26, pages 832–843, 1983.

[2] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. In *Proceedings of the 13th conference on World Wide Web*, pages 583–594, 2004.

[3] R. Baeza-Yates and G. Navarro. Proximal Nodes: A Model to Query Document Databases by Content and Structure. In *ACM Transactions on Information Systems 15 (4)*, volume 15, pages 401–435, 1997.

[4] A. Berglund, S. Boag, D. Chamberlin, M. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0. Technical report, W3C, 2004.

[5] S. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. Technical report, W3C, 2004.

[6] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Texts. In *Proceedings of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–125, 1992.

[7] S. Buxton and M. Rys. XQuery and XPath Full-Text Requirements. Technical report, W3C, 2003.

[8] 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.

[9] M. Consens and T. Milo. Algebras for Querying Text Regions. In *Proceedings of the ACM Conference on Principles of Distributed Systems*, pages 11–22, 1995.

[10] John Cowan and Richard Tobin. XML Information Set (Second Edition). Technical Report REC-xml-infoset-20040204, W3C, February 2004.

[11] A.P. de Vries. Content independence in multimedia databases. *Journal of the American Society for Information Science and Technology*, 52(11):954–960, September 2001.

[12] D. Draper, P. Frankhauser, M. Fernández, A. Malhotra, K. Rose, M. Rys, J. Siméon, and P. Wadler. XML Path Language (XPath) 2.0. Technical report, W3C, 2004.

[13] D. Florescu and I. Manolescu. Integrating Keyword Search into XML Query Processing. In *Proceedings of the 9th International World Wide Web Conference*, pages 67–76, 2000.

[14] N. Fuhr. Models for Integrated Information Retrieval and Database Systems. *IEEE data engineering bulletin*, 19(1):3–13, 1996.

[15] N. Fuhr and K. Grossjohann. XIRQL: A query language for Information Retrieval in XML Documents. In *Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001.

[16] N. Fuhr and K. Großjohann. XIRQL: An XML Query Language Based on Information Retrieval Concepts. *ACM TOIS*, 22(2):313–356, 2004.

[17] N. Fuhr, M. Lalmas, and S. Malik, editors. *Proceedings of the Second Workshop of the INitiative for the Evaluation of XML retrieval (INEX)*, ERCIM Publications, 2004.

[18] T. Grust. Accelerating XPath Location Steps. In *Proceedings of the 21st ACM SIGMOD International Conference on Management of Data*, pages 109–120, 2002.

[19] T. Grust, S. Sakr, and J. Teubner. XQuery on SQL Hosts. In *Proceedings of the 30th Int'l Conference on Very Large Data Bases (VLDB)*, 2004.

[20] T. Grust and M. van Keulen. Tree Awareness for Relational DBMS Kernels: Staircase Join. In H. M. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors, *Intelligent Search on XML*, volume 2818 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 179–192. Springer-Verlag, Berlin, New York, etc., August 2003.

[21] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Twente, The Netherlands, 2001.

[22] D. Hiemstra. A database approach to content-based XML retrieval. In *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval*, 2002.

[23] J. Jaakkola and P. Kilpelainen. Using sgrep for Querying Structured Text Files. Technical Report C-1996-83, Department of Computer Science, University of Helsinki, 1996.

[24] J. Jaakkola and P. Kilpelainen. Nested Text-Region Algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki, 1999.

[25] J. List, V. Mihajlović, A. de Vries, G. Ramirez, and D. Hiemstra. The TIJAH XML-IR System at INEX 2003. In *Proceedings of the 2nd Initiative on the Evaluation of XML Retrieval (INEX 2003)*, ERCIM Workshop Proceedings, 2004.

[26] K. Masuda. A Ranking Model of Proximal and Structural Text Retrieval Based on Region Algebra. In *Proceedings of the ACL-2003 Student Research Workshop*, pages 50–57, 2003.

[27] K. Masuda, T. Ninomiya, Y. Miyao, T. Ohta, and J. Tsujii. A Robust Retrieval Engine for Proximal and Structural Search. In *Proceedings of HLT-NAACL 2003 Short papers*, pages 58–60, 2003.

[28] V. Mihajlović, D. Hiemstra, and P. Apers. On Region Algebras, XML Databases, and Information Retrieval. In *Proceedings of the 4th Dutch-Belgian Information Retrieval Workshop*, 2003.

[29] V. Mihajlović, G. Ramírez, A. P. de Vries, D. Hiemstra, and H. E. Blok. TIJAH at INEX 2004: Modeling Phrases and Relevance Feedback. In *Proceedings of the Third Workshop of the INitiative for the Evaluation of XML retrieval (INEX), to appear*, 2005.

[30] R.C. Miller. *Light-Weight Structured Text Processing*. PhD thesis, Computer Science Department, Carnegie-Mellon University, 2002.

[31] R.C. Miller and B.A. Myers. Lightweight Structured Text Processing. In *Proceedings of 1999 USENIX Annual Technical Conference*, pages 131–144, 1999.

[32] G. Ramirez and A. P. de Vries. Combinining Indexing Schemes to Accelerate Querying XML on Content and Structure. In *Proceedings of the first Twente Data Management Workshop (TDM'04),* to apear, 2004.

[33] S. E. Robertson and S. Walker. Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval. In *Proceedings of 17th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–241, 1994.

[34] A. Salminen and F.W. Tompa. PAT Expressions: An Algebra for Text Search. In *Proceedings of the 2nd International Conference in Computational Lexicography, COMPLEX'92*, pages 309–332, 1992.

[35] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGrow-Hill, New York, NY, USA, 1st edition, 1983.

[36] B. Sigurbjörnsson, J. Kamps, and M. de Rijke. An element-based approach to XML retrieval. In N. Fuhr, M. Lalmas, and S. Malik, editors, *Proceedings of the Second Workshop of the INitiative for the Evaluation of XML retrieval (INEX)*, ERCIM Publications, 2004.

[37] A. Trotman and R. A. O'Keefe. The Simplest Query Language That Could Possibly Work. In N. Fuhr, M. Lalmas, and S. Malik, editors, *Proceedings of the Second Workshop of the INitiative for the Evaluation of XML retrieval (INEX)*, ERCIM Publications, 2004.

[38] D. Tsichritzis and A. Klug. The ANSI/X3/SPARC DBMS framework report of the study group on database management systems. *Information systems*, 3:173–191, 1978.

[39] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 425–436, 2001.

A Plain Region Algebra Operator Properties

Here we list the properties of plain region algebra (region algebra without score operators) operators. A numerous examples are given as a proof of different operator properties mentioned in the paper that do not hold[12].

A.1 Operators $\sqsupset$ and $\not\sqsupset$

Identity:

$$R \not\sqsupset Root = R$$
$$R \not\sqsupset \emptyset = R$$

Inverse:

$$Root \not\sqsupset R = \emptyset, R \neq Root$$

Commutativity:

$$R_1 \sqsupset R_2 \neq R_2 \sqsupset R_1$$
$$R_1 \not\sqsupset R_2 \neq R_2 \not\sqsupset R_1$$

Example 1: If $R_1 = \{(5, 25), (50, 75)\}$ and $R_2 = \{(1, 35)\}$ are two SRA sets then $R_1 \sqsupset R_2 = \emptyset$, and $R_2 \sqsupset R_1 = \{(5, 25)\}$. Furthermore, $R_1 \not\sqsupset R_2 = R_1$ and $R_2 \not\sqsupset R_1 = \emptyset$.

Associativity:

$$(R_1 \sqsupset R_2) \sqsupset R_3 \neq R_1 \sqsupset (R_2 \sqsupset R_3)$$
$$(R_1 \not\sqsupset R_2) \not\sqsupset R_3 \neq R_1 \not\sqsupset (R_2 \not\sqsupset R_3)$$

Example 2: If $R_1 = \{(5, 50), (65, 95)\}$, $R_2 = \{(25, 45), (70, 90)\}$, and $R_3 = \{(10, 15), (80, 85)\}$ are three SRA sets then $(R_1 \sqsupset R_2) \sqsupset R_3 = \{(5, 50), (65, 95)\}$ and $R_1 \sqsupset (R_2 \sqsupset R_3) = \{(65, 95)\}$. Similarly, $(R_2 \not\sqsupset R_3) \not\sqsupset R_1 = \{(25, 45)\}$ and $R_2 \not\sqsupset (R_3 \not\sqsupset R_1) = \emptyset$.

Distributivity:

$$R_1 \sqsupset (R_2 \not\sqsupset R_3) \neq (R_1 \sqsupset R_2) \not\sqsupset (R_1 \sqsupset R_3)$$
$$R_1 \not\sqsupset (R_2 \sqsupset R_3) \neq (R_1 \not\sqsupset R_2) \sqsupset (R_1 \not\sqsupset R_3)$$

Example 3: If $R_1 = \{(5, 50), (65, 95)\}$, $R_2 = \{(25, 45), (70, 90)\}$, and $R_3 = \{(10, 15), (80, 85)\}$ are three SRA sets then $R_1 \sqsupset (R_2 \not\sqsupset R_3) = \{(5, 50)\}$ and $(R_1 \sqsupset R_2) \not\sqsupset (R_1 \sqsupset R_3) = \emptyset$. Also $R_1 \not\sqsupset (R_2 \sqsupset R_3) = \{(15, 50)\}$ and $(R_1 \not\sqsupset R_2) \sqsupset (R_1 \not\sqsupset R_3) = \emptyset$.

$$R_1 \sqsupset (R_2 \sqsubset R_3) \neq (R_1 \sqsupset R_2) \sqsubset (R_1 \sqsupset R_3)$$

$$R_1 \not\sqsupset (R_2 \sqsubset R_3) \neq (R_1 \not\sqsupset R_2) \sqsubset (R_1 \not\sqsupset R_3)$$

Example 4: If $R_1 = \{(5, 50), (65, 95)\}$, $R_2 = \{(10, 15), (80, 85)\}$, and $R_3 = \{(25, 45), (70, 90)\}$, are three SRA sets then $R_1 \sqsupset (R_2 \sqsubset R_3) = \{(65, 95)\}$ and $(R_1 \sqsupset R_2) \sqsubset (R_1 \sqsupset R_3) = \emptyset$. Similarly $R_1 \not\sqsupset (R_2 \sqsubset R_3) = \{(5, 50)\}$ and $(R_1 \not\sqsupset R_2) \sqsubset (R_1 \not\sqsupset R_3) = \emptyset$.

$$R_1 \sqsupset (R_2 \sqcap R_3) \neq (R_1 \sqsupset R_2) \sqcap (R_1 \sqsupset R_3)$$

$$R_1 \not\sqsupset (R_2 \sqcap R_3) \neq (R_1 \not\sqsupset R_2) \sqcap (R_1 \not\sqsupset R_3)$$

[12] We used only start and end information since these operands operate only on these region attributes.

Example 5: If $R_1 = \{(5,50),(65,95)\}$, $R_2 = \{(10,15),(80,85)\}$, and $R_3 = \{(25,45),(70,90)\}$, are three SRA sets then $R_1 \sqsupset (R_2 \sqcap R_3) = \emptyset$ and $(R_1 \sqsupset R_2) \sqcap (R_1 \sqsupset R_3) = \{(5,50),(65,95)\}$. Also $R_1 \not\sqsupset (R_2 \sqcap R_3) = \{(5,50),(65,95)\}$ and $(R_1 \not\sqsupset R_2) \sqcap (R_1 \not\sqsupset R_3) = \emptyset$.

$$R_1 \sqsupset (R_2 \sqcup R_3) = (R_1 \sqsupset R_2) \sqcup (R_1 \sqsupset R_3)$$

$$R_1 \not\sqsupset (R_2 \sqcup R_3) \neq (R_1 \not\sqsupset R_2) \sqcup (R_1 \not\sqsupset R_3)$$

Example 6: If $R_1 = \{(25,45),(70,90)\}$, $R_2 = \{(5,50),(65,95)\}$, and $R_3 = \{(10,15),(80,85)\}$, are three SRA sets then $R_1 \not\sqsupset (R_2 \sqcup R_3) = \{(25,45)\}$ and $(R_1 \not\sqsupset R_2) \sqcup (R_1 \not\sqsupset R_3) = \{(25,45),(70,90)\}$. The prof for the first expression is trivial and can be derived from the definition of the operators $\sqsupset$ and $\sqcup$.

A.2 Operators $\sqsubset$ and $\not\sqsubset$

Identity:

$$R \sqsubset Root = R,\ R \neq Root$$
$$R \not\sqsubset \emptyset = R$$

Inverse:

$$\emptyset \not\sqsubset R = \emptyset$$

Commutativity:

$$R_1 \sqsubset R_2 \neq R_2 \sqsubset R_1$$
$$R_1 \not\sqsubset R_2 \neq R_2 \not\sqsubset R_1$$

Example 7: If $R_1 = \{(5,25),(50,75)\}$ and $R_2 = \{(1,35)\}$ are two SRA sets then $R_1 \sqsubset R_2 = \{(5,25)\}$ and $R_2 \sqsubset R_1 = \emptyset$, and similarly $R_1 \not\sqsubset R_2 = \{(50,75)\}$ and $R_2 \not\sqsubset R_1 = \{(1,35)\}$.

Associativity:

$$(R_1 \sqsubset R_2) \sqsubset R_3 \neq R_1 \sqsubset (R_2 \sqsubset R_3)$$
$$(R_1 \not\sqsubset R_2) \not\sqsubset R_3 \neq R_1 \not\sqsubset (R_2 \not\sqsubset R_3)$$

Example 8: If $R_1 = \{(10,15),(80,85)\}$, $R_2 = \{(25,45),(65,95)\}$, and $R_3 = \{(5,50),(70,90)\}$ are three SRA sets then $(R_1 \sqsubset R_2) \sqsubset R_3 = \{(80,85)\}$ and $R_1 \sqsubset (R_2 \sqsubset R_3) = \emptyset$. Also $(R_1 \not\sqsubset R_2) \not\sqsubset R_3 = \emptyset$ and $R_1 \not\sqsubset (R_2 \not\sqsubset R_3) = \{(10,15)\}$.

Distributivity:

$$R_1 \sqsubset (R_2 \sqsupset R_3) \neq (R_1 \sqsubset R_2) \sqsupset (R_1 \sqsubset R_3)$$
$$R_1 \not\sqsubset (R_2 \sqsupset R_3) \neq (R_1 \not\sqsubset R_2) \sqsupset (R_1 \not\sqsubset R_3)$$

Example 9: If $R_1 = \{(10,15),(80,85)\}$, $R_2 = \{(25,45),(65,95)\}$, and $R_3 = \{(5,50),(70,90)\}$ are three SRA sets then $R_1 \sqsubset (R_2 \sqsupset R_3) = \{(80,85)\}$ and $(R_1 \sqsubset R_2) \sqsupset (R_1 \sqsubset R_3) = \emptyset$. Also $R_1 \not\sqsubset (R_2 \sqsupset R_3) = \{(10,15)\}$ and $(R_1 \not\sqsubset R_2) \sqsupset (R_1 \not\sqsubset R_3) = \emptyset$.

$$R_1 \sqsubset (R_2 \not\sqsubset R_3) \neq (R_1 \sqsubset R_2) \not\sqsubset (R_1 \sqsubset R_3)$$

$$R_1 \not\sqsubset (R_2 \sqsubset R_3) \neq (R_1 \not\sqsubset R_2) \sqsubset (R_1 \not\sqsubset R_3)$$

Example 10: If $R_1 = \{(10,15),(80,85)\}$, $R_2 = \{(5,50),(70,90)\}$, and $R_3 = \{(25,45),(65,95)\}$ are three SRA sets then $R_1 \sqsubset (R_2 \not\sqsubset R_3) = \{(10,15)\}$ and $(R_1 \sqsubset R_2) \not\sqsubset (R_1 \sqsubset R_3) = \{(10,15),(80,85)\}$. Also $R_1 \not\sqsubset (R_2 \sqsubset R_3) = \{(10,15)\}$ and $(R_1 \not\sqsubset R_2) \sqsubset (R_1 \not\sqsubset R_3) = \emptyset$.

$$R_1 \sqsubset (R_2 \sqcap R_3) \neq (R_1 \sqsubset R_2) \sqcap (R_1 \sqsubset R_3)$$

$$R_1 \not\sqsubset (R_2 \sqcap R_3) \neq (R_1 \not\sqsubset R_2) \sqcap (R_1 \not\sqsubset R_3)$$

Example 11: If $R_1 = \{(10,15),(80,85)\}$, $R_2 = \{(5,50),(65,95)\}$, and $R_3 = \{(25,45),(70,90)\}$ are three SRA sets then $R_1 \sqsubset (R_2 \sqcap R_3) = \emptyset$ and $(R_1 \sqsubset R_2) \sqcap (R_1 \sqsubset R_3) = \{(80,85)\}$. Similarly $R_1 \not\sqsubset (R_2 \sqcap R_3) = \{(10,15),(80,85)\}$ and $(R_1 \not\sqsubset R_2) \sqcap (R_1 \not\sqsubset R_3) = \emptyset$.

$$R_1 \sqsubset (R_2 \sqcup R_3) = (R_1 \sqsubset R_2) \sqcup (R_1 \sqsubset R_3)$$

$$R_1 \not\sqsubset (R_2 \sqcup R_3) \neq (R_1 \not\sqsubset R_2) \sqcup (R_1 \not\sqsubset R_3)$$

Example 12: If $R_1 = \{(10,15),(80,85)\}$, $R_2 = \{(5,50),(65,95)\}$, and $R_3 = \{(25,45),(70,90)\}$ are three SRA sets then $R_1 \not\sqsubset (R_2 \sqcup R_3) = \emptyset$ and $(R_1 \not\sqsubset R_2) \sqcup (R_1 \not\sqsubset R_3) = \{(10,15)\}$. The first equality can be easily proven following the definition of operators $\sqsubset$ and $\sqcup$.

A.3 Operator $\sqcap$

Identity:

$$R \sqcap C = R$$
$$C \sqcap R = R$$

Inverse:

Commutativity:

$$R_1 \sqcap R_2 = R_2 \sqcap R_1$$

Associativity:

$$(R_1 \sqcap R_2) \sqcap R_3 = R_1 \sqcap (R_2 \sqcap R_3)$$

Distributivity:

$$R_1 \sqcap (R_2 \sqsupset R_3) \neq (R_1 \sqcap R_2) \sqsupset (R_1 \sqcap R_3)$$
$$R_1 \sqcap (R_2 \sqsubset R_3) \neq (R_1 \sqcap R_2) \sqsubset (R_1 \sqcap R_3)$$

Proof: In case $R_1 \equiv R_2$ the equations become R_2 *op* $R_3 \neq R_2$ *op* $(R_2 \sqcap R_3)$, where *op* $= \{\sqsupset, \sqsubset\}$. If there exist a region in R_2 such that R_2 *op* $R_3 \neq \emptyset$, and $R_2 \sqcap R_3 = \emptyset$ then R_2 *op* $(R_2 \sqcap R_3) = \emptyset$ and the equations are true.

$$R_1 \sqcap (R_2 \not\sqsupset R_3) \neq (R_1 \sqcap R_2) \not\sqsupset (R_1 \sqcap R_3)$$

$$R_1 \sqcap (R_2 \not\sqsubset R_3) \neq (R_1 \sqcap R_2) \not\sqsubset (R_1 \sqcap R_3)$$

Proof: If $(R_1 \sqcap R_3) = \emptyset$, and there exist a region set R' such that $R_1 \sqcap R_1 = R' \neq \emptyset$ and R' *op* $R_3 = \emptyset$ for *op* $= \{\not\sqsupset, \not\sqsubset\}$, the left side of the equation will become $R_1 \sqcap (R_2$ *op* $R_3) = \emptyset$ and the right side will be $(R_1 \sqcap R_2)$ *op* $(R_1 \sqcap R_3) = R'$.

$$R_1 \sqcap (R_2 \sqcup R_3) = (R_1 \sqcap R_2) \sqcup (R_1 \sqcap R_3)$$

A.4 Operator $\sqcup$

Identity:

$$R \sqcup \emptyset = R$$
$$\emptyset \sqcup R = R$$

Inverse:

Commutativity:

$$R_1 \sqcup R_2 = R_2 \sqcup R_1$$

Associativity:

$$(R_1 \sqcup R_2) \sqcup R_3 = R_1 \sqcup (R_2 \sqcup R_3)$$

Distributivity:

$$R_1 \sqcup (R_2 \sqsupset R_3) \neq (R_1 \sqcup R_2) \sqsupset (R_1 \sqcup R_3)$$
$$R_1 \sqcup (R_2 \sqsubset R_3) \neq (R_1 \sqcup R_2) \sqsubset (R_1 \sqcup R_3)$$

Proof: If there exist a region set $R' \subset R_1$ which is unique (i.e., $R' \sqcap R_2 = \emptyset$ and $R' \sqcap R_3 = \emptyset$), and R' *op* $R' = \emptyset$ for $op = \{\sqsupset, \sqsubset\}$, the region set R' will be in the result region set on the left side but not in the result of the right side of the equation.

$$R_1 \sqcup (R_2 \not\sqsupset R_3) \neq (R_1 \sqcup R_2) \not\sqsupset (R_1 \sqcup R_3)$$

$$R_1 \sqcup (R_2 \not\sqsubset R_3) \neq (R_1 \sqcup R_2) \not\sqsubset (R_1 \sqcup R_3)$$

Proof: If there exist a region set $R' \subset R_1$ which is unique (i.e., $R' \sqcap R_2 = \emptyset$ and $R' \sqcap R_3 = \emptyset$), and R' *op* $R' = \emptyset$ for $op = \{\not\sqsupset, \not\sqsubset\}$, the region set R' will be in the result region set on the left side but not in the result of the right side of the equation.

$$R_1 \sqcup (R_2 \sqcap R_3) = (R_1 \sqcup R_2) \sqcap (R_1 \sqcup R_3)$$

A.5 Special cases of associativity and distributivity

Special cases of operator associativity:

$$(R_1 \sqsupset R_2) \sqsupset R_3 = (R_1 \sqsupset R_3) \sqsupset R_2$$

$$(R_1 \not\sqsupset R_2) \not\sqsupset R_3 = (R_1 \not\sqsupset R_3) \not\sqsupset R_2$$
$$(R_1 \sqsubset R_2) \sqsubset R_3 = (R_1 \sqsubset R_3) \sqsubset R_2$$
$$(R_1 \not\sqsubset R_2) \not\sqsubset R_3 = (R_1 \not\sqsubset R_3) \not\sqsubset R_2$$
$$(R_1 \sqsupset R_2) \sqsubset R_3 = (R_1 \sqsubset R_3) \sqsupset R_2$$
$$(R_1 \not\sqsupset R_2) \not\sqsubset R_3 = (R_1 \not\sqsubset R_3) \not\sqsupset R_2$$
$$(R_1 \sqsubset R_2) \sqsupset R_3 = (R_1 \sqsupset R_3) \sqsubset R_2$$
$$(R_1 \not\sqsubset R_2) \not\sqsupset R_3 = (R_1 \not\sqsupset R_3) \not\sqsubset R_2$$

Special cases of operator normalization:

$$(R_1 \sqsupset R_2) \sqsupset R_3 = (R_1 \sqsupset R_2) \sqcap (R_1 \sqsupset R_3)$$

$$(R_1 \not\sqsupset R_2) \not\sqsupset R_3 = (R_1 \not\sqsupset R_2) \sqcap (R_1 \not\sqsupset R_3)$$
$$(R_1 \sqsubset R_2) \sqsubset R_3 = (R_1 \sqsubset R_2) \sqcap (R_1 \sqsubset R_3)$$
$$(R_1 \not\sqsubset R_2) \not\sqsubset R_3 = (R_1 \not\sqsubset R_2) \sqcap (R_1 \not\sqsubset R_3)$$
$$(R_1 \sqsupset R_2) \sqsubset R_3 = (R_1 \sqsupset R_2) \sqcap (R_1 \sqsubset R_3)$$
$$(R_1 \not\sqsupset R_2) \not\sqsubset R_3 = (R_1 \not\sqsupset R_2) \sqcap (R_1 \not\sqsubset R_3)$$
$$(R_1 \sqsubset R_2) \sqsupset R_3 = (R_1 \sqsubset R_2) \sqcap (R_1 \sqsupset R_3)$$
$$(R_1 \not\sqsubset R_2) \not\sqsupset R_3 = (R_1 \not\sqsubset R_2) \sqcap (R_1 \not\sqsupset R_3)$$

Set-containment operator distributivity:

$$(R_1 \sqcap R_2) \sqsupset R_3 = (R_1 \sqsupset R_3) \sqcap (R_2 \sqsupset R_3)$$

$$(R_1 \sqcap R_2) \not\sqsupset R_3 = (R_1 \not\sqsupset R_3) \sqcap (R_2 \not\sqsupset R_3)$$
$$(R_1 \sqcap R_2) \sqsubset R_3 = (R_1 \sqsubset R_3) \sqcap (R_2 \sqsubset R_3)$$
$$(R_1 \sqcap R_2) \not\sqsubset R_3 = (R_1 \not\sqsubset R_3) \sqcap (R_2 \not\sqsubset R_3)$$
$$(R_1 \sqcup R_2) \sqsupset R_3 = (R_1 \sqsupset R_3) \sqcup (R_2 \sqsupset R_3)$$
$$(R_1 \sqcup R_2) \not\sqsupset R_3 = (R_1 \not\sqsupset R_3) \sqcup (R_2 \not\sqsupset R_3)$$
$$(R_1 \sqcup R_2) \sqsubset R_3 = (R_1 \sqsubset R_3) \sqcup (R_2 \sqsubset R_3)$$
$$(R_1 \sqcup R_2) \not\sqsubset R_3 = (R_1 \not\sqsubset R_3) \sqcup (R_2 \not\sqsubset R_3)$$

B Properties of Scoring Operators in SRA

This appendix lists the properties of scoring operators in SRA with a short description of the region scoring results after the application of score operators. We give only the most important properties that hold.

B.1 Operator $\sqcap_p$

Identity:

$$R \sqcap_p C = R, \text{ i.e., } p \cdot 1 = p, \forall r \in R$$
$$C \sqcap_p R = R, \text{ i.e., } 1 \cdot p = p, \forall r \in R$$

Inverse:

Commutativity:

$$R_1 \sqcap_p R_2 = R_2 \sqcap_p R_1$$

i.e., $p_1 \cdot p_2 = p_2 \cdot p_1, \forall r_1, r_2 \in R_1 \sqcap R_2$

Associativity:

$$(R_1 \sqcap_p R_2) \sqcap_p R_3 = R_1 \sqcap (R_2 \sqcap_p R_3)$$

i.e., $(p_1 \cdot p_2) \cdot p_3 = p_1 \cdot (p_2 \cdot p_3), \forall r_1, r_2, r_3 \in (R_1 \sqcap R_2) \sqcap R_3$

Distributivity:

$$R_1 \sqcap_p (R_2 \sqcup_p R_3) = (R_1 \sqcap_p R_2) \sqcup_p (R_1 \sqcap_p R_3)$$

i.e., $p_1 \cdot (p_2 + p_3) = (p_1 \cdot p_2) + (p_1 \cdot p_3), \forall r_1, r_2, r_3 \in R_1 \sqcap (R_2 \sqcup R_3)$

B.2 Operator $\sqcup_p$

Identity:

$$R \sqcup_p \emptyset = R, \text{ i.e., } p + 0 = p, \forall r \in R$$
$$\emptyset \sqcap_p R = R, \text{ i.e., } 0 + p = p, \forall r \in R$$

Inverse:

Commutativity:

$$R_1 \sqcup_p R_2 = R_2 \sqcup_p R_1$$

i.e., $p_1 + p_2 = p_2 + p_1, \forall r_1, r_2 \in R_1 \sqcap R_2$

Associativity:

$$(R_1 \sqcup_p R_2) \sqcup_p R_3 = R_1 \sqcup (R_2 \sqcup_p R_3)$$

i.e., $(p_1 + p_2) + p_3 = p_1 + (p_2 + p_3), \forall r_1, r_2, r_3 \in (R_1 \sqcup R_2) \sqcup R_3$

Distributivity:

B.3 Operators $\sqsupset_p$, $\not\sqsupset_p$, $\sqsubset_p$, and $\not\sqsubset_p$

Properties of these operators are highly dependant on the instantiation of retrieval function.

B.4 Additional SRA operator properties

Special cases of score operator associativity:

For $op1_p = \{\sqsupset_p, \not\sqsupset_p, \sqsubset_p, \not\sqsubset_p\}$ and $op2_p = \{\sqsupset_p, \not\sqsupset_p, \sqsubset_p, \not\sqsubset_p\}$:

$$(R_1\ op1_p\ R_2)\ op2_p\ R_3 = (R_1\ op2_p\ R_3)\ op1_p\ R_2$$

i.e., for every region in the result set we obtain score p:

$$\begin{aligned} p &= (p_1 \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2)) \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3) \\ &= (p_1 \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3)) \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2), \end{aligned}$$

where $g(\bar{r}, r)$ is one of the functions $g_{\sqsupset}(\bar{r}, r)$, $g_{\not\sqsupset}(\bar{r}, r)$, $g_{\sqsubset}(\bar{r}, r)$, or $g_{\not\sqsubset}(\bar{r}, r)$, and $\bar{r} \in R \sqsupset R'$, $\bar{r} \in R \not\sqsupset R'$, $\bar{r} \in R' \sqsubset R$, or $\bar{r} \in R' \not\sqsubset R$ is based on the type of operators $op1_p$ and $op2_p$.

Special cases of score operator normalization:

For $op1_p = \{\sqsupset_p, \not\sqsupset_p, \sqsubset_p, \not\sqsubset_p\}$ and $op2_p = \{\sqsupset_p, \not\sqsupset_p, \sqsubset_p, \not\sqsubset_p\}$ and if next condition is satisfied $\forall r_1 \in R_1,\ p_1 = 1\ \wedge\ \forall r_2 \in R_2,\ r_3 \in R_3,\ p_2 = p_3 = c$:

$$(R_1\ op1_p\ R_2)\ op2_p\ R_3 = (R_1\ op1_p\ R_2) \sqcap_p (R_1\ op2_p\ R_3)$$

i.e., for every region in the result set we obtain score p:

$$\begin{aligned} p &= (1 \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2) \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3) \\ &= (1 \cdot \sum_{\bar{r}_2}(g(\bar{r}_2, r_1) \cdot \bar{p}_2) \cdot (1 \cdot \sum_{\bar{r}_3}(g(\bar{r}_3, r_1) \cdot \bar{p}_3)). \end{aligned}$$

where $g(\bar{r}, r)$ is one of the functions $g_{\sqsupset}(\bar{r}, r)$, $g_{\not\sqsupset}(\bar{r}, r)$, $g_{\sqsubset}(\bar{r}, r)$, or $g_{\not\sqsubset}(\bar{r}, r)$, and $\bar{r} \in R \sqsupset R'$, $\bar{r} \in R \not\sqsupset R'$, $\bar{r} \in R' \sqsubset R$, or $\bar{r} \in R' \not\sqsubset R$ is based on the type of operators $op1_p$ and $op2_p$.