Expressive Power of a New Model for Structured Text Databases*

Gonzalo Navarro Ricardo Baeza-Yates

Department of Computer Science
University of Chile
Blanco Encalada 2120
Santiago - Chile
{gnavarro,rbaeza}@dcc.uchile.cl

Abstract

This paper studies the expressivity of a new model for structuring and querying textual databases by both the structure and contents of the text. The key idea of the model is a set-oriented query language based on operations on proximal nodes. This model has been shown to be efficiently implementable, and the aim of this paper is to show that it is competitive in expressivity with models whose implementation is not so efficient. The expressivity is studied by directly comparing it against related models, and by defining a framework on expressivity where these models are situated. This work leads to the conclusion that the presented model is a good compromise between expressivity and efficiency.

*This work has been supported in part by grants FONDECYT (Chile) 1940271 and 1950622.

1 Introduction

Textual databases are deserving more and more attention, due to their multiple applications: libraries, office automation, software engineering, automated dictionaries and encyclopedias, and in general any problem based on keeping and retrieving textual information [9].

The purpose of a textual database is to store textual documents, structured or not. A textual database is composed by two parts: contents and structure (if present). The content is the text itself, while the structure relates different parts of the database with some criteria. Any information model related to textual databases must comprise three aspects: a text model, a structure model, and a query language.

Unlike other kinds of databases, a text database is characterized by the fact that there is no easy way to extract what the user wants. The user must specify what he/she wants, see the results, then reformulate the query, and so on, until is satisfied with the answer. Anything we can do to help users to find what they want is worth considering.

Another interesting fact is that human beings have "visual memory", e.g. they may remember that what they want was typed in *italics*, short before a figure that said something about "earth". Searching for the word "earth" may not be a good idea, as may not be searching all figures or all the text in italics. What really would help to exploit visual memory would be a language in which we can say "I want a text on italics, near a figure containing the word 'earth' ". This query mixes content and structure of the database.

Traditionally, textual databases have allowed their users to search their contents (words, phrases, etc.) or their structure (e.g. by looking at a table of contents).

Mixing contents and structure allows to pose very powerful queries, being much more expressive than each mechanism by itself. By using a query language that integrates both types of queries, we can potentiate the retrieval quality of our textual databases.

The query language we present is not necessarily intended for final users, rather it is an operational algebra onto which a more user-oriented query language can be mapped.

The aim of this paper is to study the expressivity of a new model to structure and query textual databases, which has been presented in [15, 16] and shown to have an efficient implementation. We want to show that this model is competitive in expressivity with others that are not efficiently implementable.

This paper is organized as follows. In Section 2, related work on this subject is reviewed. In Section 3, our model is informally presented, in terms of the data model and the operations allowed for queries. In Section 4, we analyze its expressivity by comparing it against each similar model we reviewed. In Section 5, we draw an informal framework on expressivity to situate similar models. Finally, in Section 6, our conclusions and future work directions are presented.

2 Related Work

In this section we cover previous approaches to the problem of querying a textual database. We first mention the traditional ones, and then cover novel ideas.

2.1 Traditional Approaches

There are many classical approaches to the problem of querying a textual database. Some of them are: attempts to adapt the relational model to include text management [19, 7]; the many traditional models for information retrieval (e.g. the boolean model, the probabilistic model, the bit-vector model, the full-text model, etc.) [9]; hypertexts and semantic networks; and object-oriented databases adapted to manage text [2].

None of these approaches satisfy our goals of mixing structure and contents in our queries. Some of them do not allow to express rich enough structures, some are too oriented to contents, some to structure. Finally, although object-oriented databases can be extended to successfully combine both areas, they are too general and do not fully exploit the semantics involved in structure. They represent a structure merely as a network and have a query language oriented to those graphs, this way making very inefficient queries that are simple if we know the inclusion semantics involved in the structure (see [5] for an excellent discussion on this topic).

Although these models are not powerful enough to extract the information we want from textual databases, they address different problems that pure textual database models oriented to structure do not address in general (e.g. tuples and joins, attributes, etc.). We do not compare our model to these, because they address different goals.

In [17] it is argued that is better to put a layer integrating a traditional database system with a textual one, than trying to design a language comprising all the features. This way, each subsystem focuses on the part of the query in which specializes (e.g. [5] integrates an object-oriented database with a structured text engine).

We rely on this approach. We design a language which is focused on exploiting the structure- and text-related features. Other features, such as tuples and joins, should be added by integrating this language with another one oriented to that kind of operations, e.g. a relational database.

On the other hand, we do not address the issue of merging structural queries with those involving operations such as relevance ranking (e.g. the sections or titles where the word "computer" is relevant). See [17] for some ideas on this topic.

2.2 Novel Approaches

These approaches are characterized by generally imposing a hierarchical structure on the database, and by mixing queries on contents and structure. Although this is simpler than, for example, hypertexts, even in this simpler case the problem of mixing contents and structure is not satisfactorily solved.

We present a sample of novel models, which cover many different approaches to solve this problem under the stated conditions. See [13] for another survey.

The Hybrid Model [1]: modelizes a textual database as a set of documents, which may have fields. Those fields need not to cover all the text of the document, and can nest and overlap. The query language is an algebra over pairs (D, M), where D is a set of documents and M is a set of match points in those documents. There is a number of operations for obtaining match points: prefix search, proximity, etc. There are

operations for set manipulation of both documents and match points; for restricting matches to only some fields; and for retrieving fields owning some match point. Inclusion relationships can only be queried with respect to a field and a match point, thus the language is flat and not fully compositional. This model can be implemented very efficiently.

PAT Expressions [18, 8]: sees only match points, which are used to define *regions*. Regions are defined by match expressions that specify how their endpoints are. Each region represents a set of disjoint segments. This allows dynamic definition of regions, and to match all queries on regions to queries on matches. The need to avoid overlapping regions cause a lot of trouble and lack of orthogonality in the language. This language achieves high efficiency at the cost of some restrictions, which for some applications are reasonable.

Overlapped Lists [3, 4]: solves the problem of PAT expressions in an elegant way, by allowing overlaps, but not nesting. Each region is a list of (possibly overlapping) segments, originated by textual searches or by named regions (like chapters, for example). The idea is to unify both searches by using an extension of inverted lists, where regions and words are indexed the same way. The implementation of this model can be as efficient as that of PAT expressions.

Lists of References [14]: is a general model to structure and query textual databases, including also hypertext-like linkages, attribute management and external procedures. The structure of documents can be hierarchical (no overlaps), but answers to queries are flat (only the top-level elements qualify), and all elements must be from the same type (e.g. only sections, or only paragraphs). Answers to queries are seen as lists of references (i.e. pointers to the database). This allows to integrate in an elegant way answers to queries to hypertext links, since all are seen as lists of references. This model is very powerful, but hard to implement efficiently [14]. To make the model suitable for comparison, we consider only the portion related to querying structures. Even this portion is quite powerful.

Parsed Strings [10]: is in fact a data manipulation language. The language used to express database schemas is a context-free grammar, that is, the database is structured by giving a grammar to parse its text. The fundamental data structure is the *p-string*, or parsed string, which is composed of a derivation tree plus the underlying text. The manipulation is carried out via a number of powerful operations to transform trees. The approach is extremely powerful, and it is shown to be relationally complete. However, it is hard to implement efficiently.

Tree Matching [12]: is a query model relying on a single primitive: tree inclusion. The idea is, seeing both the structure of the database and the query (a pattern on structure) as trees, to find an embedding of the pattern into the database which respects the hierarchical relationships between nodes of the pattern. The language is enriched by Prolog-like variables, which can be used to express requirements on equality between parts of the matched substructure, and to retrieve another part of the match, not only

the root. The complexity of the algorithms is studied, showing that the only case in which the problem is of polynomial time is when no logical variables are used and the matches have to respect the left-to-right ordering in the pattern. Even in the polynomial case, the operations have to traverse the whole database structure to find the matches.

3 A New Model for Querying Structured Text

In this section we outline the main features of our model. A more complete presentation can be found in [15, 16].

3.1 Data Model

We see a text database as composed of two parts:

- Text, which is seen as a (long) array of symbols. Whether this text is stored as it is seen, or it is filtered to hide markup or uninteresting components, is not important for the model, since we use the logical view of the text. This may include also elimination of stopwords, replacement of synonyms, etc. Additionally, symbols may be characters, words, etc.

- Structure, which is organized as a set of independent (orthogonal) hierarchies. Each hierarchy has its own types of nodes, and the areas covered by the nodes of different hierarchies can overlap, although this cannot happen inside the same hierarchy. They do not need to cover the whole text.

Each view has a set of *constructors*, which denote types of nodes of the corresponding tree. Examples of constructors are page, chapter and section. The sets of constructors of different views are disjoint.

Each node of the tree corresponding to a view has an associated constructor, and a *segment*, which is pair of numbers representing a contiguous portion of the underlying text. The segment of a node must include the segments of its children in the tree (this inclusion needs not to be strict).

Text subqueries are introduced in the model by considering any set of disjoint segments as belonging to a special *text view*.

3.2 Query Language

We have a set-at-a-time query language, where both operands and answers are sets of nodes, and nodes can be selected by regarding their context in the structure. All sets are subsets of a single view.

The idea of the query language is that the leaves of the query syntax trees (pattern-matching expressions and basic constructor names) must be solved by using some indexing scheme that avoids the need to traverse the whole database to find the answers. That is, we use a bottom-up evaluation approach. Thereafter, the operators work on sets of nodes

to produce new sets of nodes. All these operators work on *proximal* nodes, i.e. nodes whose segments are more or less proximal. This way, by using an appropriate arrangement to keep the nodes of each set (operand), we can perform all operations very efficiently.

This way, we do not define a monolithic, comprehensive query language, but point out a number of operations that are coherent with this philosophy, and consequently can be efficiently implemented. Most operations fitting into this scheme can be implemented in linear time. We refer the reader to [15] for implementation details.

Our aim is to show that a set of operators that operate exclusively on proximal nodes and regarding only their identity and associated segment can be very expressive.

Figure 1 outlines the schema of the operations. There are basic extraction operations (forming the basis of querying on structure and on contents), and there are operations to combine results from others, which are classified in a number of groups: those which operate by considering included elements, including elements, nearby elements, to manipulate sets and by direct structural relationships.

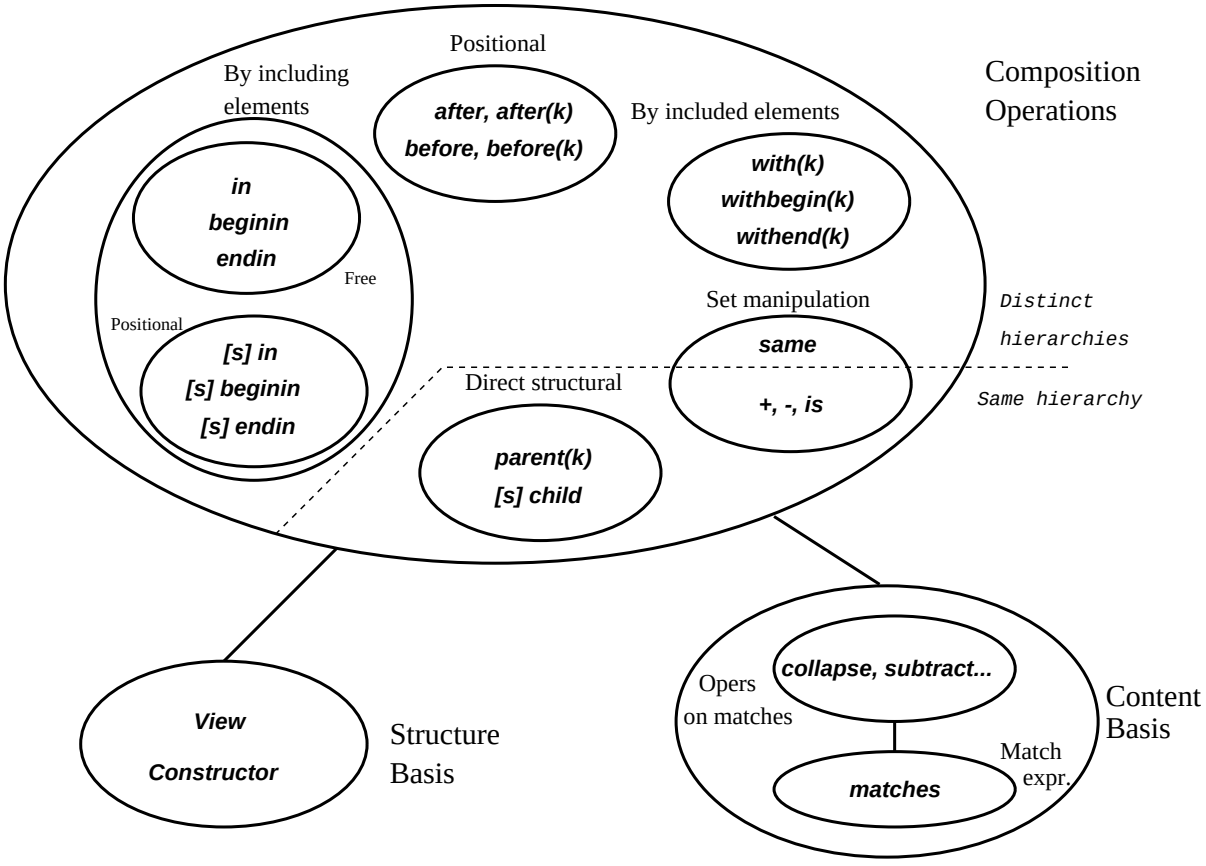

Figure 1: The operations of our model, classified by type.

4 Comparison with Similar Models

The aim of this section is to study the expressivity of our model, by comparing it against the other models we cover. The formal semantics of our model and the equations to translate it to others and vice versa are presented in [15]. Because of their length, we present only the main points in this section.

The Hybrid Model: Our model can completely represent this one, by means of two expressions that return the set of resulting documents and the set of resulting matches (size-1 text segments). On the other hand, this model can represent little from ours.

PAT Expressions: Our model can represent this one almost completely (except some undesirable features). We represent answers as a set of text regions corresponding to the "areas" of this model. On the other hand, we have to leave out a lot of structuring power to embed our model into this one (all our structures have to be flat, for example).

Overlapped Lists: Our model can represent this one, except for the overlapping feature, which is in fact central to this model. We can represent, as text regions, the resulting segments, although collapsed when they overlap. On the other hand, this model cannot represent nested answers, which is a central feature of our model. Apart from that restriction, an interesting number of operators can be translated.

Lists of References: We can represent all of this model (except its attribute management and hypertextual features, that we left aside from this comparison). On the other hand, this model poses some strong restrictions to embed our model into it: answers have to be flat and a subset of a single constructor. Also, only one hierarchy exists. Except for these (important) restrictions, an interesting subset of our operators can be translated into this one.

Tree Matching: We are not able to fully represent this model. The reason is that the tree pattern-matching semantics impose that any labeling, ancestorship and (optionally) ordering relations hold in the match point if and only if they hold in the pattern. We cannot express this except under certain restrictions. Moreover, we cannot represent any of the features of this model regarding logical variables. On the other hand, this model can represent some of ours, being its main limitations to have a single hierarchy, not being able to express direct structural relations and its weak link between text and structure.

5 A Comparison Framework

In this section, we draw an informal framework to situate the expressivity of any similar model.

Our aim is to make a move in the direction of unifying the diversity present in the similar approaches, in order to situate them in a common framework, to reasonably compare their distinguishing features.

Finding a model of expressivity as it could be the hierarchy of formal languages is certainly an ambitious goal (a first step in this direction could be [6]). We content ourselves with pointing out a number of aspects in which (at least) a model should be examined in order to analyze its expressivity.

In [17], a number of queries that this kind of language should be able to answer are pointed out. We show in [15] that we can express all in the areas we are interested in (i.e. excluding the features related to relevance ranking and connection to relational databases, which we do not address in this model).

Another attempt to classify these kind of models is made in [13], which surveys a number of approaches to structured text retrieval. In that case, the study is broader, since it also covers indexing and editing aspects, but its focus on data models and query languages is not so deep as in this one. Moreover, they have only two models under study in common.

5.1 A Methodology to Analyze a Language

We want to make a stricter analysis that the one done in [17], since its requirements are also fulfilled by less powerful languages. We divide our analysis in three main areas, and pose some example questions related to that area. More details can be found in [15].

Structuring mechanism: It refers to the capabilities of the language to express the structure of a textual database. Some questions one should ask here are: is it possible to express a hierarchy? is there any limit on it? can multiple hierarchies be expressed?

Query language for contents: It refers to the part of the query language related to the text of the database, and especially the way to relate it to the structure. Some important questions are: how is the string matching sublanguage? how is a matching subquery inserted in the context of a structural query? how is the text seen in the model? how can restrictions on distances be expressed? are text segments first-class objects? (e.g. can they be retrieved, can we ask if they include something, etc.).

Query language for structure: It refers to the part of the query language related to the structure of the database. Important questions are: how can ancestorship be expressed? can it distinguish between direct and transitive relations? can it discriminate ordering or positions among siblings? which are the set manipulation features?

5.2 A Brief Analysis

Table 1 answers the posed questions for the models we have been analyzing. We are going to be very brief here. Recall that we disregard matching sublanguages in this analysis.

In Figure 2 we present a graphical version of this analysis. The main desirable features are presented, and each model is represented as a set of the features it supports. Recall that we only consider part of the lists-of-references model.

From the figure, we can see that the main features lacking in our model are tuples, semijoin by contents and the possibility of having overlaps and combined nodes in the result set of a query.

Area	Structuring	Querying contents	Querying structure
Our model	A set of disjoint strict trees (views), with no more restrictions. Views can overlap.	Text is a special view. Nodes cannot be dissociated from segments. Text queries are leaves of query syntax trees. There are powerful distance operators. Text content is accessed only in matching subqueries, thereafter it is seen as segments. There are special set operators for text.	Can express inclusion, positions, direct and transitive relations; discriminates ordering (with restrictions) and manipulates sets. Cannot express relationships between different parts of the structure. Can express context conditions relating proximal nodes.
Hybrid model [1]	IR-like documents + fields + text. Fields can nest and overlap, but it is a flat model.	Query = matches + documents. Almost all the language is oriented to matches, which are seen as their start point. Expresses distances. Has separate set manipulation tools for matches and documents.	Only to restrict matching points to be in a given field or to select fields including matching points (selected fields are then seen as matching points).
PAT expressions [18]	Dynamic definition of regions, by pattern matching. Each region is a flat list of disjoint segments.	Powerful matching language. Has matching points and regions. Regions are just segments. Has set manipulation operations. Expresses distances.	Structures are flat. Can express inclusion, set manipulation and little more.
Overlapped lists [3]	A set of regions, each one a flat list of possibly overlapping segments.	Not specified. Words and regions are seen in a uniform way, by an inverted list metaphor.	Results are flat, although they can overlap. Can express inclusion, union and combinations ($\diamond, \triangle$).
Reference lists [14]	A single hierarchy with attributes in nodes and hypertext links.	Text queries can only be used to restrict other queries.	Results are flat and from the same constructor. Can express inclusions, complex context conditions and set manipulation.
Tree matching [12]	A single tree, with strict hierarchy. No more restrictions.	Not specified, orthogonal to the model. Apparently it can only be used to restrict sets of nodes of the tree. Weak link between contents and structure.	Powerful tree pattern matching language. Can distinguish order but not positions nor direct relationships. Can express equality between different parts of a structure, by using logical variables. Set manipulation via logical connectives.

Table 1: An analysis of similar models.

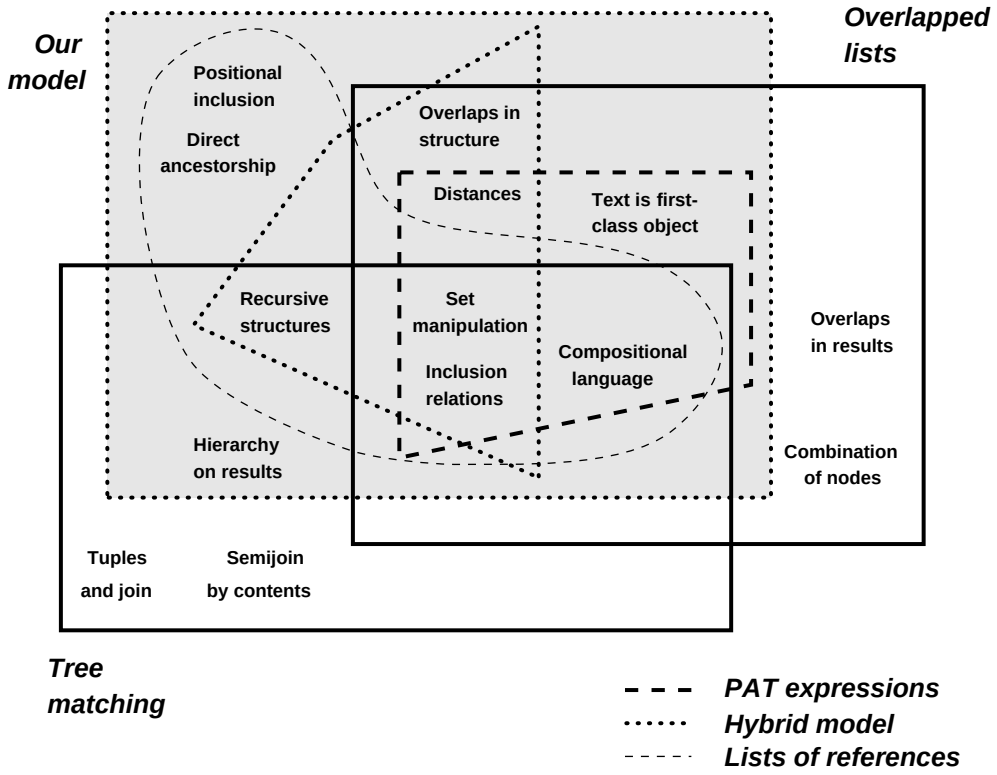

Figure 2: A graphical representation of the comparison made in the framework.

Regarding tuples, joins and semijoins, only the tree matching model can manage these features (and also *p-strings*, in its own context of a data manipulation language). These two languages do not have an efficient implementation. On the other hand, overlaps and combination of resulting nodes from a query are allowed by the overlapped lists model, but at the expense of not allowing them to form a hierarchy. We have not found an efficient implementation if we allow both features at the same time, and we consider that having hierarchies is more important in real cases.

Therefore, we have that our model has most of the features that are important in practice. Those which are not present appear to be not suitable of an efficient implementation.

6 Conclusions and Future Work

The problem of querying a textual database on both its contents and structure has been analyzed. We found the existing approaches to be either not expressive enough or inefficient. So, we propose a model based on a set-oriented query language that operates on nearby nodes and is efficiently implementable. We have compared the expressivity of our model with other related ones, and we have defined a framework in which to evaluate any similar model.

The conclusion of this comparison is that our model is competitive in expressivity with others that do not have such an efficient implementation, while the models with comparable

efficiency are less expressive than ours.

See Figure 3 for a graphical (and informal) comparison of similar models when taking into account both efficiency and expressivity. Note that we have included *p-strings* in this drawing, assuming an expressivity superior to all the languages we have analyzed. Note also that only a part of the lists-of-references model is considered. Finally, we observe that, as any quantization of concepts it is, up to a certain extent, subjective. Nevertheless, it does give an idea of where our model is.

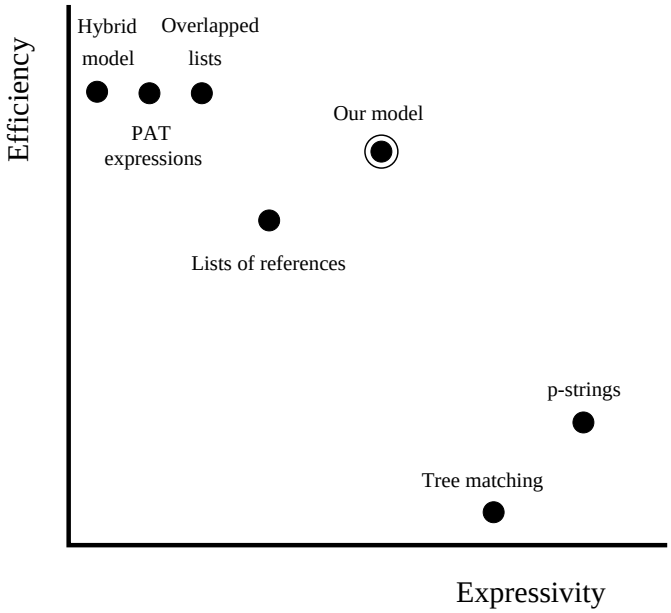

Figure 3: A comparison between similar models, regarding efficiency and expressivity.

A central issue that has appeared in this work is how to compare information models, even when they are quite similar. Although we have done one step forward in this direction, the problem is by no means solved. A formal framework in which to compare expressivity is needed. The long-term goal should be a formal and sound hierarchy like what can be found in the area of formal languages (see [6, 11] for some initial attempts). Many of the covered models are known to have expressivity limitations (e.g. semijoins).

References

[1] R. Baeza-Yates. An hybrid query model for full text retrieval systems. Technical Report DCC-1994-2, Dept. of Computer Science, Univ. of Chile, 1994.

[2] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proc. ACM SIGMOD'94*, pages 313–324, 1994.

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

[4] C. Clarke, G. Cormack, and F. Burkowski. Schema-independent retrieval from heterogeneous structured text. In *Procs. of the 4th Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, Apr. 1995.

[5] M. Consens and T. Milo. Optimizing queries on files. In *Proc. ACM SIGMOD'94*, pages 301–312, 1994.

[6] M. Consens and T. Milo. Algebras for querying text regions. In *Proc. PODS'95*, 1995. California.

[7] B. Desai, P. Goyal, and S. Sadri. A data model for use with formatted and textual data. *Journal of ASIS*, 37(3):158–165, 1986.

[8] H. Fawcett. *PAT 3.3 User's Guide*. UW Centre for the New OED and Text Research, Univ. of Waterloo, 1989.

[9] W. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1992.

[10] G. Gonnet and F. Tompa. Mind Your Grammar: a new approach to modelling text. In *Proc. VLDB'87*, pages 339–346, 1987.

[11] M. Gyssens, J. Paredaens, and D. V. Gucht. A grammar-based approach towards unifying hierarchical data models. In *Proc. ACM SIGMOD'89*, pages 263–272, 1989.

[12] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proc. ACM SIGIR'93*, pages 214–222, 1993.

[13] A. Loeffen. Text databases: A survey of text models and systems. *ACM SIGMOD Conference. ACM SIGMOD RECORD*, 23(1):97–106, Mar. 1994.

[14] I. MacLeod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3):254–264, 1991.

[15] G. Navarro. A language for queries on structure and contents of textual databases. Master's thesis, Dept. of Computer Science, Univ. of Chile, Apr. 1995.

[16] G. Navarro and R. Baeza-Yates. A language for queries on structure and contents of textual databases. In *Proc. ACM SIGIR'95*, July 1995. Seattle, WA.

[17] R. Sacks-Davis, T. Arnold-Moore, and J. Zobel. Database systems for structured documents. In *Proc. ADTI'94*, pages 272–283, 1994.

[18] A. Salminen and F. Tompa. PAT expressions: an algebra for text search. In *COMPLEX'92*, pages 309–332, 1992.

[19] M. Stonebraker, H. Stettner, N. Lynn, J. Kalash, and A. Guttman. Document processing in a relational database system. *ACM TOIS*, 1(2):143–158, Apr. 1983.