Querying Hierarchical Text and Acyclic Hypertext with Generalized Context-Free Grammars

Yves MARCOUX[*] Martin SÉVIGNY[†]

EBSI, Université de Montréal
C.P. 6128, Succ. Centre-ville, Montréal, Québec, Canada, H3C 3J7
E-mail: `marcoux@ere.umontreal.ca | msevigny@sympatico.ca`

Abstract

We present a formal data model and query language for hierarchically structured text and acyclic hypertext. The data model is related to SGML (Standard Generalized Markup Language, ISO 8879) and the retrieval mechanism is based on pattern-matching using context-free grammars (CFGs) with generalized regular expressions on the right-hand sides of productions. This novel approach is particularly well suited for dealing with recursive elements, and defines a very expressive query language exploiting both the structure and the contents of the text. The language includes complementation and intersection operations. We also discuss an algorithm for an efficient implementation of a search engine based on the model.

Keywords

SGML; Structured Documents; Context-Free Grammars; Information retrieval; Structured Text; Structured Queries; Hypertext

[*]Supported by a grant from the Social Sciences and Humanities Research Council of Canada.
[†]Supported by a grant from the National Sciences and Engineering Research Council of Canada.

1 Introduction

As the use of global networks increases, it is becoming every day more obvious that the development of search mechanisms for retrieving relevant material from the mass of available information is one of the prime challenges facing the IR community today. One important subproblem of this general goal is that of searching for relevant information within databases of structured text. Already today, a vast portion of the information available through global networks is in the form of structured text, be it HTML (HyperText Markup Language), SGML (Standard Generalized Markup Language, ISO 8879), XML (Extensible Markup Language), or HyTime (Hypermedia/Time-based Document Structuring Language, ISO 10744) [Gol91, DD94, BS97]. No solution to the general problem of network information retrieval can be quite satisfactory if it does not also address the issue of searching through structured text.

There are basically two research directions that can be taken for investigating IR in structured text: the general hypertext approach, in which the topology of the text can be an arbitrary graph; and the hierarchical text approach, in which the topology is restricted to be a tree. Although less general than the hypertext approach, the hierarchical approach is important because it is a natural and widely used way of structuring text, and because the restriction to trees allows the use of valuable search mechanisms (e.g., parsing) not applicable to unrestricted hypertext.

The retrieval model presented here differs from previously defined models because it applies not only to hierarchical text, but to acyclic hypertext in general. This feature is interesting because important acyclic subtopologies can often be identified within hypertext networks, for instance, the Web. The data model is closely related to the canonical form of SGML documents.

Another element of originality is the use of generalized context-free grammars (CFGs) for expressing queries. This approach gives rise to a very powerful retrieval language, one interesting feature of which is the ability to express very sophisticated criteria based on the nesting of elements. In particular, elaborate nesting criteria involving recursive elements (e.g., a section within a section or a list within a list) can be expressed. Recursive elements are permitted, and even encouraged, by structured text standards such as SGML (for example, see [TW95, p. 311]) and HyTime. Nevertheless, most retrieval models introduced so far either do not allow recursive elements, or have no powerful capabilities for expressing search criteria involving them.

One of the design objectives of the model was to preserve the simplicity and naturalness of common word-oriented retrieval operations, like adjacency, proximity, etc. Thus, we designed the model so that all these existing operations can be "imported" directly into it. Another objective is to attain a very high level of generality, while at the same time making sure that the retrieval mechanisms are efficiently implementable. Our goal is to delimit what it means to search through hierarchically structured text and acyclic hypertext in a general, but also practical, way.

About this paper

When an information retrieval model is introduced, it can be described and analyzed on at least two different levels: the mechanical aspects of the retrieval operations; and the amount of "semantic contents" that can be put into the data structures and retrieval mechanisms by judicious use.

Ultimately, the potential benefits of a model in real-life applications are related more to the second aspect than to the first one. However, from a research perspective, the purely mechanical aspects of an IR model are also of interest in themselves. We believe that separating the mechanical aspects of a model from its semantic ones (when possible), reveals more clearly the power of the underlying operations. The model we present here lends itself to such a separation, and we have chosen to concentrate in this paper on its mechanical aspects.

We could argue that the logical structure of text does convey semantic information, and that structure based retrieval is *per se* a form of semantic retrieval; however, we prefer saying that we are deliberately presenting the model from a purely mechanical point of view, even though we realize that the real interest of a model lies in its ability to capture the semantic nature of things (ability which could be investigated in future experimentation).

2 Related work

While they have tackled both the hypertext and the hierarchical text approaches, *implemented* solutions to searching in structured text have so far been rather modest. The most successful endeavour is probably the PAT software package by Open Text Corporation, which is widely used in various settings, among others, as a Web search engine. Other end-user packages for browsing and managing SGML documents (for example Explorer from SoftQuad or Dynatext from Electronic Book Technologies) allow users to formulate queries exploiting the structure of the underlying SGML documents, but these capabilities are fairly limited and usually don't go much beyond the use of SGML elements as index identifiers.

As far as research is concerned, a fair amount of work has been devoted to IR in both hierarchical text and hypertext. However, the matter is far from closed on either path. In the hierarchical line, a number of models have been introduced in the past, and an excellent survey (as well as an original model) can be found in [NB95].

An approach such as grammatical tree matching [KM92, KM93] uses grammars as a document model for structured text. Queries are represented by giving a tree pattern, which is then matched against the whole tree-like structure of the documents. The *parsed strings* model [GT87] also uses a grammar to represent structured text, and defines a manipulation language with operations on objects such as parsed strings or subtrees, along with operators for selecting and transforming part of the tree-structured text. According to [NB95], these two models are very expressive but lack efficient implementation. In [KS97], Kuikka and Salminen also use grammars as part of "templates", by which portions of structured text can be extracted. In [Bur92, Bur92a], Burkowski introduces an algebra for hierarchically structured text, based on the concepts of "contiguous extents" and "concordance lists."

Other approaches have been taken, like extending traditionnal command languages or SQL to handle structured documents [Mac90, Mac91]. In general, however, these approaches give rise to query languages that are less expressive than the aforementioned models.

The recently approved ISO standard "Document Style Semantics and Specification Language" (DSSSL) [ISO96] includes the "Structured Document Query Language" (SDQL), designed to manipulate and extract any part of SGML documents. SDQL adds retrieval primitives to the expression language of DSSSL, which is itself a variant of the Scheme programming language.

3 The data model

3.1 Basic definitions

Let N^+ denote the set of positive integers.

An alphabet is a non-empty finite set of characters (symbols). If A is an alphabet, the A^* denotes the set of finite strings over A, and A^+ denotes the set of non-empty finite strings over A. Character constants are written using double quotes, for example: `"a"`, `"string"`. String concatenation is denoted by a comma: `"<"`, `">"` = `"<>"`. The length of (i.e., the number of character occurrences in) a string w is denoted by $|w|$.

For graph-related definitions, we follow as closely as possible [CLR90].

A *finite ordered directed multi-graph* (FODMG) is a pair $G = (V, E)$, where V is a finite non-empty set of *vertices* (or *nodes*), and E is a finite partial function from $V \times N^+$ to V such that for all $v \in V$ and all $n > 1$, if $E(v, n)$ is defined, then $E(v, n-1)$ is also defined. The partial function E is called the *child-function* of G.

Iff $E(u, n) = v$ for some given n, we say that v is the n-th *child* of u, and that u is a *parent* of v; we also say that v is *adjacent* to u. Note that a vertex v can be both the n-th *and* the m-th child of the same vertex, with $m \neq n$. We define the out-degree of vertex v, denoted by $\delta(v)$, as the largest n for which $E(v, n)$ is defined, if such an n exists, and 0 otherwise. Note that $\delta(v)$ is the number of (not necessarily distinct) children of v.

A *path of length* $n - 1$ $(n \geq 1)$ *from* u to v is a sequence of n vertices $(v_1, \ldots, v_n)$ such that $v_1 = u$, $v_n = v$, and v_i is adjacent to v_{i-1} for each i satisfying $1 < i \leq n$. We say that v is *reachable* from u iff there exists a path from u to v. Note that there exists a length 0 path from any vertex to itself; thus, any vertex is reachable from itself.

We say that G is *acyclic* iff there exists no positive length path from any vertex to itself.

A *finite ordered directed multi-tree* (FODMT) is an acyclic FODMG $G = (V, E)$ in which there exists a vertex $r \in V$ such that all vertices in V are reachable from r by a (possibly empty) directed path. It is easy to show that there can only be one such vertex r; we call it the *root* of G. It is also easy to show that r is the only vertex that is not adjacent to any other vertex.

A *sequence of consecutive siblings* is either the sequence (r), where r is the root of G, or a (possibly empty) sequence of (not necessarily distinct) vertices $(v_1, \ldots, v_n)$, $(n \geq 0)$ such that there exist $v \in V$ and $k \geq 0$ for which $v_i = E(v, k + i)$ as soon as $1 \leq i \leq n$.

A *leaf* is a vertex without any child, i.e., with an out-degree of 0. There is always at least one leaf in G. The root r is a leaf iff it is the only vertex in V. A vertex that is not a leaf is called an *internal node*.

3.2 Document-bases

We define a *document-base* D as follows.

The basic component of D is a FODMT (finite ordered directed multi-tree; see above) $G = (V, E)$ comprising at least two vertices. The root of G is denoted by r. The "topology" of the document-base is determined by G. We refer to the vertices in V as the vertices of the document-base.

The document-base D also comprises a *markup* alphabet M, and a *text* alphabet T. Without loss of generality, we assume that none of the characters `"<"`, `">"`, `"/"`, `"("`, `")"`, `"["`, `"]"`, nor the double quote character `"`, is in $M \cup T$.

Finally, D comprises a total *contents-function* $\kappa : V \rightarrow (M^+ \cup T^*)$ such that, for all v, if v is a leaf, then $\kappa(v) \in T^*$, and if v is an internal node, then $\kappa(v) \in M^+$.

A string of *generated-text* is associated with each vertex of D by way of a total *generated-text function* $\gamma : V \rightarrow (\{\texttt{"<"}, \texttt{">"}, \texttt{"/"}\} \cup M \cup T)^*$, which is recursively defined as follows:

$$\begin{aligned} \gamma(v) &\overset{\mathrm{d}}{=} \kappa(v) \quad \text{if } v \text{ is a leaf,} \\ \gamma(v) &\overset{\mathrm{d}}{=} \texttt{"<"}, \kappa(v), \texttt{">"}, \gamma(E(v,1)), \ldots, \gamma(E(v,n)), \texttt{"</>"} \\ &\qquad\quad \text{where } n = \delta(v)\text{, otherwise.} \end{aligned}$$

The function γ is easily generalized to sequences of vertices as follows:

$$\begin{aligned} \gamma(\sigma) &\overset{\mathrm{d}}{=} \gamma(v_1), \ldots, \gamma(v_n) && \text{where } \sigma = (v_1, \ldots, v_n),\ n \geq 1, \\ \gamma(\sigma) &\overset{\mathrm{d}}{=} \texttt{""} && \text{if } \sigma \text{ is the empty sequence.} \end{aligned}$$

In the remainder of this paper, we suppose D is a document-base with FODMT $G = (V, E)$, root r, associated alphabets M and T, and functions κ and γ, as defined here.

3.3 Comments on the data model

Here are some intuitive comments on our data model.

The document-base is the body of data against which search queries are formulated. Queries will return results that are sequences of vertices from V. The FODMT (together with function κ) can be thought of as an abstract view of the document-base, whereas $\gamma(r)$ is its actual expanded text, seen as a sequential character string. The FODMT is much richer than $\gamma(r)$, because it allows the representation of text-segment reuse and of acyclic hypertextual structures (by the fact that nodes can have multiple parents).

We define a document-base as a FODMT, instead of simply as the character string $\gamma(r)$, for many reasons. The most important one is to allow the definition of query results as sequences of nodes. They can thus be browsed by hypertextual navigation and linked to the actual contents of the document-base. Another reason is to allow for extensions of the query language that include node identity criteria.

The data model is obviously inspired by SGML and HTML, in particular, the text $\gamma(r)$ of the document-base includes start-tags (strings of the form `<TITLE>`, `<PARA>`, ...) and end-tags (`</>`) that delimit segments of text.

The markup alphabet comprises those characters that can be used to form generic IDs, i.e., identifiers found within start-tags. The text alphabet comprises those characters that can occur between tags, i.e., in the contents of leaves. Note that all start-tags are given matching end-tags by γ.

The text $\gamma(r)$ of any document-base always corresponds to some valid SGML document-instance. However, there are important differences between SGML and our data model, among which we point out the following (this list can be skipped by readers unfamiliar with SGML):

- The text $\gamma(r)$ of a document-base is not restricted to conform to any particular DTD (Document Type Definition) and includes no SGML declaration or document type declaration (it is thus closer to an XML well-formed document than to an SGML valid document [BS97]).
- An SGML declaration specifying `SHORTTAG=YES` is always implied, to allow the use of empty end-tags `</>`.
- In the FODMT, there can be sequences of two or more consecutive sibling leaves. This is as though consecutive `#PCDATA` segments were allowed within an element. However, as far as the text $\gamma(r)$ of the document-base is concerned (and as far as retrieval operations are concerned), such sequences of consecutive sibling leaves are indistinguishable from single `#PCDATA` segments. We permit them mainly to allow for extensions of the query language in which the markup associated with specific nodes can be ignored.
- There is no notion of attribute. A possible way to treat attributes is to consider them as sub-elements. The capabilities of the query language (in particular, the presence of the intersection operation) render such a treatment fairly easy.
- SGML's empty elements can only be represented by internal nodes having a single child, which is a leaf whose contents is the empty string. The generated text of such vertices is of the form `<A></>`, rather than the correct SGML `<A>` (with no end-tag).
- We require the document-base to have at least two vertices mainly because, with only one vertex, the generated-text would have no start- or end-tag, and thus would not be a valid SGML document-instance.

3.4 Segments

A *segment* of D is a pair of integers (x, y) such that $1 \leq x \leq y \leq |\gamma(r)| + 1$. A segment (x, y) is said to be *empty* iff $x = y$. A segment (x, y) *includes* another segment (x', y') iff $x \leq x'$ and $y' \leq y$. Then, (x', y') is said to be a *subsegment* of (x, y). We say that (x, y) *starts with* (x', y') iff $x = x'$. We say that (x, y) *ends with* (x', y') iff $y = y'$.

Let (x, y) be a segment. The *text* of (x, y), denoted $\tau(x, y)$ is defined to be the empty string iff $x = y$, and otherwise, the portion of $\gamma(r)$ ranging from character positions x to $y - 1$ if $x < y$ (positions start at 1). We now define several types of segments based on the properties of x, y and $\tau(x, y)$. A *start-tag* is a string of the form `<`m`>`, where $m \in M^+$, and an *end-tag* is the string `</>`.

Genid segments Iff (x, y) is a segment such that $\tau(x, y)$ is a start-tag, then $(x + 1, y - 1)$ is said to be a *genid* segment.

Well-formed segments We say that (x, y) is a *well-formed* segment (WFS) iff all of the following statements hold:

1. $\tau(x, y)$ has an equal number of start-tag and end-tag occurrences;
2. no prefix of $\tau(x, y)$ has more end-tag occurrences than start-tag occurrences;
3. either $\tau(x, y)$ is non-empty and starts with a start-tag, or $x > 1$ and character position $(x - 1)$ of $\gamma(r)$ is a `">"`;
4. either $\tau(x, y)$ is non-empty and ends with an end-tag, or $y \leq |\gamma(r)|$ and character position y of $\gamma(r)$ is a `"<"`.

Single-element WFSs If (x, y) is a WFS and, in addition, $\tau(x, y)$ starts with a start-tag and ends with the matching end-tag (i.e., no proper prefix of $\tau(x, y)$ has an equal positive number of start- and end-tags), then we say that (x, y) is a *single-element* WFS.

Text-only WFSs If (x, y) is a WFS and, in addition, $\tau(x, y)$ contains no occurrence of `<`, `>` or `/`, then (x, y) is said to be a *text-only* WFS.

Compound WFSs If (x, y) is a WFS but is neither a single-element WFS nor a text-only WFS, then it is said to be a *compound* WFS.

Intuitively, a WFS delimits a well-balanced and complete portion of $\gamma(r)$. Note in particular that a WFS cannot start or end in the middle of a sequence of characters located between tags; it must totally include or totally exclude any such sequence. Also, a single tag (or part of a tag) does not constitute a WFS. An empty WFS is necessarily a text-only WFS, and necessarily falls between two adjacent tags.

3.5 Mapping WFSs to sequences of nodes

The retrieval operations of our model will be defined in terms of pattern-matching on the text $\gamma(r)$ of the document-base. However, the results of queries will be defined as sequences of vertices from V. Thus, we need to be able to map segments of $\gamma(r)$ back to the vertices of V that "generate" them. This mapping is very natural, but its formal definition is rather complex, mainly because the text generated by a vertex is "used" more than once to form $\gamma(r)$, if that vertex has more than one ancestral line.

We first define recursively the function ℓ, which associates with each vertex $v \in V$ the set of all locations where v's generated text is "used" in the construction of $\gamma(r)$.

Algorithm for $\ell : V \to 2^{N^+}$
Input: $v \in V$

```
IF $v = r$ THEN RETURN $\{1\}$ END IF
$X \leftarrow \emptyset$
FOR EACH $(u, n)$ such that $E(u, n) = v$
  FOR EACH $x \in \ell(u)$
    $X \leftarrow X \cup \{x + |\kappa(u)| + 2 + |\gamma(E(u, 1), \ldots, E(u, n-1))|\}$
  END FOR
END FOR
RETURN $X$
```

Note that the $|\kappa(u)| + 2$ term in the above definition is to allow for the length of the start-tag of the parent u of v. Also note that, when $n = 1$, $|\gamma(E(u, 1), \ldots, E(u, n-1))| = 0$.

Finally, we define the function ν that maps any WFS to the node or sequence of nodes in V that generates it. The idea is simple, but care must be taken to properly treat leaves with empty contents.

Let (x, y) be a WFS. If (x, y) is a single-element WFS, then there is a unique $v_0 \in V$ such that $x \in \ell(v_0)$ and $y = x + |\gamma(v_0)|$, and $\nu(x, y)$ is defined as the sequence (v_0). Otherwise, for each $v \in V$, define z_v as the unique $z \in \ell(v)$ such that $z < x$ and $z + |\gamma(v)| > y$ if such a z exists, and 0 otherwise. Note that z_r is always equal to 1. Then, let $u \in V$ be such that $z_u = \max_{v \in V}(z_v)$. There is always a unique such u and it is always an internal node. Intuitively, u is the vertex of D that most tightly covers (x, y). Now, let n_0 be the smallest $n \in N^+$ for which $z_u + |\kappa(u)| + 2 + |\gamma(E(u, 1), \ldots, E(u, n-1))| \geq x$, and let n_1 be the largest $n \leq \delta(u)$ for which $z_u + |\kappa(u)| + 2 + |\gamma(E(u, 1), \ldots, E(u, n))| \leq y$. Then, $\nu(x, y)$ is defined as the sequence $(E(u, n_0), \ldots, E(u, n_1))$ (if $n_1 < n_0$, then $\nu(x, y)$ is the empty sequence).

Note that empty WFS are always mapped either to the empty sequence or to a sequence of empty sibling leaves.

4 Retrieval operations

4.1 Intuitive overview

The retrieval mechanism of the model is based on pattern-matching using Context-Free Grammars (CFGs) with generalized regular expressions on the right-hand sides of productions. The generalized regular expressions ("expressions", for short) are of two disjoint types: *genid*-expressions and *ordinary*-expressions. Expressions can "match" certain segments of D.

One important class of ordinary-expressions are *word-expressions*. They are of the form `[word-expression]`. Without loss of generality, we suppose that word-expressions contain no `[` or `]` characters. Word-expressions are written in a language independent of the rest of the model. This language could be based, for instance, on word-oriented boolean operations, adjacency, proximity, etc. Word-expressions can only match text-only WFSs. Moreover, they are the only expressions that can match non-empty text-only WFSs. For example, `[library]` might be a word-expression matching any text-only WFS containing the word "library".

A genid-expression is used to match collectively a number of generic IDs. For instance, if a genid-expression represents all of M^+, then it would match all occurrences of all generic IDs. If it represents a single generic ID, then it would match only occurrences of this specific generic ID. For example, `TITLE` might be a genid-expression matching all occurrences of the generic ID `TITLE`.

One way to obtain an ordinary-expression is to include a genid-expression α in a construction of the form `(<`α`>`β`</>)`, where β is another ordinary-expression. The ordinary-expression `(<`α`>`β`</>)` will match any single-element WFS that starts with a start-tag containing a generic ID matched by α, and that has a contents matching β. For example, `(<TITLE>[library]</>)` could be an ordinary-expression matching any single-element WFS with generic ID `TITLE` and having a contents composed of one text-only WFS containing the word "library".

One or more ordinary-expressions can be combined with various connectors to yield other ordinary-expressions. The set of strings represented by the resulting expression is defined in terms of the set(s) of strings represented by the original expression(s). For example, `(`α `|` β`)` will represent the union of the sets represented by α and β, and `(`α`*)` will represent the Kleene closure of the set represented by α.

A *query* Q on D is a *generalized* CFG, i.e., a CFG in which the right-hand sides of the productions are ordinary-expressions. Each non-terminal of Q is also considered an ordinary-expression. In order for a query to return anything, Q's start-symbol must match the segment $(1, |\gamma(r)| + 1)$, i.e., the whole document base. What is returned as *results* of the query, however, depends on how it is formulated. Indeed, we allow some ordinary-subexpressions to be *underlined* in the query, and which subexpressions are underlined is what determines the results returned. Roughly speaking, if a segment (x, y) is matched by an underlined subexpression of the query, then the sequence $\nu(x, y)$ will be returned as a result. However, as we shall see formally below, the matching must occur in the context of a global matching of the document-base by Q's start-symbol.

4.2 Generalized CFGs

A *query* Q on D is a *generalized CFG*, which we now define. We assume the reader is familiar with CFGs (see, for example, [HU79]).

The set of terminals of Q is $\{$`"<"`, `">"`, `"/"`$\} \cup M \cup T$. Q has a finite set of non-terminals, denoted $\mathbf{N}$. A distinguished non-terminal $S \in \mathbf{N}$ serves as the start-symbol of Q. For each non-terminal $A \in \mathbf{N}$, there is exactly one production in Q with A as a left-hand side. We call it the A-production. The right-hand side of the A-production is an *ordinary-expression*, as defined hereafter. We must first define *genid-expressions*.

4.2.1 Genid-expressions

Due to space limitations, we introduce here a very simple form for genid-expressions; however, they can be defined with as much pattern-matching capability as ordinary-expressions (see [MS97]). We define genid-expressions (and their semantics) by the following rules:

1. For all $x \in M^+$, x and $\neg x$ are genid-expressions; they represent respectively the sets $\{x\}$ and $M^+ - \{x\}$.
2. *any* is a genid-expression; it represents the set M^+.
3. Nothing else is a genid-expression.

4.2.2 Ordinary-expressions

Ordinary-expressions are defined by the following rules:

1. Any word-expression is an ordinary-expression.
2. For each $A \in \mathbf{N}$, A is an ordinary-expression.
3. If α is a genid-expression and β is an ordinary-expression, then `(<`α`>`β `</>)` is an ordinary-expression.
4. If α and β are ordinary-expressions, then (α, β), (α | β), (α+), (α*), (α?), ($\alpha \cap \beta$), and ($\neg\alpha$) are also ordinary-expressions.
5. Nothing else is an ordinary-expression.

We immediately point out that, because of the presence of the $\cap$ (intersection) and $\neg$ (complementation) connectors, our grammars can generate languages that are *not* context-free (context-free languages are not in general closed under intersection or complementation). However, as we shall see, the presence of these connectors does not change the basic parsing algorithm, nor its complexity.

Without loss of generality, we assume that the set of non-terminals is disjoint from the set of terminals and from the set of connecting symbols used to form ordinary-expressions.

4.2.3 Syntactic restriction to avoid inconsistency

The non-terminals of Q are further subject to the following special condition, to ensure that the use of negations ($\neg$) does not later lead to inconsistencies. Consider the directed graph which has $\mathbf{N}$ as set of vertices and in which (A, B) is an edge iff B appears anywhere in the A-production. We say that (A, B) is a *special* edge iff B appears in any negated subexpression of the A-production. Then, no cycle of that graph is allowed to contain any special edge. Thus, for example, obviously inconsistent productions of the form $A \to (\neg A)$ are forbidden.

4.2.4 Segments matching an ordinary-expression

Certain segments of D can *match* certain ordinary-expressions, according to the following rules. Let η be a genid- or ordinary-expression.

1. If η is a word-expression, then it is matched by any WFS (x, y) such that $\tau(x, y)$ is in the set of strings represented by η. The semantics of word-expressions is not included in the present model. For the sake of examples, we assume that an expression of the form `[word]` represents the set of all strings from T^* that contain the word `word`, and that `[$]` represents all of T^*.

   Note that only text-only WFSs can match a word-expression.

2. If η is a non-terminal, and a WFS (x, y) matches the right-hand side of the η-production, then (x, y) also matches η.

3. If η is a genid-expression, then it is matched by any genid segment (x, y) such that $\tau(x, y)$ is in the set of strings represented by η.

4. If η is of the form `(<`α`>`β`</>)`, then it is matched by any WFS (x, y) such that $\tau(x, y)$ is of the form `<`a`>`b`</>`, where a matches α and b matches β.

   Note that only single-element WFSs can match an expression of the form `(<`α`>`β`</>)`.

5. If η is of the form `(`α`,` β`)`, then it is matched by any WFS (x, y) such that there exists a k, satisfying $0 \leq k \leq (y - x)$, for which $(x, x + k)$ matches α and $(x + k, y)$ matches β.

6. If η is of the form `(`α`*)`, then it is matched by any WFS (x, y) that either is empty, or is such that there exists a k, satisfying $0 \leq k \leq (y - x)$, for which $(x, x + k)$ matches α and $(x + k, y)$ matches `(`α`*)`.

7. If η is of the form `(`α`+)`, then it is matched by any WFS (x, y) such that there exists a k, satisfying $0 \leq k \leq (y - x)$, for which $(x, x + k)$ matches α and $(x + k, y)$ matches `(`α`*)`.

8. If η is of the form `(`α `|` β`)`, then it is matched by any WFS that matches α or β.

9. If η is of the form `(`α`?)`, then it is matched by any WFS that either is empty, or matches α.

10. If η is of the form `(`α $\cap$ β`)`, then it is matched by any WFS that matches both α and β.

11. If η is of the form `(`$\neg\alpha$`)`, then it is matched by any WFS that does not match α.

Note that, because of the special condition introduced in § 4.2.3, the last rule does not lead to inconsistency.

4.3 Results of a query

We can now establish for any given query Q on D, whether Q is matched by the segment $(1, |\gamma(r)| + 1)$, and thus, whether Q can return any results or not. However, we still need to define *what* the results will be, in the case Q does return something.

First, we allow certain ordinary-expressions in Q to be underlined. The underlining is done in such a manner that an expression can be underlined independently from its subexpressions.

Syntactically, this could be realized by underlining the opening parenthesis (or bracket [of the expressions.

Second, we define a scheme by which the results of a query are defined, based on the underlining of expressions in Q. Due to space limitations, we define here a very simple scheme; in [MS97], we present a much more powerful one.

The idea of the simple scheme is to associate certain ordinary-expressions in Q to certain WFSs of D according to some rules. The associations are done in a top-down manner, and in a such a way as to take into account all possible matchings of Q by D. At the end of the association process, iff a segment s is associated with any underlined expression, then $\nu(s)$ will be a result of Q. Thus, the overall result of a query is a *set of sequences of vertices from* V.

Here are the rules by which we associate ordinary-expressions found in Q to WFSs of D. Note that zero, one, or more than one expression can be associated with any given WFS, and that only (but not necessarily all) expressions matching a WFS will ever be associated with it. Let η be any ordinary-expression occurrence found in Q (η can be underlined or not, and can contain underlined subexpressions or not).

1. If S (the start symbol of Q) is matched by $(1, |\gamma(r)|+1)$, then we associate S to $(1, |\gamma(r)|+1)$.

2. If η is a non-terminal and is associated with a WFS, then we also associate the right-hand side of the η-production to the same WFS.

3. If η is of the form (<α>β</>) and is associated with WFS (x, y), then we associate β with $(x+k, y-3)$, where k is the length of the initial start-tag of $\tau(x, y)$.

4. If η is of the form (α, β) and is associated with a WFS (x, y) then, for all k such that $1 < k < (y - x)$, if α matches the WFS $(x, x+k)$ and β matches the WFS $(x+k, y)$, then we associate α with $(x, x+k)$ and β with $(x+k, y)$.

5. If η is of the form (α*) or (α+), and is associated with a WFS (x, y), then for all k such that $1 < k < (y-x)$, if α matches the WFS $(x, x+k)$ and (α*) matches the WFS $(x+k, y)$, then we associate α to $(x, x+k)$ and (α*) to $(x+k, y)$.

6. If η is of the form (α?) and is associated to a non-empty WFS, then we associate α to the same WFS.

7. If η is of the form (α | β) and is associated to a WFS (x, y), then we associate α with (x, y) iff α matches (x, y), and β with (x, y) iff β matches (x, y).

8. If η is of the form ($\alpha \cap \beta$) and is associated to a WFS (x, y), then we associate both α and β to (x, y).

It can be shown that this association process always terminates.

5 Comments and examples

The retrieval language is inspired by SGML DTDs, however, there are important differences. Among others, regular expressions can be ambiguous in our model, but not in SGML. Also, complementation and intersection are absent from SGML. Finally, the fact that productions do not necessarily generate markup gives our model the full expressiveness of CFGs, and in fact, more (because of intersection and complementation).

In all examples, we omit parentheses whenever possible and convenient. We also assume that ω is an ordinary non-terminal defined by:

$$\omega \rightarrow \texttt{(}\omega\texttt{*)} \;\texttt{|}\; \texttt{[\$]} \;\texttt{|}\; \texttt{(<}\mathit{any}\texttt{>}\omega\texttt{</>)}$$

where `[$]` is the word-expression matching all of T^*. The non-terminal ω will thus match any WFS. We also assume, except where otherwise indicated, that S (the start-symbol) is defined as:

$$S \rightarrow \texttt{(}\omega\texttt{,}\ R\texttt{,}\ \omega\texttt{)} \;\texttt{|}\; \texttt{(<}\mathit{any}\texttt{>}S\texttt{</>)}$$

where R is a non-terminal that will vary with each example. R will thus, in effect, be searched for in the whole document-base. As far as underlining is concerned, we shall only underline expressions of the form $\texttt{(<}\alpha\texttt{>}\beta\texttt{</>)}$, and we shall do it by underlining the opening $\underline{\texttt{<}}$.

Example 1 Titles of sections that *follow* any section having the word "painting" in its title.

$$R \rightarrow \texttt{(<SEC>(<TI>[painting]</>)}\omega\texttt{</>)<SEC>(}\underline{\texttt{<}}\texttt{TI>[\$]</>)}\omega\texttt{</>}$$

Note the underlined $\underline{\texttt{<}}$.

Example 2 Bibliographies of chapters containing two or more paragraphs containing the word "painting."

$$R \rightarrow \texttt{<CHAP>}\omega\texttt{,}\ P\texttt{,}\ \omega\texttt{,}\ P\texttt{,}\ \omega\texttt{,}\ \texttt{(}\underline{\texttt{<}}\texttt{BIB>}\omega\texttt{</>)</>}$$

$$P \rightarrow \texttt{<PAR>[painting]</>}$$

We now illustrate the use of negation by an example with two solutions.

Example 3 Sections that do not immediately contain a title.

1)

$$R \rightarrow \texttt{(}\underline{\texttt{<}}\texttt{SEC>}\omega\texttt{</>)} \cap \neg\texttt{(<}\mathit{any}\texttt{>}\omega\texttt{,}\ \texttt{(<TI>}\omega\texttt{</>),}\ \omega\texttt{</>)}$$

2)

$$R \rightarrow \underline{\texttt{<}}\texttt{SEC>(<}\neg\texttt{TI>}\omega\texttt{</>} \;\texttt{|}\; \texttt{[\$])*</>}$$

The next two examples illustrate the use of recursive non-terminals.

Example 4 Elements immediately containing the word `alarm`, except if located anywhere within a `WARNING`.

$$S \rightarrow \underline{\texttt{<}}\neg\texttt{WARN>[alarm]</>}\ \texttt{|}\ \texttt{<}\neg\texttt{WARN>}\ \omega\texttt{,}\ S\texttt{,}\ \omega\ \texttt{</>}$$

Example 5 Titles located at an even-numbered level of depth, counting from the root of the document base.

$$S \rightarrow \texttt{<}\mathit{any}\texttt{>}\ \omega\texttt{,}\ \texttt{(}\underline{\texttt{<}}\texttt{TI>}\omega\texttt{</>)}\texttt{,}\ \omega\ \texttt{</>}\ \texttt{|}\ \texttt{<}\mathit{any}\texttt{>}\ \omega\texttt{,}\ \texttt{(<}\mathit{any}\texttt{>}\ \omega\texttt{,}\ S\texttt{,}\ \omega\ \texttt{</>)}\texttt{,}\ \omega\ \texttt{</>}$$

We conclude with an example, inspired by nested lists in HTML, which combines recursive non-terminals and negation.

Example 6 Lists (`<UL>` elements, for unordered list) that do not contain any nested lists anywhere in them.

$$R \rightarrow \texttt{(}\underline{\texttt{<}}\texttt{UL>}\neg BUL\texttt{</>)}$$

$$BUL \rightarrow \omega\texttt{,}\ \texttt{(<UL>}\omega\texttt{</>}\ \texttt{|}\ \texttt{<}\mathit{any}\texttt{>}BUL\texttt{</>)}\texttt{,}\ \omega$$

Here, the non-terminal *BUL* is to be interpreted as "any WFS containing a buried unordered list."

6 Conclusion

6.1 Implementation issues

Parsing algorithms for generalized CFGs do not seem to have been studied extensively in the literature. A basic dynamic programming polynomial-time parsing algorithm can be obtained by combining the classical Cocke-Younger-Kasami algorithm for CFGs with an algorithm by Aho, Hopcroft and Ullman for generalized regular expressions [HU79, pp. 75, 139]. This basic algorithm can be modified to handle underlining, and still remain polynomial-time. If the query is taken to be part of the input, then the algorithm remains polynomial-time.

While polynomial-time, the algorithm is not likely to ever be implemented in full (at least on sequential processors), because it is not linear-time. However, we believe that interesting subsets of the retrieval language can be implemented using efficient algorithms that do not need to scan the whole document-base. We are currently developing a prototype of such a system, which supports intersection and a restricted form of complementation.

6.2 Current developments

When word-expressions can include operations like adjacency or proximity, it might be useful to ignore some markup in the document-base. For instance, one would not want the stream of words to be broken by non-structural markup, like HTML's `<B>` tags, used to display a passage in boldface. We are currently working out the details of an addition to the model that will allow some markup to be ignored, in a very flexible way. The addition will also allow whole segments (not just markup) to be ignored for searching purposes.

We are also developing a mechanism by which the results of a search can be reused in subsequent searches.

6.3 Future work

Future research avenues include pursuing the investigation of interesting and efficiently implementable subsets of the model. We also believe that important improvements could be achieved by developing query optimization techniques.

An interesting area would be to see how variables (in the manner of [KM93]) could be introduced in the model to allow further criteria based on equality, identity, or arbitrary relations among the sequence of nodes that form a result of a query.

Massive distributed computing (for instance, cooperating Web hosts) might constitue a promising avenue for the actual implementation of the full retrieval language of our model. Indeed, efficient parallel algorithms for parsing CFGs (based on boolean circuits or systolic arrays) have been known for quite some time. It would be interesting to see if and how these algorithms can be modified to support our model.

Work needs to be done to find out what kind of interfaces are needed to pass-on to the end-users the expressive power of the model. Already, there are some results in this area [SM97].

Finally, the true usefulness of the model would need to be evaluated by experimentation in real-life situations.

Acknowledgement

The authors wish to thank Novell Corporation for providing the SGML sources used for experimentation in this project.

This work has been supported by grants from the Social Sciences and Humanities Research Council of Canada and from the National Sciences and Engineering Research Council of Canada.

References

[BS97] Bray, T.; Sperberg-McQueen, C.M. Extensible Markup Language *Proceedings of SGML '96*, pp. 399-404.

[Bur92] Burkowski, F. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text. *Proceedings of SIGIR '92*, pp. 112-125.

[Bur92a] Burkowski, F. An Algebra for Hierarchically Organized Text-Dominated Databases. *Information Processing & Management*, Vol. 28, No. 3, pp. 333-348, 1992.

[CLR90] Cormen, T.; Leiserson, C.; Rivest, R. *Introduction to Algorithms.* New York: McGraw-Hill, 1990.

[DD94] DeRose, S.; Durand, D. *Making Hypermedia Work. A User's Guide to HyTime.* Boston: Kluwer Academic Publishers, 1994.

[Gol91] Goldfarb, C. *The SGML Handbook.* New York: Oxford University Press, 1991.

[GT87] Gonnet, G.; Tompa, F. Mind Your Grammar: a New Approach to Modelling Text. *Proceedings of the 13th VLDB Conference*, 1987, pp. 339-346.

[HU79] Hopcroft, J.; Ullman, J. *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley, 1979.

[ISO96] International Organization for Standardization. *Information Technology - Processing Languages - Document Style Semantics and Specification Language (DSSSL).* ISO/IEC 10179:1996.

[KM92] Kilpeläinen, P.; Mannila, H. Grammatical Tree Matching. *Proceedings of Combinatorial Pattern Matching '92*, pp. 162-174.

[KM93] Kilpeläinen, P.; Mannila, H. Retrieval from Hierarchical Texts by Partial Patterns. *Proceedings of SIGIR '93*, pp. 214-222.

[KS97] Kuikka, E.; Salminen, A. Two-dimensional filters for structured text. *Information Processing & Management*, Vol. 33, No. 1, pp. 37-54, 1997.

[Mac90] Macleod, I. Extending the Command Language Interface to Handle Marked-up Documents. *Proceedings of the American Society for Information Science Annual Meeting 1990*, pp. 192-196.

[Mac91] Macleod, I. A Query Language for Retrieving Information from Hierarchic Text Structures. *The Computer Journal*, Vol. 34, No. 3, pp. 254-264, 1991.

[MS97] Marcoux, Y.; Sévigny, M. *Querying Hierarchical Text and Acyclic Hypertext with Generalized Context-Free Grammars*, Technical Report, EBSI, Université de Montréal, in preparation.

[NB95] Navarro, G.; Baeza-Yates, R. A Language for Queries on Structure and Contents of Textual Databases. *Proceedings of SIGIR '95*, pp. 93-101.

[SM97] Sévigny, M.; Marcoux, Y. Conception et réalisation d'une interface-utilisateurs pour l'interrogation de bases de documents structurés. *Canadian Journal of Information and Library Science*, Vol. 21, No. 3/4, pp. 59-77, 1996.

[TW95] Travis, B.; Waldt, D. *The SGML Implementation Guide: A Blueprint for SGML Migration.* New York: Springer-Verlag, 1995.