Using sgrep for querying structured text files[1]

Jani Jaakkola
Pekka Kilpeläinen
University of Helsinki
Department of Computer Science
P.O. Box 26
FIN-00014 UNIVERSITY OF HELSINKI
Finland
{jjaakkol,kilpelai}@cs.helsinki.fi

Abstract

Sgrep is a Unix tool for searching the contents of text files. Sgrep implements an algebra of unrestricted text fragments called regions. The algebra allows the retrieval of document components, represented as regions, based on conditions on their relative containment and ordering. This simple yet powerful model is suitable for querying structured document formats like electronic mail, RTF, LaTeX, HTML, or SGML documents. We describe the sgrep query language and give examples of its use. Especially, we explain how sgrep can be used for querying and assembling SGML documents.

Keywords: Text search tools, structured documents, SGML.

1 Introduction

Every-day data processing creates large amounts of *structured documents* that have a more or less regular structure. Familiar examples of such structured documents include electronic mail, Usenet news, program source code, HTML pages, and SGML documents. Users seldom bother to index or to store such volatile texts in advanced information retrieval systems, even that would support the eventual searching of the documents based on their content and their structure. Therefore general purpose text search tools, such as `grep` [ThR75], `Awk` [AKW88] and `Perl` [WaS91], are often used to search structured text documents.

The drawback with ordinary text tools is that they do not support any high-level structures in the documents - they treat text only in units of rows, words and characters. Utilizing further hierarchical structure is either impossible or requires considerable programming. There is obviously a need for simple tools that would make the structure of text available for querying and manipulating. *Sgrep* (structured `grep`) [JK96]

1. Presented at SGML Finland 1996, Espoo, Finland, October 4-5, 1996.

offers such a tool by implementing a query algebra for retrieving hierarchical text structures.

The main power of sgrep is to treat text as unrestricted fragments called *regions*, which are natural representations for components of, e.g., SGML documents. Sgrep retrieves these regions from the text based on queries expressing relative containment and ordering conditions. The regions may nest and overlap arbitrarily, and neither their number nor their length is restricted. Sgrep does not assume anything about the structure or content of the files it searches. Because there are no "syntactically incorrect" documents for sgrep, it can be applied in a wide variety of tasks. Sgrep can be used to search either C source programs, SGML documents or binary files. On the other hand, the generality of sgrep also means that the user has to write queries carefully to match the markup in the files.

The query language of sgrep is based on a region algebra which was first introduced by Burkowski [Bur92] and later developed further by Clarke, Cormack and Burkowski [CCB95]. The region algebra consists of operations similar to query languages in some existing text retrieval systems such as PAT [SaT92]. Sgrep differs from its predecessors in its ability to treat arbitrarily nesting text regions, which is necessary for manipulating nested document elements like lists within lists.

The rest of the paper is organized as follows. First we describe briefly the basic notions of how sgrep models text in Section 2. Then we describe the query language of sgrep using simple examples in Section 3. Section 4 describes how sgrep is used for manipulating SGML documents. Section 5 is a short conclusion.

2 Data model of sgrep

In this section we describe the data model of sgrep, i.e., the basic structures that sgrep provides for manipulating text. The basic unit of data in sgrep is a region. A *region* is a non-empty fragment of text identified by its start and end positions. A region can be anything from a single character to the whole contents of a file. Sgrep operates on sets of regions called *region sets*. A region set can contain any regions, but each region can occur at most once. Often the regions in a set share some common property. For example, they could be document elements of the same type.

Region sets can be written in sgrep queries explicitly by enclosing in brackets [ ] a sequence of constant regions denoted by pairs of indices (*start*, *end*). For an example, assume that we have a file named `file10` consisting of ten characters:

% cat file10

0123456789

Now the region (0,9) equals to the contents of the entire file. The regions (0,0) and (9,9) comprise the first and the last character of the file, correspondingly. There are also built-in expressions `start` and `end` for region sets consisting of the first and the last character, correspondingly. Below is an example of evaluating a constant region set. The `-o` option allows an output format for regions to be specified. In this case the contents of each region is output between two hyphens.

% sgrep -o"-%r-" '[(0,5)(2,9)(3,8)(4,7)(4,9)]' file10

-012345--23456789--345678--4567--456789-

Regions expressed explicitly by indices, as above, are rarely used. Instead, regions retrieved as occurrences of a text pattern are the normal starting point for an sgrep query, as we see in the next section.

The manipulation of region sets described in the next section is based on conditions on the relative containment and ordering of regions. Let *a = (a.s, a.e)* and *b = (b.s, b.e)* be two regions. If *b* occurs properly within *a*, that is, if *a.s* < *b.s* <= *b.e* <= *a.e* or *a.s* <= *b.s* <= *b.e* < *a.e*, we say that region *a contains* region *b*, that region *b is included in* region *a*, and that the regions are *nested*. If region *b* starts after the end of region *a*, region *a precedes* region *b*, region *b follows* region *a*, and the regions are *disjoint*. If two regions are neither nested or disjoint, they are *overlapping*.

3 Sgrep query language

In this section we briefly describe the most central sgrep query language features. Sgrep implements a fully compositional algebra of region sets, which means that the result of any operation can be used as an operand of further operations, and it is possible to construct arbitrarily complex queries.

3.1 Constructing regions

The simplest sgrep expression is a text pattern, which evaluates to the set of the regions that are occurrences of the pattern. The current implementation of sgrep supports text pattern matching only by searching for either exact or case-insensitive occurrences of text phrases. For example, the file is searched below for occurrences of the phrases given between quotes. The `-o` option is used to specify that the regions are output as pairs of start and end positions, implementing a sort of inverse operation of the query in the previous section.

% sgrep -o"(%s,%e)" '"012345" or "23456789" or \

"345678" or "4567" or "456789"' file10

(0,5)(2,9)(3,8)(4,7)(4,9)

We say that a region set is *nested*, if it contains nested regions. The most important difference between sgrep and other region algebras is that sgrep is capable of manipulating nested region sets. Nested region sets are most often created by the central "followed-by" operator of sgrep, which is denoted by two dots "`..`". The "followed-by" operator takes two region sets and creates new regions starting with a region from the first set and extending to the end of a following region from the second set. The operation is useful for retrieving complete document elements by their delimiting tags:

% sgrep '"<TITLE>" .. "</TITLE>"' sgrepman.html

<TITLE>sgrep - Manual page</TITLE>

Pairs of parentheses are a typical example of nested (or recursive) structures. For an example, assume that we have the following file containing parenthesized expressions:

% cat example1

(2+5) * (10 * (7* (sqr(9) + 3))

Now the “followed-by” operator allows recognizing regions surrounded by balanced parentheses:

% sgrep -o"-%r-" '"(" .. ")"' example1

-(2+5)--(7* (sqr(9) + 3))--(sqr(9) + 3)--(9)-

The semantics of “..” is defined for arbitrary operand sets as a generalization of the way how parentheses are matched together “from inside out”. Note that the second opening parenthesis was excluded from the result above, since there was no matching pair for it.

There are also variations “`_.`”, “`._`” and “`__`” of the “`..`” operator, which exclude, correspondingly, either the opening region, or the closing region, or both, from the resulting regions. For example, the “`__`” operator can be used to retrieve contents of document elements excluding their delimiters:

% sgrep '"<TITLE>" __ "</TITLE>"' sgrepman.html

sgrep - Manual page

The operator *A* `extracting` *B*, where *A* and *B* are region sets, evaluates to the set of the regions that result by eliminating from the regions in *A* any overlap with regions in *B*. For example, contents of SGML or HTML documents without any tags could simply be viewed as follows:

% sgrep 'start .. end extracting ("<"..">")' sgrep.html

Sgrep home page

What is sgrep?

From the man page:

...

Sgrep evaluates the queries from left to right, if not indicated otherwise by parentheses. For example, above it was necessary to indicate that the sub-expression `"<".."">"` evaluating to the regions of tags was to be evaluated first.

3.2 Containment conditions

All region algebras provide similar operations for choosing regions either by their contents or by their context. In sgrep these operations are denoted for region sets *A* and *B* by operators *A* `containing` *B*, *A* `not containing` *B*, *A* `in` *B*, and *A* `not in` *B*.

The *A* `containing` *B* operator returns those regions in region set *A* that contain some region of region set *B*. For example, the H2 elements containing a given string could be retrieved as follows:

% sgrep -o"%r\n" '"<H2>" .. "</H2>" containing "EXPRES"' sgrepman.html

<H2>SYNTAX OF EXPRESSIONS</H2>

<H2>SEMANTICS OF EXPRESSIONS</H2>

The *A* `not containing` *B* operator evaluates to those regions in region set *A* that do not contain any region in region set *B*:

% sgrep -o"%r\n" '"<H2>" .. "</H2>" not containing "EXPRES"' sgrepman.html

<H2>CONTENTS</H2>

<H2>SYNOPSIS</H2>

<H2>DESCRIPTION</H2>

...

The *A* `in` *B* operation evaluates to the regions in region set *A* that are included in some region in region set *B*. The below query retrieves the TITLE-elements of any HTML file that contains the string “SGML” in the current directory:

% sgrep -o"%r\n" '"<TITLE>".."</TITLE>" in \

(start .. end containing "SGML")'   *.html

<TITLE>sgrep - Home page</TITLE>

The *A* `not in` *B* operation returns the regions of *A* that are not included in any region of set *B*. The next command counts, using the `-c` option, the number of strings “HREF” not occurring within tags in the HTML files:

%sgrep -c '"HREF" not in ("<" .. ">")'   *.html

38

3.3 Removing nested regions

Sometimes there is a need to select the innermost or the outermost regions from a nested region set, for example to select the document root element or elements consisting of text only. For this purpose there are two functions defined for a region set *A*: `inner(A)` and `outer(A)`. The `inner(A)` function returns the regions of set *A* that are not included in any other region of *A*. The `outer(A)` function returns the regions of set *A* that do not include any other region of *A*.

Often a group of regions that cover a single fragment of text is better seen as a single region. This can be obtained by the `concat(A)` function, which returns a minimal region set covering the text fragments covered by region set *A*. That is, for the result `concat` chooses the outermost regions of *A*, and combines any regions that overlap or start immediately after the preceding one into a single region. By default, if no other output format is specified, sgrep applies `concat` to the result before displaying it to the user.

3.4 Set operations

Sgrep offers typical set operations for forming unions, intersections and differences of sets. The operations are denoted by the operators *A* `or` *B*, *A* `equal` *B*, and *A* `not equal` *B*.

The operator *A* `or` *B* evaluates to the set of regions that occur either in region set *A* or in region set *B*, or in both. For example, `article` elements that discuss either dogs or cats could be retrieved by the following query:

%sgrep '"<article>" .. "</article>" containing ("dog" or "cat")' doc.sgml

The value of operator *A* `equal` *B* is the set of regions that occur in both region sets *A* and *B*. For example, any second level elements in an SGML document, which contain the string “HTML” could be obtained as follows:

%sgrep 'outer(ELEMS in ELEMS) equal (ELEMS containing "HTML")’ doc.sgml

The above query refers to a macro called `ELEMS`, which is a shorthand notation for a query that evaluates to regions of SGML elements. We will discuss using macros in the next section.

Finally, the set difference of region sets *A* and *B* can be formed using operator *A* `not equal` *B*.

3.5 Nearness conditions

Sgrep allows combining a given number of subsequent regions of a region set. Operator `join(n, A)` produces for each region of set *A* a new region which extends from the start of that region to the end of the *n*th region after it, in increasing dictionary order of the start and end points of regions. The main use of the join operator is in expressing nearness conditions. The built-in region set `chars`, whose value is the set of all single character regions, is useful in this context. For example, the following query would retrieve articles where “SGML” precedes “database” within 30 characters:

"<article>" .. "</article>" containing ("SGML" .. "database" in join(30,chars))

4 Manipulating SGML documents

In this section we show how sgrep can be applied to the manipulation of SGML documents. It should be emphasized that sgrep is *not* a real SGML tool, since it does not contain an SGML parser. Sgrep can be made to *mimic* SGML awareness, but that requires the documents to be fully tagged, and the queries need to consider the concrete markup syntax and some peculiarities like empty elements. After all, sgrep only picks text regions satisfying the patterns and conditions in its search query.

Expressing the fetching of SGML elements by sgrep queries directly would often be too tedious. For this reason sgrep provides a command line option `-p` for filtering the queries through an external preprocessor before evaluating them. Currently the `m4` macro processor is used for this purpose. Macro definitions can be passed to sgrep in files using its `-f` option. Below we show samples of m4 macro definitions that we have found useful for querying SGML documents. The first eight lines define ready-to-use queries for locating markup of tags, and allow using shorthand identifiers like `ELEMS` in sgrep queries.

define(ETAGO, "</") # End-TAG Open

define(MDO, "<!") # Markup Declaration Open

define(PIO, "<?") # Proc. Instruction Open

define(STAGO, ("<" not in (ETAGO or MDO or PIO))) # Start-TAG Open

define(TAGC, ">") # TAG Close

define(STAG, (STAGO .. TAGC)) # Start TAG

define(ETAG, (ETAGO .. TAGC)) # End TAG

define(ELEMS, (STAG .. ETAG)) # all elements

The next macro definitions pertain to recognizing generic identifiers and attribute lists of elements. These macros allow retrieving values of named attributes by calls like `NAMED_ATTRVAL(status)`. The `BLANK` macro evaluates to the regions of single white space characters.

define(VI, "=") # value indicator

define(LIT, "\"") # literal string delimiter

define(ALIT, "'") # alternative literal string delimiter

define(GI, (STAGO __ (BLANK or TAGC) in STAG)) # Generic Identifier

define(ATTR_LISTS, (GI __ TAGC)) # All attribute lists

define(ATTR_NAMES, (BLANK __ (VI or BLANK)

in (BLANK __ (VI or BLANK) .. VI in ATTR_LISTS)))

define(ATTRS, (ATTR_NAMES ._ (ATTR_NAMES or TAGC)))

define(NAMED_ATTRS,(ATTRS containing ("$1" equal ATTR_NAMES)))

define(NAMED_ATTRVAL, ((VI __ (ATTR_NAMES or TAGC)

```
not containing (LIT or ALIT)  extracting BLANK)
or (LIT __ LIT) or (ALIT __ ALIT)
in (NAMED_ATTRS($1) extracting ATTR_NAMES)) )
```

The next macro definitions allow sgrep queries to retrieve element tags, elements and their contents by the GI (generic identifier) of the element:

```
define(NAMED_STAG,( "<$1>" or ( ("<$1 " or "<$1\t" or "<$1\n")  .. ">")) )

define(NAMED_ELEMS,(NAMED_STAG($1) .. "</$1>") )

define(NAMED_CONTENTS,( NAMED_STAG($1) __ "</$1>" ))
```

Writing such macro definitions is subtle, but once they have been written and tested, they make expressing high-level queries quite easy. For example, if an SGML document file contains `book` elements that express the number of pages in the book by a `pages` attribute and the price of the book in a `price` element, the following query would find the books that have 200 pages and cost $20:

```
NAMED_ELEMS(book) containing
    (NAMED_ATTRVAL(pages) equal "200" in NAMED_STAG(book))
    containing (NAMED_CONTENTS(price) equal "$20")
```

The below command counts the total number of chapters in “The SGML Handbook”, when the title of a book is given in its `name` element:

```
sgrep -c 'NAMED_ELEMS(chapter) in      \
    (NAMED_ELEMS(book) containing   \
        (NAMED_CONTENTS(name) equal "The SGML Handbook"))'
```

Similarly books could be retrieved by the contents of their `author` element:

```
NAMED_ELEMS(book) containing
    (NAMED_ELEMS(author) containing "Herwijnen")
```

4.1 Applying sgrep to document assembly

An on-going research and development project called SID (Structured and Intelligent Documents) is concerned with adding intelligent features to documents in order to make their reuse and assembly more efficient [AHH96]. The SID project participates in the development of a dynamic collection of text books for the National Board of Education. A goal is to allow easy production of tailored high-quality versions of the books in the collection. To enable this the books have been converted from a Word format to a slightly modified ISO 12083 book DTD. The modifications include the enrichment of books with attributes for specifying, e.g., the didactic form (introductory, base material, advanced, applied, etc.) of the contents of document elements.

We have developed an architecture which allows tailored versions of the books to be created by filling an order form, specifying in a table of contents which parts of a book are to be chosen to the assembly, and possibly restricting the chosen contents to be of appropriate didactic form. (See Figure 4.1.)

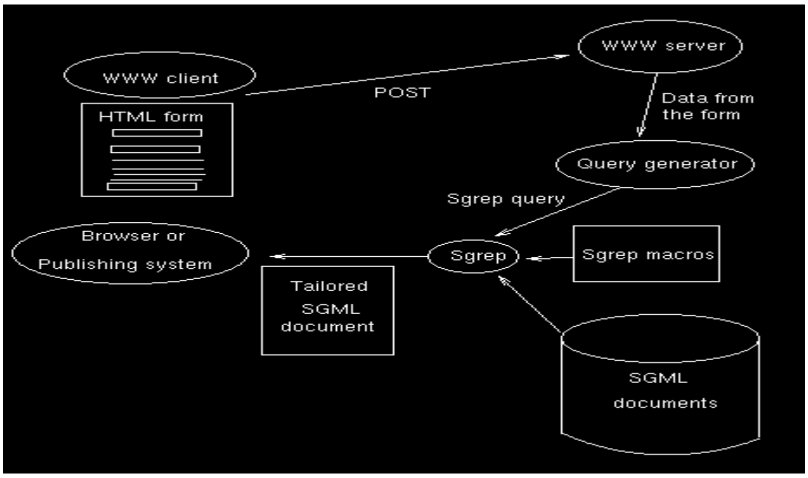

Figure 4.1 The assembly architecture

The crux of the architecture is the compilation of the order form into a query that selects and combines the document fragments form the collection. As the current solution we translate the form into an sgrep query. The translation is based on the following principles. First, the query generator creates a definition of a macro called `BASE_FRAGMENTS`, which evaluates to the fragments chosen by the form. A chapter selected and restricted to a didactic form in the order form is translated into a sub-expression of `BASE_FRAGMENTS`, roughly of the following form:

or CHAP_BY_ID(<Id of the chosen chapter>)

extracting P_NOT_OF_CONTENT(<didactic form chosen>)

These expressions refer to sgrep macros written specially for the material in the collection. The macros, written in upper-case above, allow retrieving chapters and sections by their ID attribute values, and paragraphs by the attribute value for their didactic form. The definitions of these macros are in principle similar to the macro definitions we have seen above.

Similarly, sub-queries for the regions of each selected section are added to the query macro. Then some additional parts are added to the result query, both to wrap the book fragments into complete SGML elements and to add introductory context like initial text paragraphs to the sections retrieved in the assembly:

NAMED_STAG(chapter) in (CHAPTERS containing BASE_FRAGMENTS)

. _ NAMED_STAG(section)

or NAMED_ETAG(chapter) in (CHAPTERS containing BASE_FRAGMENTS)

Finally, the front and back materials of the original books are included in the assembly.

The generated sgrep query creates a digest of the original SGML documents by retrieving their fragments in order. When the fragments are picked carefully, by applying the principles outlined above, the result is a valid SGML instance, which can be loaded either to a browser or to an editor. We have tested the prototype of the architecture using the Softquad Panorama browser. The production of book-quality layouts is planned to be done by loading the results in FrameMaker+SGML.

5 Conclusion

We have described the main features of the sgrep query language, which implements an unlimited algebra of text regions. Even though the algebra and its implementation give no interpretation to the contents of documents, we have seen above that sgrep can be used for rather high-level manipulation of, say, SGML documents.

Sgrep has been developed and tested in Linux and Unix systems. The software, currently in version 0.99, is freely available at `ftp://ftp.cs.helsinki.fi/pub/Software/Local/Sgrep/`.

Biography: Jani Jaakkola. Jani Jaakkola is a full-time student of computer science at the University of Helsinki. Currently he works as a systems designer in the SID (Structured and Intelligent Documents) project.

Biography: Pekka Kilpeläinen. Pekka Kilpeläinen is an assistant professor of computer science at the University of Helsinki. He received his master's degree in 1989, the degree of the licentiate of philosophy in 1992, and his Ph.D. in 1993, all in computer science at the University of Helsinki. Currently he acts as the project manager in the SID project.

References

[AHH96] Constructing tailored SGML documents. H. Ahonen, B. Heikkinen, O. Heinonen, J. Jaakkola, P. Kilpeläinen, G. Lindén and H. Mannila. This volume. 1996.

[AKW88] The AWK Programming Language. A. V. Aho, B. W. Kernighan and P. J. Weinberger. Addison-Wesley. 1988.

[Bur92] An Algebra for Hierarchically Organized Text-Dominated Databases. F. J. Burkowski. Information Processing & Management. 1992. (333-348)

[CCB95] An Algebra for Structured Text Search and a Framework for its Implementation. C.L.A. Clarke, G.V. Cormack and F. J. Burkowski. The Computer Journal. 1995. (43-56)

[JK96] Sgrep - A Tool to Search Structured Text. J. Jaakkola and P. Kilpeläinen. University of Helsinki. (In preparation) 1996.

[SaT92] PAT Expressions: an Algebra for Text Search. A. Salminen and F. Wm. Tompa. UW Centre for the New Oxford English Dictionary and Text Research report OED-92-02. 1992.

[ThR75] UNIX Programmer's Manual. K. Thompson and D.M. Ritchie. Bell Laboratories. 1975.

[WaS91] Programming Perl. L. Wall and R.L. Schwartz. O'Reilly & Associates, Inc.. 1991.