PAT expressions: an algebra for text search

Airi Salminen

Department of Computer Science
University of Jyväskylä
Seminaarinkatu 15
40100 Jyväskylä
Finland

Frank Wm. Tompa

Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

1. Introduction

Text search operations are used to locate and retrieve needed information from some text collection. In traditional information retrieval, text search is a means for identifying relevant documents [Salton83, Lee85]. By specifying selection criteria for the text of a document, the reader can choose a subset of a given set of documents. If the text collection is defined not as a set of documents, but more generally as a structure containing some parts, then text search involves the specification of those parts of interest to the reader. The structure of the documents may be determined by the search system, by the author, by the text installer, or by the reader.

In the PAT™ system [Gonnet87a, Fawcett89a, Fawcett89b] text search operations are expressions that efficiently combine traditional search capabilities with some new, powerful facilities. PAT contains means for lexical search, proximity search, contextual search and Boolean search [Hollaar79, Larson84, Lee85, Burkowski91]. It also contains more rare operation types, including position and frequency search. Furthermore, a novel feature in PAT is the capability by which a reader can define structures for a text and use these structures in subsequent operations. One of the goals of this paper is to introduce the powerful search capabilities of PAT expressions.

Text search is usually considered so simple that only a rough description of the operations is given. For example, when word search is discussed, we are seldom told what is meant by a "word". The reader has to find out through experimentation how many words are contained in the strings "Jean-Marie" and "O'Hara". However, a careless description of search operations may lead to search errors or unnecessarily long retrieval sessions. A second goal of the paper, therefore, is to introduce a mechanism for precise specification of text search semantics.

† PAT is a registered trademark of Open Text Corporation.

Text search using PAT is typically simple and straightforward [Raymond90]. However, because of the powerful definition capabilities included in PAT, explaining and understanding the semantics of some operations may be difficult. As a side-effect of our systematic specification of PAT, we have identified some features of PAT expressions that cause problems and thus would benefit from further development. From this we see that precise specification also serves as a means for evaluation and offers a means for comparing text search systems.

As is common in information retrieval systems, a PAT search is applied to indexed text [see, for example, Gonnet83, Croft84, Larson84, Faloutsos85, Salton89, Burkowski91]. Indexing is usually described from the point of view of implementation, for example, by giving an algorithm for the indexing [Salton81, Salton89, Gonnet91]. However, since the way text is indexed affects search behaviour, our systematic approach to precise description must include mechanisms that accommodate indexing definition capabilities.

2. The PAT system

PAT is a text searching system developed at the University of Waterloo's Centre for the New Oxford English Dictionary and Text Research [Berg91, Tompa92], and commercially available from Open Text Corporation in Waterloo. PAT is used, for example, for searching the 570 megabyte *Oxford English Dictionary*, among other texts and text collections. As distributed, the PAT system has two different end-user interfaces: one based on a command language using PAT expressions, the other based on direct manipulation interaction in which the system creates PAT expressions from the point-and-click actions of the end user. PAT also provides an applications programmers' interface by means of which customized front ends can be written, communicating with the search engine via PAT expressions.

The text search is based on three innovative techniques: PAT indexing, region definitions, and PAT expressions. PAT accesses the original text with the aid of an associated index [Gonnet91] that maps index terms to their occurrences in the text, which we shall call *indexed elements*. The choice of which text segments to index in PAT is not fixed by the system. Instead, PAT's indexing offers the text installer the capability to define the indexed elements of the text. The indexed elements may consist of, for example, all individual characters of the text, or multicharacter words separated by delimiters. Each indexed element is considered to be the start of a "semi-infinite string" that continues to the end of the text. In processing a query, the system finds those characters that begin semi-infinite strings matching the string given by the user. The text installer defines how the matching is determined. It is as simple to find single indexed elements or their prefixes as it is to find longer phrases, consisting of several indexed elements. If all characters are defined as indexed elements, the search in an indexed text corresponds to truly full-text search (where "the" matches the second character in "other").

In response to a query, PAT returns a result set, which is either the set of *match points* or the set of *regions* satisfying a given criterion, expressed by a PAT expression. A match point is a character (starting a semi-infinite string), whereas a region is a substring of the text beginning and ending with specified characters. Regions are defined either by the text installer or by the reader. If the text is tagged (e.g., using SGML's reference syntax [Goldfarb90]), the regions with a given name can be defined to begin and end with given tags. Alternatively, any characters found by text search can be used to denote the starts and ends of regions.

Result sets from one PAT expression can be used as operands for subsequent expressions. By default, a search command shows the count of match points or regions in the result set. If the result set is a match point set, characters from the left and right context of the match points in the set can be printed by a printing command. If the result set is a region set, the reader can print either regions from the set or characters from the left and right context of the beginning characters of the regions. In a workstation environment the text of a region may be displayed in a formatted form in a LECTOR™ window [Raymond91].

3. Text in PAT

In earlier work, grammars were used to describe indexing as a view of the original text, and this view was then further used to specify text retrieval operations [Tague91]. In this paper we will use the same approach, based on the text model introduced in [Salminen92], which extends the earlier grammar-based model of [Gonnet87b].

3.1. Text as a character string

Any PAT text can be viewed as a sequence of characters. The following grammar, consisting of two productions, can be used as the schema for any such text:

(S) **string ::= character+ .**
character ::= 'a' | 'b' | ... .

The grammar defines two text *types*: **string** and **character**. The first production indicates that a string consists of one or more characters. (The symbol + in the production denotes iteration one or more times.) Thus, a *value* of type **string** is any nonempty sequence of values of type **character**. The second production enumerates all available values of type **character**. (The symbol | separates alternatives.)

Consider the following sample text:

```
<h>Consumer spending in U.S. up 1.5 per cent in June</h>
```

When we use this text as a text context for the schema **(S)**, it is viewed as having been parsed by the grammar, and we can refer to *parts* of the text. Each part is a text entity corresponding to one of the defined types. The sample text contains one part of type **string**, i.e. the whole text, and 56 parts of type **character**. The value of the **string** part is the complete sample text above. The value of the first **character** part is "<", the value of the second **character** part is "h", the value of the 49th **character** part is "J", and so on. PAT capitalizes on the fact that each **character** part is identified with a unique position in the whole text.

3.2. Text as an indexed string

Alternatively, PAT text can be viewed as a string consisting of indexed elements and delimiters separating indexed elements. Each indexed element begins a new phrase, continuing to the right from the indexed element until the end of the text. Therefore, if the string contains n indexed elements, it also contains n such phrases. For the above sample text, if (for simplicity) each contiguous sequence of non-blank characters is an indexed element and blanks are delimiters, there are ten phrases corresponding to the ten indexed elements:

† LECTOR is a registered trademark of Open Text Corporation.

```
<h>Consumer spending in U.S. up 1.5 per cent in June</h>
spending in U.S. up 1.5 per cent in June</h>
in U.S. up 1.5 per cent in June</h>
U.S. up 1.5 per cent in June</h>
up 1.5 per cent in June</h>
1.5 per cent in June</h>
per cent in June</h>
cent in June</h>
in June</h>
June</h>
```

We can define this formally using the following rules:

(a) **string ::= [delimiter] phrase .**
(b) **phrase ::= indexed_element [delimiter] [phrase] .**

The first production indicates that a string may be a phrase or it consists of a delimiter followed by a phrase. (The square brackets denote optionality.) The second production shows that a phrase always begins with an indexed element, which may be followed by a delimiter and then by another phrase. Thus, two indexed elements may be separated by a delimiter, or one indexed element may follow another directly.

Indexed elements and delimiters are described specifically for each text file by the text installer. For most text retrieval applications, text installers choose indexed elements to correspond as closely as possible to the users' perception of "words". Figure 1 shows a possible grammar describing the indexing of a text containing standard markup tags (e.g., this grammar describes the indexing used for the *Oxford English Dictionary* at the University of Waterloo and elsewhere).

The grammar defines one way to divide a string into indexed elements and delimiters. So that an indexed element can be easily recognized syntactically, it must either begin with a specific character (**preemptive_element**) or it must be preceded by a delimiter. This constraint is specified in production (2) by replacing the use of the simple non-terminal in (b) above by the "property" **phrase {where indexed_element {is preemptive_element or preceded by delimiter} }** (see [Salminen92] for a detailed explanation of properties). Preemptive elements are either single characters (for this specific grammar, hyphens constitute the only such elements), or they start with a **signal_char** (for this grammar, either a less-than-sign or an ampersand) and then continue with zero or more element characters (here any combination of letters, digits, hash-sign, or slash). Delimiters contain characters distinct from any of these characters. Productions (7) - (10) contain the text-specific assignment of characters to the four classes that define which substrings of the text are indexed elements. They use the nonterminals defined in productions starting from (11) so that character transformations can be separately specified for evaluating whether a given phrase matches a query (see Section 4.1). The effect of this particular choice of indexing is that most punctuation characters are treated as blanks (periods, commas, etc. are not part of any indexed elements).

Consider again the sample text above. As PAT text this is a sequence of characters, containing parts of types **string** and **character**. However, using Figure 1 as an indexing description for the text means that the text is reparsed by the grammar. The text context then includes simultaneously parts having types from the schema (S) and parts having types from the indexing description. The text context contains 14 indexed elements which are shown below (the extent of each marked by `|-----`).

```
 (1)    string ::= [delimiter] phrase .
 (2)    phrase ::= indexed_element [delimiter]
                [phrase {where indexed_element {is preemptive_element or preceded by delimiter} }] .
 (3)    indexed_element ::= preemptive_element  |  limited_element .

 (4)    preemptive_element ::= stand_alone_char  |  signal_char element_char* .
 (5)    limited_element ::= element_char+ .
 (6)    delimiter ::= delim_char+ .

 (7)    stand_alone_char ::= hyphen .
 (8)    signal_char ::= less | ampersand .
 (9)    element_char ::= a | ... | z | A | ... | Z | d1 | ... | d9 | hash | slash .
(10)    delim_char ::= blank | period | comma | greater | colon | apostrophe | ... .

(11)    hyphen ::= '-' .
        less ::= '<' .                                ampersand ::= '&' .
        a ::= 'a' .              ...                  z ::= 'z' .
        A ::= 'A' .              ...                  Z ::= 'Z' .
        d0 ::= '0' .             ...                  d9 ::= '9' .
        hash ::= '#' .         slash ::= '/' .
        blank ::= ' ' .        period ::= '.' .       comma ::= ',' .
        greater ::= '>' .      colon ::= ':' .        apostrophe ::= ''' . ... .
```

Figure 1: A grammar describing an indexed text.

```
<h>Consumer spending in U.S. up 1.5 per cent in June</h>
|- |------- |------- |- | |  |- | | |-- |--- |- |---|--
```

Notice that delimiters are not contained in **indexed_element** parts. Furthermore, the two last indexed elements are not separated by a delimiter: the character '<' begins a new indexed element immediately.

The grammar described in Figure 1 has been used most often for PAT applications. From the grammar it is clear that “Jean-Marie” contains three indexed elements and “O’Hara” contains two. An alternative index that makes *every* character an indexed element has been used to support detailed proofreading of the *Oxford English Dictionary* in preparation for the second edition. Between these two extremes exist many other possibilities: for example, specification and program text can be indexed on all upper case letters as well as conventional word starts so as to allow a software engineer to find all mention of functions and variables having to do with a “window” even if they are named “AdjustWindowImage” and so forth. Such indexing can be simply described through reassignment of upper case letters to **signal_char** instead of **element_char**.

This is not intended to describe the way PAT indexing is implemented [Gonnet91]. Instead, we describe the indexing from a reader’s viewpoint, specifically, the effect of indexing on search operations. From Figure 1, the addressable units of a text are clearly identified. As opposed to the simplistic listing of phrases given above, it is precisely stated that the sample text with such an index has 14 retrievable phrases:

```
<h>Consumer spending in U.S. up 1.5 per cent in June</h>
Consumer spending in U.S. up 1.5 per cent in June</h>
spending in U.S. up 1.5 per cent in June</h>
in U.S. up 1.5 per cent in June</h>
U.S. up 1.5 per cent in June</h>
S. up 1.5 per cent in June</h>
up 1.5 per cent in June</h>
1.5 per cent in June</h>
5 per cent in June</h>
per cent in June</h>
cent in June</h>
in June</h>
June</h>
</h>
```

Because the indexing described in Figure 1 includes period as a delimiter character, the strings "`U.S.`" and "`1.5`" each contain two indexed elements. The current PAT indexing capability is very simple, and we cannot define context-dependent differences in the use of a character. Thus even though the strings "`U.S.`" and "`1.5`" might seem more appropriately considered as one indexed element each, we cannot define them so without causing end-of-sentence periods to also be included in indexed elements.

Slashes may also cause some problems with this indexing. Because the indexing description is made for text with standard markup begin and end tags (for example, "`<h>`" and "`</h>`"), it is defined such that each begin and end tag contains one indexed element: for example, "`<h`" and "`</h`". (The character '>' ending a tag is defined to be a delimiter.) Slash must be defined as an indexed element character if the identifier within the end tag (e.g., the "`h`" within "`</h>`") is not to be made a separate indexed element. However, as a result the substring "`USA/Canada`" within a text context will also be treated as one indexed element. Subsequently, if the reader searches for occurrences of "`Canada`", the substring within "`USA/Canada border`" fails to match.

From these examples, we see that defining the indexing of text properly is somewhat problematic with PAT. By extending PAT's indexing definition capabilities, more flexibility for application-dependent indexing could be achieved. However, whatever the indexing definition techniques, the richness of natural language and the variety of information needs from natural language text will always cause problems in defining indexing satisfactorily.

3.3. Regions

A region is a substring of PAT text, beginning and ending at specified characters. Each region belongs to one or more *region sets*. A region cannot overlap other regions belonging to the same region set, but it can overlap regions in other sets arbitrarily. This concept is thus a unification and generalization of the concepts of "document" and "field" used in conventional information retrieveal systems.

From within PAT, a region set can be created by a region definition, which gives the condition determining the first and last characters of each component region. Region sets can also be derived from prior region sets. The form of the region definitions is described in Section 4.4.

A region definition corresponds to a new grammar by which the text context can be reparsed, and through which new parts can be identified in the text. For example, for the following text we could define a region set, named "year", to include the two regions as indicated:

```
Fascicles of the OED appeared between 1884 and 1928.
                                      ----     ----
```

The capability to handle regions is the feature that most distinguishes PAT from conventional document retrieval systems. Regions are defined either by the text installer (pre-defined regions) or by the reader (user-defined regions), the latter serving as temporary definitions of scope or as personal "views" of the text for limiting queries or responses. It is primarily through judicious definition of region sets that texts can be structured to meet the needs of diverse applications.

4. PAT operations

PAT is a set-at-a-time algebra for manipulating results of text queries. Each PAT expression is either a match point expression, specifying a set of characters in the text context, or a region expression, specifying a set of regions in the text context. The sets specified by match point expressions or region expressions are called match point sets and region sets, respectively; collectively they are called *result sets*.

Result sets produced by search commands are numbered sequentially, and they can be referenced in subsequent expressions by number. A user can optionally assign a name to a result set via a search command, in which case the result set can be subsequently referenced by name. If e is a PAT expression, then a command of the form

$$n = e$$

gives the name $*n$ to the result set specified by e. If e is a match point expression, then $*n$ refers to the corresponding match point set; if e is a region expression, $*n$ names the corresponding region set. Finally, the symbol % refers to the immediately preceding result set.

The syntax of PAT expressions is described in the Appendix. PAT operations can be classified by type:

	search class	*result set type*
(1)	lexical search	match points
(2)	position search	match points
(3)	frequency search	match points
(4)	region definition	regions
(5)	restriction	match points / regions
(6)	augmentation	match points / regions

As indicated in the table, the resulting expressions in a class can be match point expressions or region expressions only, or both.

Expressions in classes (1), (2), and (3) are always match point expressions, i.e. they are used to search for characters. In lexical search the user searches for phrases by giving one or two patterns. The result set consists of the first characters in the found phrases. Position search means searching for a character in a given position in the whole text, or searching for characters at fixed offsets to the left or right of the match points of a given match point set. Frequency search means identifying match points for frequently appearing substrings or long repetitions from the text. Expressions from classes (1), (2), and (3) are discussed further in Sections 4.1, 4.2 and, 4.3, respectively.

Expressions in class (4) are always region expressions: a region definition specifies a region set as a function of two match point sets. Region definitions are discussed further in Section 4.4.

Expressions in classes (5) and (6) are either match point expressions or region expressions. These produce new result sets as a function of existing sets and are discussed in more detail in Sections 4.5 and 4.6, respectively.

4.1. Lexical search

A PAT expression for lexical search is either a character string or of the form $s_1..s_2$ where s_1 and s_2 are character strings. Lexical search always yields a match point set.

By giving a string s the reader searches for all characters in the text that begin phrases matching s. The matching of a phrase with a string pattern is determined after normalizing both the phrase and the pattern. The normalization inherent in PAT maps delimiter characters to blanks. However, concurrently with the indexing, the text installer may describe additional normalization, through which characters can be deleted or replaced by alternative characters.

In grammar based text modelling, normalization can be described as a text translation, defined by a set of translation productions that redefine those text types of the indexing description whose parts are changed by normalization. The translation productions, together with the indexing description, define a translation of a character string to a normalized string.

Figure 1 showed one possible indexing for a text. For such a text, normalization could be defined by a table showing replacement productions as follows:

indexing production	*normalization replacement*
string ::= [delimiter] phrase .	**string ::= phrase .**
delimiter ::= delim_char+ .	**delimiter ::= ’ ’ .**
A ::= ’A’ .	**A ::= ’a’ .**
...	**...**
Z ::= ’Z’ .	**Z ::= ’z’ .**

The replacement for the **string-**production indicates that if the string begins with a delimiter, the delimiter is removed in normalization. The **delimiter-**production causes the replacement of all delimiters by blanks, and the rest of the productions define that each upper case letter is replaced by the corresponding lower case letter. The effect of these replacements is that pattern matching is case-insensitive, delimiters cannot be distinguished, and the presence or absence of a delimiter before the start of a match cannot be determined.

As described in Section 3.2, PAT allows a text installer to define the partitioning of characters into the classes **stand_alone_char**, **signal_char**, **element_char** and **delim_char** to customize indexing. The text installer customizes normalization by assigning replacement characters as well, under the constraint that replacements must be chosen from the same class as the character being replaced (e.g., lower case letters can be substituted for upper case letters only if both are in the same class, in this case **element_char**). There is also a limited facility for defining stopwords, that is, specific characters strings that are to be treated as delimiters rather than as indexed elements. Extensions to the language are being developed to generalize these capabilities by allowing a text installer to define the indexing and normalization transductions using a powerful language for describing Mealy machines [Gonnet92]. In the examples of the rest of the paper we assume that the indexing is defined by the grammar in Figure 1 and the normalization by the translation productions given above.

A phrase in the text matches a given string if the normalized string is a prefix of the normalized phrase. Consider our earlier sample text:

```
<h>Consumer spending in U.S. up 1.5 per cent in June</h>
```

The string “in” matches two phrases:

```
in U.S. up 1.5 per cent in June</h>
in June</h>
```

The string “`s`” also matches two phrases:

```
spending in U.S. up 1.5 per cent in June</h>
S. up 1.5 per cent in June</h>
```

because through normalization the sample text is translated to the form

```
<h consumer spending in u s up 1 5 per cent in june</h
```

Thus for each of these lexical searches, the expression denotes a result set consisting of two match points: the first characters of each matching phrase. Note that the strings “`U.S. `”, “`u s u`”, and “`..., ’U:’`” all match the phrase “`U.S. up 1.5 per cent in June</h>`”:

string	*normalized form*
`U.S. up 1.5 per cent ...`	`u s up 1 5 per cent ...`
`U.S.`	`u s`
`u s u`	`u s u`
`.., ’U:’`	`u`

By giving two strings $s_1..s_2$ the reader searches for all characters that begin phrases such that the normalized phrase matches either normalized s_1 or normalized s_2 as above or it follows normalized s_1 and precedes normalized s_2 in lexicographic order. In many applications this kind of search capability is very useful and may often replace a long sequence of searches for one string at a time [Logan88]. As an example consider the text

```
shortages hit in 1973 and 1979. In the 1980s ... in 1978 ...
```

For the expression "hi".."jo", the result set contains four match points corresponding to the initial characters of the phrases

```
hit in 1973 and 1979. In the 1980s ... in 1978 ...
in 1973 and 1979. In the 1980s ... in 1978 ...
In the 1980s ... in 1978 ...
in 1978 ...
```

which can be represented diagrammatically by the notation

```
shortages hit in 1973 and 1979. In the 1980s ... in 1978 ...
          |   |                 |                |
```

For the expression "1975".."1980" the result set contains three match points:

```
shortages hit in 1973 and 1979. In the 1980s ... in 1978 ...
                          |            |            |
```

4.2. Position search

Position search is used to find a character in a given position in the whole text, or characters at a given distance to the left or right of match points in a match point set. The expression for the first kind of search is

$$[n]$$

where n is a positive integer. As an example, with respect to the following text the PAT expression “[15]” denotes the 15th character, namely the “`u`” in “`Yugoslavs`”, and thus the indicated match point:

```
Some 700,000 Yugoslavs live in Germany, most of them Croats.
              |
```

Unlike for lexical searches, the match point denoted by a position search need not be the initial character of an indexed element.

Match points can be shifted by expressions of the form

$$\text{shift.}n\ e$$

where n is a positive or negative integer and e is an expression. The expression e is considered to specify a match point set: if it is a region expression, the match points correspond to the first characters of the regions. For example, with the expression

shift.3 "1800".."2000"

we can find the two indicated match points in the following text:

```
Fascicles of the OED appeared between 1884 and 1928.
                                          |        |
```

Again the match points in the result set need not correspond to starts of phrases.

4.3. Frequency search

PAT includes two groups of frequency expressions, used to find frequently occurring or repeated substrings in a text. The “signif” expressions are used to find the most frequent substrings, consisting of whole indexed elements, and beginning a phrase that matches a given string. The “lrep” expressions are used to find the longest repeated substrings, each consisting of whole indexed elements, beginning with a phrase that matches a given string. The frequency search expressions are always match point expressions.

Consider the frequency expression

signif *e*

where *e* is a match point expression. This operation first normalizes the character strings starting at match points corresponding to *e* and identifies for each one its prefix up to the first delimiter. From the corresponding match points, “signif” returns the subset associated with the most frequently occurring prefix. In the complete works of Shakespeare [OUP88], 792 words begin with the string “thro”. The command

signif "thro"

returns a match point set with 329 members, corresponding to the initial characters of each occurrence of the word “through” in the text, that being the most frequent word beginning with “thro”.

Using the form

signif.*n e*

a user can specify that extended prefixes, including *n* delimiters rather than stopping at the first one, should be compared. For Shakespeare, the following results can be obtained:

expression	*number of matches*	*matching phrase*
signif.2 "thro"	107	`through the`
signif.3 "thro"	8	`through the world`
signif.4 "thro"	2	`throat our height can`

Using the form

signif.–*n s*

the user can retrieve result sets corresponding to the *n* most frequent normalized prefixes beginning with string *s*. For Shakespeare, the following example illustrates this form:

```
signif.-10 "thro"

  329 matches, text=through
  116 matches, text=throw
  107 matches, text=through  the
   77 matches, text=throne
   58 matches, text=throat
   46 matches, text=throws
   35 matches, text=thrown
   31 matches, text=throats
   21 matches, text=throwing
   20 matches, text=throng
```

Note that since the two word prefix "through the" occurs more frequently than do remaining single words, the results set corresponding to this extension of "through" is returned before, for example, the match points corresponding to "throne". Because a *sequence* of match point sets are returned, this form of the command cannot be used within other PAT expressions.

The operation

lrep *e*

chooses from the match points identified by *e* those having the longest normalized extensions such that there are at least two occurrences of the same extension. The result set consists of the subset of match points corresponding to the first characters of these phrases. Using the form

lrep.*n* *e*

all repeated phrases having at least *n* characters in the common prefix can be identified. Looking again at Shakespeare,

```
lrep "thro"
```

returns a result set containing two match points corresponding to a repeated phrase having 162 characters (including line numbers from the play):

```
            ..throw incense. Have I caught thee?
22  He that parts us shall bring a brand from heaven
23  And fire us hence like foxes. Wipe thine eyes.
24  The goodyear shall devour [’em, flesh and fell,]

            ..throw incense. Have I caught thee?
22  He that parts us shall bring a brand from heaven
23  And fire us hence like foxes. Wipe thine eyes.
24  The goodyear shall devour [them, flesh and fell,]
```

which represents a variant edition included in the text. Similarly,

```
lrep.100 "thro"
```

returns a result set with four match points, two of which are as above and the other two representing another variant edition:

```
                 ..through itself to that full issue
      4  For which I razed my likeness. Now, banished Kent,
      5  If thou canst serve where thou dost stand condemned,
      6  [So may it come thy master, whom thou lov’st,]

                 ..through itself to that full issue
      4  For which I razed my likeness. Now, banished Kent,
      5  If thou canst serve where thou dost stand condemned,
      6  [Thy master, whom thou lov’st...]
```

Frequency expressions have been included in PAT for the needs of linguists and editors. In an experiment with users the frequency search operations were found among the most difficult to use [Raymond90]. Therefore, improvements resulting in more useful frequency search operations are currently being investigated.

4.4. Region definitions

A region definition specifies a region set consisting of new regions. It has the form

$$\text{docs } e_1..e_2$$

where e_1 and e_2 are match point expressions. Expression e_1 gives the condition for the first character of a new region and e_2 a condition for the last character. If e_1 or e_2 is a region expression, it denotes the set of the first characters of the argument regions.

Suppose we define

year = docs ("1800 ".."2000 ") .. (shift.3 "1800 ".."2000 ")

Then both years in our earlier sample text would be regions in the set named by the expression *year:

```
    Fascicles of the OED appeared between 1884 and 1928.
                                          ----     ----
```

Alternatively, if a region set named “year” had been defined by the text installer, it could be referenced by the expression “docs year”.

From a tagged text we can define regions by matching the tags. For example, if the text consists of articles with headlines, publication dates, authors, and paragraphs, such that each of these parts is denoted by tags, we can define the structure in terms of PAT expressions. Specifically, assuming that headlines are denoted by the tags “<h>” and “</h>”, the definition

headline = docs "<h>"..(shift.3 "</h>")

defines a region set called *headline in which each element spans the substring from the first character of the opening tag to the last character of the closing tag.

Each region definition creates a region set independently of other region definitions. There are no constraints on how regions in one set overlap earlier defined regions. However, the regions defined in one definition are not self-overlapping. Thus, if a text includes the following fragment:

```
    <h>Editor denies <h>Massive Failure</h> misleading</h>...
```

the definition of headline given above would include the region corresponding to the substring “`<h>Massive Failure</h>`” but not the surrounding headline.

4.5. Restriction

Users often require means to select subsets from a given set. PAT provides several binary operators of the form

$$e_1 \; op \; e_2$$

which designate result sets that are subsets of the set corresponding to e_1, consisting of elements that satisfy the condition expressed by “$op\ e_2$”. Thus if e_1 is a region expression, the result is a region set; if e_1 is a match point expression, the result is a match point set.

The first operator “including” allows users to find those regions that contain at least one member of a given match point set. For example, consider again the complete works of Shakespeare and assume that the name *speech has been defined to designate the set of regions corresponding to all speeches from all the plays. Hence, the expression

*speech including "wherefore art"

returns the subset of speeches containing match points corresponding to the lexical search, namely, the two speeches:

```
 ..<S JULIET> <T asd> {(not knowing Romeo hears her)}<T verse> O Romeo,     +
75  Romeo, wherefore art thou Romeo?
76  Deny thy father and refuse thy name,
77  Or if thou wilt not, be but sworn my love,
78  And I’ll no longer be a Capulet.
```

and

```
    ..<S FLAVIUS><T verse> But wherefore art not in thy shop today?
28  Why dost thou lead these men about the streets?
```

In general, the region expression

$$e_1 \text{ including.} n \; e_2$$

yields a result set consisting of those regions in the set specified by e_1 which contain at least n match points specified in e_2 (with “. 1” as the default). Thus *speech including.7 "Romeo"

returns one speech (Juliet's final monologue).

Similarly,

$$e_1 \text{ not including.}n\ e_2$$

returns the complementary subset. Thus, to find all Shakespeare's speeches that include the word "dream" but no word beginning with "sleep", a user can write the expression

(*speech including "dream ") not including "sleep"

If the constraining expression is a region expression, then the match point set used to test membership in the result set consists of the first characters of the regions. Continuing with the previous examples, if *scene designates all scenes from Shakespeare's plays,

*scene including.100 *speech

returns all scenes having 100 or more speeches. However, it is important to recognize that PAT only checks that the *start* of the speech is in the scene, which for nested regions is sufficient for the *whole* speech to be in the scene. Thus it follows that if "line" designates all lines from the plays,

*line including *speech

returns the set of lines that contain starts of speeches, as opposed to those containing complete speeches.

The "including" operator produces a result set that is a subset of a region set. PAT provides other operators that produce a subset of an *arbitrary* set by restricting membership according to the following constraints:

op	*explanation*
^	coincident with some member of e_2
–	not coincident with any member of e_2
fby.*n*	preceding some member of e_2 by at most n characters
near.*n*	separated by at most n characters from some member of e_2
within	contained in some region designated by e_2

A user wishing to know which sonnets contain suffixed instances of the word "love" could write the expression

*sonnet including ("lov" – "love ")

assuming prior construction of the set *sonnet. The expression "lov" includes (among many others) match points in the following lines from several sonnets:

```
13  And so of you, beauteous and lovely youth,
 3  Both grace and faults are loved of more and less;
14  Those that can see thou lov’st, and I am blind.
 3  Have put on black, and loving mourners be,
 3  But ’tis my heart that loves what they despise,
13  So true a fool is love that in your will,
 9  Rise, resty muse, my love’s sweet face survey
```

the last two of which are also included in the set denoted by the expression "love " and therefore excluded from the difference set.

For all of these operators, the set constraints are based on match points. Thus if a region expression is involved, the constraint on each member is evaluated in terms of the match point corresponding to its first character. For example,

*sonnet fby.100 "love "

returns regions in the set *sonnet for which the word “love” occurs within the first 100 characters; the proximity is measured from the start of the region, not the end. This semantics occasionally causes misunderstandings and errors on the part of some users and should therefore be reconsidered.

In an expression of the form

$$e_1 \text{ within } e_2$$

e_2 must be a region expression. For example,

("lov" – "love ") within *sonnet

returns the match points corresponding to suffixed uses of “love” occurring in sonnets (contrast with the use of “including” above), and

sonnetline = *line within *sonnet

returns the regions corresponding to lines within sonnets. As for the previous operators, when e_1 is a region expression, the constraint is based on the match points corresponding to the starts of the regions, but the result set is a (region) subset of e_1.

4.6. Augmentation

Because PAT provides restriction operations corresponding to set intersections “^” and set difference “–”, it also includes the binary operator “+” to indicate set union. As expected, when the two operands are match point expressions, the result set is a match point set including all members of both argument sets. For example, to find all sonnet lines that include the words “boy” or “youth”, one can write

BoyOrYouth = *sonnetline including ("boy " + "youth ")

which will return 18 lines. To find in which sonnets such lines appear (assuming prior definition of the region expression *title), a user could write

*title within (*sonnet including *BoyOrYouth)

yielding 16 regions containing the titles. To examine these titles together with the sonnet lines, one could then take the union of the two region sets:

*BoyOrYouth + %

(where % refers to the previous result), yielding the following text:

```
[[Sonnet]] 2
  3  Thy youth’s proud livery, so gazed on now,
[[Sonnet]] 7
  6  Resembling strong youth in his middle age,
[[Sonnet]] 11
  4  Thou mayst call thine when thou from youth convertest.
[[Sonnet]] 15
 10  Sets you most rich in youth before my sight,
 12  To change your day of youth to sullied night;
[[Sonnet]] 22
  2  So long as youth and thou are of one date;
[[Sonnet]] 37
  2  To see his active child do deeds of youth,
[[Sonnet]] 41
 10  And chide thy beauty and thy straying youth
[[Sonnet]] 54
 13  And so of you, beauteous and lovely youth,
[[Sonnet]] 60
  9  Time doth transfix the flourish set on youth,
[[Sonnet]] 73
 10  That on the ashes of his youth doth lie
[[Sonnet]] 96
  1  <T verse>Some say thy fault is youth, some wantonness;
  2  Some say thy grace is youth and gentle sport.
[[Sonnet]] 98
  3  Hath put a spirit of youth in everything,
[[Sonnet]] 108
  5  Nothing, sweet boy; but yet like prayers divine
[[Sonnet]] 110
  7  These blenches gave my heart another youth,
[[Sonnet]] 138
  3  That she might think me some untutored youth
[[Sonnet]] 153
 10  The boy for trial needs would touch my breast.
```

It should be noted that the semantics of PAT expressions are different from those of Boolean operators in traditional document retrieval systems. A naïve user wishing to find sonnets that include the word “love” as well as either “boy” or “youth” might write

*sonnet including (("boy " + "youth ") ^ "love")

and be surprised when PAT reports “no match”. The semantics of PAT expressions indicate that the correct way to pose this query is (*sonnet including ("boy " + "youth ")) including "love"

If one of the operands of the union operator is a match point expression and the other is a region expression, a simple union of the sets cannot be performed; PAT instead defines the result to be a match point set, using the match points corresponding to the starts of the argument regions in place of the regions themselves. Similarly, if both arguments are region expressions, but the regions in the corresponding sets overlap, PAT cannot form a simple union, since overlapping regions in one set are disallowed. Thus in this situation, PAT again defines the result to be a match point set, using the match points corresponding to the starts of the regions in place of the regions themselves. As a result,

(*speech including "to be or not to be") + (*line including "dying")

is a *region* expression denoting 58 regions of text, but

(*speech including "to be or not to be") + (*line including "death")

is a *match point* expression denoting 1030 match points in the text, since two lines containing the word "death" occur within the speech. Although these semantics are self-consistent, this feature in PAT is easily misunderstood and therefore changing it should be considered, perhaps by suitably defining the union of overlapping regions (as done, for example, in [Burkowski91]).

5. Conclusions

We have described the query capabilities included in the PAT text search system. We have divided PAT expressions into six classes and introduced the syntax and semantics of the expressions in the classes. We have shown that PAT indexing can be specified by productions as a view of PAT text seen as a character sequence. The matching of a phrase with a given string was also described by productions and text translation.

During the specification of PAT's search capabilities we have identified some problem areas which are interesting, not only from the point of view of PAT evaluation, but also in the evaluation of search capabilities in text search systems more generally. The indexing definition capabilities in PAT offer a means for application dependent indexing. However, the current indexing definition techniques are limited and designed for homogeneous text, whereas some document collections, such as multilingual collections, might include texts with heterogeneous indexing needs. Open Text Corporation supports a facility known as "Parallel PAT" that provides a single PAT interface to a collections of independently managed PAT texts, each using its own indexing definition.

The flexible definition of regions is an important and original feature in PAT. Together with the indexing definition capability, this feature supports application dependent handling of text. Regions may be defined both during the text installation phase and by the reader. Using PAT expressions the reader may search for either match points or regions, using a uniform syntax and semantics. In most cases the user may consider the restriction operations as if they were truly region operations, but occasionally the semantic implications are surprising. This is even more apparent in the augmentation operation. To support users familiar with more conventional document-based Boolean searching, a variety of front ends should be designed and implemented using PAT as a back-end search engine invoked through the program interface using PAT expressions. To date, experience with a restricted front end that provides simple look-ups in the *Oxford English Dictionary* and with a graphics-based front end to search an automobile engine manual, among others, shows that the separation of end-user convenience from search capabilities can be successful.

PAT with its region definition capabilities is designed for handling structured text. The structure of text is often expressed by tags, which can be used to define most regions. After the region definitions, the user might want to handle the text in the form where all tags are hidden. In the workstation environment, LECTOR allows a user to read regions without reading tags. For example, the reader of the *Oxford English Dictionary*, Shakespeare, or the Bible can read text displayed in a form similar to that in the printed volumes. However, the search is always applied to the tagged text, requiring users to be aware of the tags. Through the introduction of regular expressions in the definition of PAT indexing and search, more convenient search can be supported, wherein, for example, standard markup tags can be defined to be delimiters. The development of text search systems where the user performs *all* operations on text in various forms is an area of ongoing research.

Finally, the unique PAT indexing technique has made it possible to implement frequency search operations. However, the semantics of the current frequency operations is difficult to define, and they are limited in utility. Thus further studies for the development of these operations are needed.

Acknowledgements

Gaston Gonnet was the principal designer and implementer of PAT; several others, including Tim Snider and Byron Weber Becker, contributed substantially to the implementation. Financial support from the Natural Sciences and Engineering Research Council of Canada, under grants CRD0000862 and STR0045480, and from the Academy of Finland is gratefully acknowledged.

References

[Berg91] Berg, D.L., Gonnet, G.H., and Tompa, F.W., The New Oxford English Dictionary Project at the University of Waterloo, in *Computational Lexicology and Lexicography: Special Issue Dedicated to Bernard Quemada* (edited by A. Zampolli, L. Cignoni, and C. Peters), (series: Linguistica Computazionale, Vol. VII), Giardini Editori, Pisa, 1991, 29-44.

[Burkowski91] Burkowski, F.J., Textriever: A retrieval engine for multimedia databases, *Proc. of the Int. Conf. on Multimedia Information Systems*, Singapore 1991, 71-76.

[Croft84] Croft, W.B., Implementing a text storage and retrieval tool for the office, *Proc. of the First Int. Conf. on Office Automation*, IEEE Computer Society, 1984, 137-144.

[Fawcett89a] Fawcett, H., PAT3.3 User's Guide, UW Centre for the New OED and Text Research, University of Waterloo, 1989.

[Fawcett89b] Fawcett, H., PAT Installation Guide, UW Centre for the New OED and Text Research, University of Waterloo, 1989.

[Faloutsos85] Faloutsos, C., Signature files: Design and performance comparison of some signature extraction methods, *Proc. of ACM-SIGMOD 1985, Int. Conf. on Management of Data, SIGMOD Record 14*, 4 (Dec. 1985), 63-82.

[Goldfarb90] Goldfarb, C.F., *The SGML Handbook*, Oxford University Press, 1990.

[Gonnet83] Gonnet, G.H., Unstructured databases — or — very efficient text searching, *Proc. of ACM Principles of Database Systems*, 1983.

[Gonnet87a] Gonnet, G.H, Examples of PAT applied to the *Oxford English Dictionary*, Tech. Rept. OED-87-02, UW Centre for the New OED and Text Research, University of Waterloo, 1987.

[Gonnet87b] Gonnet, G.H. and Tompa, F.W., Mind your grammar: a new approach to modelling text, *Proc. of the 13th Int. Conf. on Very Large Data Bases*, 1987, 339-346.

[Gonnet91] Gonnet, G.H., Baeza-Yates, R.A., and Snider, T., Lexicographical indices for text: inverted files vs. PAT trees, Tech. Rept. OED-91-01, UW Centre for the New OED and Text Research, University of Waterloo, 1991.

[Gonnet92] Gonnet, G.H. and Snider, T., Mealy Machines, unpublished manuscript, UW Centre for the New OED and Text Research, University of Waterloo, 1992.

[Hollaar79] Hollaar, L.A., Text retrieval computers, *Computer 12*, 3 (March 1979), 40-50.

[Larson84] Larson, P.-A., A method for speeding up text retrieval, *ACM Data Base 15*, 2 (Winter 1984), 19-23.

[Lee85] Lee, D.L., The design and evaluation of a text-retrieval machine for large databases, Ph.D. Thesis, Computer Systems Research Institute, University of Toronto, 1985.

[Logan88] Logan, H.M. and Logan, G. An inquiry into inquiry systems: a discussion of some applications of data retrieval to the New OED database, *Proc. of the Fourth Conf. of the UW Centre for the New OED and Text Research*, “Information in Text,” Waterloo, ON, 26-28 October, 1988, 81-95.

[OUP88] Oxford University Press, *Complete Electronic Shakespeare*, 1988.

[Raymond90] Raymond, D.R. and Fawcett, H.J., Playing detective with full text searching software, *Proc. of SIGDOC ’90, SIGDOC Asterisk 14*, 4 (1990), 157-166.

[Raymond91] Raymond, D.R, Flexible Text Display with LECTOR, *Computer 25*,8 (August 1992).

[Salminen92] Salminen, A. and Tompa, F.W., Data modelling with grammars, unpublished manuscript, UW Centre for the New OED and Text Research, University of Waterloo, 1992.

[Salton81] Salton, G., A blueprint for automatic indexing, *ACM SIGIR Forum 16*, 2 (Fall 1981), 22-38.

[Salton83] Salton, G. and McGill, M.J., *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.

[Salton89] Salton, G., *Automatic Text Processing*, Addison-Wesley, Reading, 1989.

[Tague91] Tague, J., Salminen, A., and McClellan, C., A complete model for information retrieval systems, *Proc. of the 14th Int. ACM/SIGIR Conf. on Research and Development in Information Retrieval*, 1991, 14-20.

[Tompa92] Tompa, F.W., An Overview of Waterloo’s Database Software for the *OED*, *Proc. Symp. on Historical Dictionary Databases and Data Retrieval Requirements* (edited by T.R. Wooldrich) (Toronto, October 1991), in *CCH (Centre for Computing in the Humanities) Working Papers 2* (1992) 123-143.

APPENDIX. The syntax of PAT expressions

```
PAT_expression ::=
        [match_point_name ’=’] match_point_expr  |
        [region_name ’=’] region_expr  |
        match_point_expr_sequence .

match_point_expr ::=
        lexical_search  |
        position_search  |
        frequency_search  |
        match_point_restriction  |
        match_point_augmentation  |
        ’*’ match_point_set_name  |
        match_point_set_number  |
        ’%’  |
        ’(’ match_point_expr ’)’  |
        region_expr .

lexical_search ::=
        string  |
        string ’..’ string  .

position_search ::=
        ’[’ positive_integer ’]’  |
        ’shift’  ’.’ integer match_point_expr .

frequency_search ::=
        ’signif’ [  [’.’ positive_integer ] match_point_expr ]  |
        ’lrep’ [  [’.’ positive_integer ] match_point_expr ] .

match_point_expr_sequence ::=
        ’signif’  ’.’ negative_integer string .

match_point_restriction ::=
        match_point_expr ’^’ match_point_expr  |
        match_point_expr ’-’ match_point_expr  |
        match_point_expr [ ’not’ ] ’fby’ [ ’.’ positive_integer ] match_point_expr  |
        match_point_expr [ ’not’ ] ’near’ [ ’.’ positive_integer ] match_point_expr  |
        match_point_expr ’within’ region_expr .

match_point_augmentation ::=
        match_point_expr ’+’ match_point_expr .

region_expr ::=
        region_definition  |
        ’docs’ installed_region_definition |
        region_restriction  |
        region_augmentation  |
        ’*’ region_set_name  |
        region_set_number  |
        ’%’  |
        ’(’ region_expr ’)’ .

region_definition ::=
        ’docs’ match_point_expr ’..’ match_point_expr .

region_restriction ::=
        region_expr [ ’not’ ] ’including’ [ ’.’ positive_integer ] match_point_expr  |
        region_expr ’^’ match_point_expr  |
        region_expr ’-’ match_point_expr  |
        region_expr [ ’not’ ] ’fby’ [ ’.’ positive_integer ] match_point_expr  |
        region_expr [ ’not’ ] ’near’ [ ’.’ positive_integer ] match_point_expr  |
        region_expr [ ’not’ ] ’within’ region_expr .

region_augmentation ::=
        region_expr ’+’ region_expr .
```

N.B. An expression of the form

region_expr ’+’ region_expr

is a match point expression if some region specified by the first region expression overlaps some region specified by the other region expression.