Implementing Language-Dependent Lexicographic Orders in Scheme

Jean-Michel HUFFLEN
LIFC (EA CNRS 4157) — University of Franche-Comté
16, route de Gray — 25030 BESANÇON CEDEX — FRANCE
hufflen@lifc.univ-fcomte.fr

Abstract

The lexicographical order relations used within dictionaries are language-dependent, and we explain how we implemented such orders in Scheme. We show how our sorting orders are derived from the Unicode collation algorithm. Since the result of a Scheme function can be itself a function, we use generators of sorting orders. Specifying a sorting order for a new natural language has been made as easy as possible and can be done by a programmer who just has basic knowledge of Scheme. We also show how Scheme data structures allow our functions to be programmed efficiently.

Keywords Lexicographical order relations, collation algorithm, Unicode, MlBibTeX, Scheme.

1. Introduction

Sorting words belonging to a natural language, like in a dictionary, depends on this language. As part of the MlBibTeX project—for '**M**ulti**L**ingual BibTeX'—we developed functions implemented *collation*, that is, determining sorting orders of strings of characters. Let us recall that MlBibTeX aims to be a 'better' BibTeX [Patashnik, 1988]—the bibliography processor usually associated with the LaTeX word processor [Lamport, 1994]—especially about multilingual features. When BibTeX or MlBibTeX builds a 'References' section for a LaTeX document, a *bibliography style* is used to rule the layout of this section. Most of BibTeX styles sort this section's items w.r.t. the alphabetical orders of authors' names. The `SORT` function used within these styles [Mittelbach et al., 2004, Table 13.7] ignores accents and other diacritical signs, so in practice, it is suitable only for the English language. MlBibTeX allows bibliographical items of a document written in English (resp. French, German, ...) to be sorted according to the English (resp. French, German, ...) order.

MlBibTeX has been developed in Scheme, we explained the reasons of this choice and outlined its architecture in [Hufflen, 2005b]. Here we focus on the definitions of sorting orders as part of this framework. MlBibTeX's first version only deals with the European languages using the Latin alphabet. Our method is derived from an algorithm related to Unicode, an industry standard designed to allow text and symbols from all of the writing systems of the world to be consistently represented [2006]. Our method can be generalised to other alphabets. In addition, let us mention that if a document cites some works written using the Latin alphabet and some written using another alphabet (Arabic, Cyrillic, ...), they are usually itemised in two separate bibliography sections. So this limitation is not too restrictive within MlBibTeX's purpose.

The next section of this article is devoted to some examples in order to give some idea of this task's complexity. We briefly recall the principles of the Unicode collation algorithm [2006a] in Section 3 and explain how we adapted it in Scheme in Section 4. Section 5 discusses some points, gives some limitations of our implementation, and sketches possible future work.

2. Lexicographic orders and natural languages

The basic lexicographic order, well-known in Mathematics, can be defined by:

$$\begin{array}{rcl} [\,] & \leq & s_1 \\ [x_0|s_0] & \leq & [x_1|s_1] \iff x_0 < x_1 \vee (x_0 = x_1 \wedge s_0 \leq s_1) \end{array}$$

where x_0 and x_1 are characters, s_0 and s_1 strings of characters, and the notations '$[\,]$' and '$[x_0|s_0]$' are for the empty string and a non-empty string whose first character is x_0 and rest is s_0. This simple order relation may be used for English words, except that the differences in case—between uppercase and lowercase letters—are to be ignored in a first pass. Then if two words differ only be the case of a letter, an uppercase letter takes precedence over the corresponding lowercase one, according to a left-to-right order. In addition, let us notice that this relation can be implemented efficiently for unaccented letters since the ASCII[1] codes for letters follow the English alphabetical order.

deal with: an uppercase letter takes precedence over the corresponding downcase one if two words differ only by the case of a letter, and the order is left-to-right.

As shown by some non-limitative examples in Figure 1, this problem may be more complicated for other languages. We can observe some changes in the alphabetical order of unaccented letters: in the Estonian language, '`z`' is not the last letter of the alphabet, it is ranked between '`s`' and '`t`'. Accented letters may be treated as individual letters, like in Swedish, or interleaved with unaccented letters, like in the most common orders used in Germany. The same for ligatures: '`æ`' is viewed as a separate letter in Swedish, alphabeticised like '`ae`' in French. If accented letters are interleaved with unaccented ones, the latter take precedence when two words differ only by accents. In most cases, the order is left-to-right—that is true

[1] **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange.

Copyright © 2007 Jean-Michel Hufflen.
Proceedings of the 2007 Workshop on Scheme and Functional Programming
Université Laval Technical Report DIUL-RT-0701

- The Czech alphabet is: a < b < c < č < d < … < h < ch < i < … < r < ř < s < š < t < … z < ž.
- In Danish, accented letters are grouped at the end of the alphabet: a < … < z < æ < ø < å ∼ aa.
- The Estonian language does not use the same order for unaccented letters than usual Latin order; in addition, accented letters are either inserted into the alphabet or alphabeticised like the corresponding unaccented letter:

  a < … < s ∼ š < z ∼ ž < t < … < w < õ < ä < ö < ü < x < y

- Here are the accented letters in the French language: à ∼ â, ç, è ∼ é ∼ ê ∼ ë, î ∼ ï, ô, ù ∼ û ∼ ü, ÿ.
  When two words differ by an acccent, the unaccented letter takes precedence, but w.r.t. a right-to-left order: cote < c**ô**te < cot**é** < c**ôté**.
  The French language also use two ligatures: '`æ`' (resp. '`œ`'), alphabeticised like 'ae' (resp. 'oe').
- There are three accented letters in German—'`ä`', '`ö`', '`ü`'—and three lexicographic orders:
  - DIN[a]-1: a ∼ ä, o ∼ ö, u ∼ ü;
  - DIN-2: ae ∼ ä, oe ∼ ö, ue ∼ ue;
  - Austrian: a < ä < … < o < ö < … < u < ü < v < … < z.
- The Hungarian alphabet is:

  a ∼ á < b < c < cs < d < dz < dzs < e ∼ é < f < g < gy < h < i ∼ í < j < k < l < ly < m <
  n < ny < o ∼ ó < ö ∼ ő < p < … < s < sz < t < ty < u ∼ ú < ü ∼ ű < v < … < z < zs

- In Swedish, accented letters are grouped at the end of the alphabet: a < … < z < å < ä < ö.

'$a < b$' denotes that the words beginning with a are less than the words beginning with b, whereas '$a \sim b$' expresses that the letters a and b are interleaved, except that a takes precedence over b if two words differ only by these two letters.

[a] ***D**eutsche **I**nstitut für **N**ormung* (German Institute of normalisation).

Figure 1. Some order relations used in European languages.

for Italian and Portuguese—but not always: the French language uses a right-to-left order (cf. Fig. 1). In some languages, digraphs may sort as separate letters: for example, '`ch`' is ranked between '`h`' and '`i`' in Czech. The Hungarian language uses a trigraph, '`dzs`', as a separate letter. In addition, there are special rules for double digraphs in this language: for example, '`sz+sz`' is written '`ssz`' in this language, but the two successive digraphs should be restored before sorting: '`depresszió`' should be sorted as '`depreszszió`'. The same rule holds for the double trigraph '`ddzs`', for '`dzs+dzs`'. Other equivalences exist: in Danish, '`aa`' is equivalent to '`å`'[2].

So it clearly appears that there cannot be a universal order, encompassing all lexicographic orders. In addition, let us recall that we are interested in such order relations in order to sort bibliographical items w.r.t. authors' names. These names may be 'foreign' proper names if we consider the language used for the bibliography, that is, the language of the document. A very simple example is the use of English names within the bibliography of a document written in French. Such foreign names may include characters outside the alphabet of the document's language. As a consequence, an order relation for sorting the items of a bibliography should be able to deal with any letter belonging to a language written using the Latin alphabet, since such letters may appear in foreign names. A good choice is to associate accented foreign letters with the corresponding unaccented letter. If we consider the English language, this means that accented letters are interleaved with unaccented letters, but unaccented letters take precedence over the foreign letter if two words differ only by these two letters. So proceed most of implementations of lexicographic order relations. Let us notice that a foreign name may include additional letters whose association with a basic letter may be difficult: for example, the Icelandic '`þ`' letter.

3. The Unicode collation algorithm

Unicode provides a default algorithm [2006a] to sort all the strings build over its characters. It consists of a *multilevel* algorithm: each step sorts the elements left unsorted by the preceding step. Here are these levels:

L1	Base characters	`role` < `roles` < `rule`
L2	Accents	`role` < `rôle` < `roles`
L3	Case	`role` < `Role` < `rôle`
L4	Punctuation	`role` < `"role"` < `Role`
Ln	Tie-breaker	`role` < `ro□le` < `"role"`

The differences indicated by the underlined characters are swamped at stronger-level steps, for example, the difference between '`o`' and '`ô`' at Level 1. In the last example, '□' represents a format character, which is otherwise ignorable.

The first two steps are based on a *decomposition property* [2006b] for composite characters. For example, the '`ô`' letter, whose name and code point—given using hexadecimal numbers—are:

LATIN SMALL LETTER O WITH CIRCUMFLEX, U+00F4

can be decomposed into these two 'simpler' characters put together when a text is to be written:

LATIN SMALL LETTER O, U+006F
COMBINING CIRCUMFLEX ACCENT, U+0302

The *sort keys* used for each level are extracted from a Unicode collation element table, which defaults to the DUCET[3], given by

[2] The fact that a sequence of several letters may be equivalent to one has been pointed out in an example given in the proposal for the new standard of Scheme [cf. Sperber et al., 2007, § 1.2]:

```
(string-ci=? "Straße" "Strasse") ⟹ #t
```

because in German, the uppercase form of the '`ß`' letter is '`SS`'. On the contrary:

```
(string=? "Straße" "Strasse") ⟹ #f
```

As mentioned in [Flatt and Feeley, 2005], the implementation of the functions comparing strings can no longer be defined in terms of character-by-character comparisons, as they are in the present standard *R*[5]*RS* [1998].

[3] **D**efault Unicode **C**ollation **E**lement **T**able.

the file allkeys.txt, available at the Web site of Unicode[4]. These keys are *weight values*. For example, these values for the letters 'c', 'o', and the combining circumflex accent, given using hexadecimal values, are:

LATIN SMALL LETTER C	[.0FFE.0020.0002.0063]
LATIN SMALL LETTER O	[.113B.0020.0002.006F]
COMBINING CIRCUMFLEX ACCENT	[.0000.003C.0002.0302]

The first pass uses the first column's values as primary sort key, the second pass uses the second column's values as secondary sort key, and so on. '.0000' values are to being ignored. In our example, this means that a combining circumflex accent is to be ignored by the first pass of the algorithm. In fact, this algorithm now consists of a binary comparison between double bytes, until the two strings can be distinguished. If a right-to-left order is to be used for the second step, like in the French language, the lists of double bytes belonging to the second column should be reversed before applying comparisons. For example, these values are .0020.0020.003C.0020.0020 for the 'côte' word, '.0020.0020.0020.0020.0032 for the 'coté' word. Doing a double-byte-to-double-byte comparison allows us to conclude that côte $<$ coté, which is the case if the default collation algorithm is applied. If we consider the right-to-left order (e.g., for French), these lists of double-bytes are to be reversed, and the comparison between .0020.0020.003C.0020.0020 and .0032.0020.0020.0020.0020 implies that côte $>$ coté, which is correct for this language.

Weight values used as sort keys may be different from the code points ranking all the characters of Unicode and may be language-dependent. If a letter should be viewed as synonym of consecutive letters, for example, the 'æ' ligature in English, the table gives several 4-uples:

LATIN SMALL LETTER AE	[.0FD0.0020.0004.00E6]
	[.0000.0199.0004.00E6]
	[.1029.0020.001F.00E6]

'0FD0' and '1029' being the primary sort keys for the letters 'a' and 'e'. If a digraph should be viewed as a single letter, two consecutive characters are given a unique 4-uple of weight values. For example, 'ch' in traditional Spanish:

LATIN SMALL LETTER C, LATIN SMALL LETTER H; "ch"	[0707.0020.0002.00E6]

Given a particular language, some characters are given *variable* weight values for the Unicode collation algorithm [see 2006a, § 3.2.2]: they may be *ignorable*, *non-ignorable*, *shifted*, or *shift-trimmed*. 'Shifted' (resp. 'shift-trimmed') means that the variable-weighted characters are ranked before (resp. after) the others at the fourth step.

4. Our implementation

As part of MlBibTeX, our implementation of the collation algorithm aims to serve two purposes:

- end-users should be able to add a new order for a new language easily, provided that they can express how this order is built in an abstract way;
- resulting sorting orders for strings should be efficient, because they are used to sort list of bibliographical items, these lists being possibly big: such an operation requires many comparisons among strings. Of course, efficient sort algorithms are known for a long time, but the more efficient the comparisons among strings, the more efficient the list sort.

In addition, the general collation algorithm can be simplified in our case. Let us recall that we deal with natural languages written using the Latin alphabet.

- Only letters and whitespace characters are of interest for us: the punctuation signs can be dropped out, so the last step of the general algorithm is not needed. According to languages, some additional characters can be recognised—for example, the hyphen '-' character—they are either ignored or ranked between the space character and all the letters.
- As far as we know, the third step is the same for all the languages we deal with: an uppercase letter takes precedence over the corresponding downcase one if two words differ only by the case of a letter, and the order is left-to-right.

So, to derive a sorting order for strings from a generator, we have to provide four arguments.

- A list whose elements are *separator* characters, viewed less than any letter. It should begin by the space character and often this list contains only this character, in which case the <space-only variable can be used. This is not universal: for example, space characters are ignored when words are sorted in Hungarian (cf. the definition of the <hungarian? variable in Figure 2).
- An alphabet, given w.r.t. the increasing order, as a list of strings. If the 'classical' alphabet is used—unaccented letters of the Latin alphabet, sorted according to the usual order— just put the 'false' value (cf. the definition of the <english? variable).
- An association list for additional sequences of characters, each sequence being followed by a replacement and a weight value. That means that a decomposition is to be applied to these sequences.
- A function related to the sense of the second step: when the first is finished, weight values prepared for the second step appear in reverse order, so put reverse![5] if this second step's order is left-to-right, put identity—the identity function) for a right-to-left order. Cf. the use of these two values for <french? and <english?.

It should be noticed that only lowercase letters have to be specified, the equivalent relations among uppercase letters will be deduced.

Figure 2 shows how the order relations for the European languages described in Figure 1 are put into action. The result of our generator of order relations, <mk-order-relation, is a 2-argument function. Such a function takes two strings, s_0 and s_1, and returns #t if s_0 is strictly less than s_1 according to the order relation for the corresponding language, #f otherwise. Such an order relation is able to deal with strings containing 'foreign' letters, since there are default associations for the accented letters of all the European languages. For example, the Polish letter 'ł' is associated with the 'l' letter by default, the weight value allowing 'l' to take precedence over 'ł' at the second step of the sorting order if need be. Let us give two examples:

```
(<english? "coté" "côte") ⟹ #t
(<french? "coté" "côte")  ⟹ #f
```

In the first example, 'é' and 'ô' are foreign letters for the English language, and the order for the second step is left-to-right. In the second example, this order is right-to-left, as in French.

Our generator proceeds as follows.

[4] http://www.unicode.org/Public/UCA/latest/allkeys.txt

[5] Some Schemers could observe that this function does not belong to pure functional style, because it is potentially destructive [Shivers, 1999, see]. But it is more efficient than the reverse function and the weight information list is not shared with other lists.

```
(define <english (<mk-order-relation <space-only #f '() reverse!))

(define <austrian?
  (<mk-order-relation <space-only
                      '("a" "ä" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "ö" "p" "q" "r" "s"
                        "t" "u" "ü" "v" "w" "x" "y" "z")
                      '() reverse!))

(define <czech?
  (<mk-order-relation <space-only
                      '("a" "b" "c" "č" "d" "e" "f" "g" "h" "ch" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r"
                        "ř" "s" "š" "t" "u" "v" "w" "x" "y" "z" "ž")
                      '() reverse!))

(define <danish?
  (<mk-order-relation <space-only
                      (<push-default-alphabet '("æ" "ø" "å"))    ; Put these three letters at the end of the standard
                                                                 ; alphabet.
                      '(("aa" ("å" . 2))) reverse!))

(define <estonian?
  (<mk-order-relation <space-only
                      '("a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "z" "t"
                        "u" "v" "w" "õ" "ä" "ö" "ü" "x" "y")
                      '(("š" ("s" . 2)) ("ž" ("z" . 2))) reverse!))

(define <french?
  (<mk-order-relation <space-only #f
                      '(("à" ("a" . 2)) ("â" ("a" . 3)) ("è" ("e" . 2)) ("é" ("e" . 3)) ("ê" ("e" . 4))
                        ("ë" ("e" . 5)) ("î" ("i" . 2)) ("ï" ("i" . 3)) ("ö" ("o" . 2)) ("ù" ("u" . 2))
                        ("ü" ("u" . 3)) ("ÿ" ("y" . 2)))
                      identity))

(define <german-din-1?
  (<mk-order-relation <space-only #f '(("ä" ("a" . 2)) ("ö" ("o" . 2)) ("ü" ("u" . 2))) reverse!))

(define <german-din-2?
  (<mk-order-relation <space-only #f
                        '(("ä" ("a" . 2) ("e" . 2)) ("ö" ("o" . 2) ("e" . 2)) ("ü" ("u" . 2) ("e" . 2)))
                        reverse!))

(define <hungarian?
  (<mk-order-relation '()    ; In Hungarian, a whitespace character is irrelevant when words are sorted.
                      '("a" "b" "c" "cs" "d" "dz" "dzs" "e" "f" "g" "gy" "h" "i" "j" "k" "l" "ly" "m" "n"
                        "ny" "o" "ö" "p" "q" "r" "s" "sz" "t" "ty" "u" "ü" "v" "w" "x" "y" "z" "zs")
                      '(("á" ("a" . 2)) ("é" ("e" . 2)) ("ccs" ("cs" . 2) ("cs" . 2))
                        ("ddz" ("dz" . 2) ("dz" . 2)) ("ddzs" ("dzs" . 2) ("dzs" . 2))
                        ("ggy" ("gy" . 2) ("gy" . 2)) ("í" ("i" . 2)) ("lly" ("ly" . 2) ("ly" . 2))
                        ("nny" ("ny" . 2) ("ny" . 2)) ("ó" ("o" . 2)) ("õ" ("ö" . 2))
                        ("ssz" ("sz" . 2) ("sz" . 2)) ("tty" ("ty" . 2) ("ty" . 2)) ("ú" ("u" . 2))
                        ("ű" ("ü" . 2)))
                      reverse!))

(define <swedish?
  (<mk-order-relation <space-only (<push-default-alphabet '("æ" "ä" "ö")) '() reverse!))
```

Figure 2. Building order relations for some European languages.

- All the letters of the alphabet—the second argument of the `<mk-order-relation` function—and all the members—its third argument—supersede the default definitions.
- All the separator characters and letters of the alphabet are numbered and used as entries of a hash table, getting access to corresponding numbers. Such hash tables have been put into action by means of the functions of SRFI 69 [see Kalliokoski, 2005].
- All the separator characters, all the letters of the alphabet, all the associations' keys, and all the default definitions are used to build a *trie*[6]. If a single letter—or a digraph or trigraph—is recognised, this trie gets access to either the corresponding value of an association, or the `#t` value, in which case the recognised sequence belongs to the alphabet. The other characters are ignored.

[6] This word originates from the central letters of the word 're**trie**val' Fredkin [1960]. A **digital tree** is a tree for storing strings in which nodes are organised by substrings common to two or more strings, a **trie** is a particular case of a digital tree: there is only one node for every common prefix.

```
(define g0      ; Definition of a zero-argument function that will section the word 'szőlő' ('grape').
  (mk-hungarian-word-sectioner "szőlő"))
(g0) ⟹ ("sz" . 1)    ; The successive equivalent letters, digraphs, etc. of this word are returned in turn, with the corresponding
(g0) ⟹ ("ö" . 2)     ; weight value.
(g0) ⟹ ("l" . 1)
(g0) ⟹ ("ö" . 2)
(g0) ⟹ #f            ; The word is finished, so all the calls of this function will return the 'false' value, from now on.

(define g1 (mk-hungarian-word-sectioner "depresszió"))    ; Another example.
(g1) ⟹ ("d" . 1)
(g1) ⟹ ("e" . 1)
(g1) ⟹ ("p" . 1)
(g1) ⟹ ("r" . 1)
(g1) ⟹ ("e" . 1)
(g1) ⟹ ("sz" . 2)    ; Although the double digraph is written as 'ssz', it is replaced by two occurrences of the 'sz' digraph.
(g1) ⟹ ("sz" . 2)
(g1) ⟹ ("i" . 1)
(g1) ⟹ ("o" . 2)
(g1) ⟹ #f
```

Figure 3. Sectioning Hungarian words.

- Our tries are implemented by balanced ternary search trees. 'Balanced' means that for each non-empty subtree, the numbers of elements of the left, middle, and right branches do not differ from more than 1. To get this trie, we sort the alphabet according to the Latin 1 encoding, so our hash table is used to retain information about precedence within this alphabet. On another point, the resulting trie allows us to efficiently implement word sectioning into letters, digraphs, etc.

The weight value associated with each string belonging to the alphabet is 1. So you can use weight values greater than or equal to 2 for accented letters belonging to the language. In comparison with the Unicode collation algorithm, we skip combining characters resulting from the decomposition procedure and only give their weight value. For example, the accents allowed in French over the 'a' letter are the grave and circumflex accents ('à' and 'â'), but not the acute one ('á'). The allowed accents are given 2 and 3 as weight values, they come before the default value for the acute accent over this letter. On the contrary, we do not specify that the 'æ' ligature is alphabeticised like 'ae' because it is the default definition for this character.

We show how strings are sectioned in Figure 3. When an order relation is applied to two strings, we build sectioner functions for these two strings. We section a string as few times as possible and stop as soon as we can conclude. The example given is a sectioner for Hungarian words, possibly using digraphs and double digraphs (cf. § 2). This example also includes words containing accented letters interleaved with unaccented ones ('õ' and 'ó', interleaved with 'ö' and 'o').

5. Discussion

As we explain in [Hufflen, 2005b], we decided that MlBibTeX should be used with several Scheme interpreters, in order to enforce this program's portability. There is a proposal to make Scheme Unicode-compliant [Flatt and Feeley, 2005]; that is planned for the future standard [Sperber et al., 2007, §§ 1.1 & 1.2]; but only a little support for Unicode is provided now, rudimentary about possible encodings [Serrano, 2006, p. 35], MIT Scheme [Hanson et al., 2002, § 5.7], PLT Scheme [Flatt, 2007, § 1.2.1]. In fact, MlBibTeX's basic encoding is Latin 1, and European characters outside it are obtained by means of a workaround: the LaTeX commands to produce them [see Mittelbach et al., 2004, Table 7.33]. For example, the Hungarian word 'szõlõ' (cf. Figure 3) should be typed by '"sz\\H{o}l\\H{o}"'. As part of MlBibTeX, this is not a real drawback since end-users get used to type accented letters by means of LaTeX commands within their bibliography database files[7]. In addition, it will easy to adapt our functions when Scheme becomes Unicode-compliant.

Another limitation is given by exceptions. For example, let us consider the following Hungarian person names: Kótz < Kótyi. They follow the Hungarian rules for sorting names (cf. Figure 1). But we have Kóty < Kótz because of etymological reasons, superseding the usual decomposition of words. Probably a dictionary of exceptions would be the best way to solve this problem, but we have not implemented it yet.

In MlBibTeX, we chose to allow the introduction of a new sorting order by means of only one definition. This allows a global view of this new order relation and makes easier some coherence tests among the information about this relation. A different approach has been followed by xindy [Kehr, 1998], a multilingual index processor associated with LaTeX, and written using COMMON LISP [Steele et al., 1990]. The specification of an order relation is different because it is done step by step. There are forms:

```
define-alphabet     define-letter-group
merge-rule          sort-rule
```

to specify an alphabet, a letter group (digraph, trigraph, etc.), and the replacement of a pattern. If a sort procedure is quite close to the standard way used in English, it is probably easier to use xindy's forms, because only small changes have to be expressed. On the contrary, MlBibTeX allows users to define a new order relation by applying only one function, encompassing all the aspects of this new order relation.

Even if we have adapted the Unicode collation algorithm to our requirements for MlBibTeX, we think that we could easily implement an efficient version of the whole algorithm—not limited to languages written using the Latin alphabet—by means of the same structures: tries and hash tables. A possible improvement

[7] When a bibliography data base file is parsed by MlBibTeX, the LaTeX commands that result in characters belonging to the Latin 1 encoding are expanded, the others are left unchanged. So parsing 'c\^{o}t\'{e}' within the value associated with a BibTeX field results in 'côté', whereas parsing 'sz\H{o}l\H{o}' results in the Scheme string 'sz\\H{o}l\\H{o}'.

could be the extraction of sort keys from the files available at the Web site of Unicode: it would just require an *ad hoc* parser.

Finally, let us remark that we used continuation-based functions to put into action the sectionning of a string into letters, including the case of digraphs. A more concise specification could have been given by using a lazy functional programming language based on the call by need—e.g., Haskell [Peyton Jones, 2003]—getting the next letter is done only if need be.

6. Conclusion

The availability of sorting orders depending on natural languages is planned in XSLT[8] [1999], the language of transformations used for XML[9] documents. XSLT provides an `xsl:sort` element that can sort strings according to the rules of a natural language [see W3C, 1999, § 10]. But in practice, most of XSLT processors implement this feature only partially, and the way to design new order relations, if need be, is unspecified by the W3C[10] recommendation as well as the documentation of these processors. Designers of bibliography styles for MlBibTeX can use order relations by means of an element analogous to XSLT's [see Hufflen, 2005a]. But as shown in § 4, only basic knowledge of Scheme is needed for people interested in enlarging MlBibTeX by new relations.

When MlBibTeX's first experimental versions were launched, there was only an order relation implementing the default collation algorithm roughly[11]. In parallel, we developed our order relation generator. That aimed to ask some people for tests about their own language. We do not forget that natural languages are an open domain, that is, it is difficult to establish general rules that may fail on particular cases since the features of natural languages are very diverse. So we consider our present work as a first version subject to changes when we explore other languages or get criticisms from end-users. But until now, feedback has been good, and as a consequence, our order relation generator has been integrated into MlBibTeX's first public version.

Acknowledgements

Many testers of MlBibTeX helped me fix some errors concerning various languages: thank to them. I am also grateful to the referres, who suggested me some improvements.

References

Matthew Flatt. PLT *MzScheme: Language Manual. Version 370*. `http://download.plt-scheme.org/doc/370/pdf/mzscheme.pdf`, May 2007.

Matthew Flatt and Marc Feeley. *R6RS Unicode Data*. `http://srfi.schemers.org/srfi-75/`, July 2005.

Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9): 490–499, September 1960.

Chris Hanson, the MIT Scheme team, et al. MIT *Scheme Reference Manual*. Massachusetts Institute of Technology, 1.96 edition, March 2002.

Jean-Michel Hufflen. Bibliography styles easier with MlBibTeX. In *Proc. EuroTeX 2005*, pages 179–192, Pont-à Mousson, France, March 2005a.

Jean-Michel Hufflen. Implementing a bibliography processor in Scheme. In J. Michael Ashley and Michel Sperber, editors, *Proc. of the 6th Workshop on Scheme and Functional Programming*, volume 619 of *Indiana University Computer Science Department*, pages 77–87, Tallinn, September 2005b.

Panu Kalliokoski. *Basic Hash Tables*. `http://srfi.schemers.org/srfi-69/`, September 2005.

Roger Kehr. *xindy Manual*. `http://www.xindy.org/doc/manual.html`, February 1998.

Richard Kelsey, William D. Clinger, Jonathan A. Rees, Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman, and Mitchell Wand. Revised[5] report on the algorithmic language Scheme. HOSC, 11(1):7–105, August 1998.

Leslie Lamport. *LaTeX: A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig, and Joachim Schrod. *The LaTeX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2 edition, August 2004.

Oren Patashnik. *BibTeXing*. Part of the BibTeX distribution, February 1988.

Simon Peyton Jones, editor. *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press, April 2003.

Manuel Serrano. *Bigloo. A Practical Scheme Compiler. User Manual for Version 2.9a*, December 2006.

Olin Shivers. *List Library*. `http://srfi.schemers.org/srfi-1/`, October 1999.

Michael Sperber, William Clinger, R. Kent Dybvig, Matthew Flatt, Anton van Straaten, Richard Kelsey, and Jonathan Rees. *Revised[5.97] Report on the Algorithmic Language Scheme — Standard Libraries*. `hhttp://www.r6rs.org`, June 2007.

Guy Lewis Steele, Jr., Scott E. Fahlman, Richard P. Gabriel, David A. Moon, Daniel L. Weinreb, Daniel Gureasko Bobrow, Linda G. DeMichiel, Sonya E. Keene, Gregor Kiczales, Crispin Perdue, Kent M. Pitman, Richard Waters, and Jon L White. COMMON LISP. *The Language. Second Edition*. Digital Press, 1990.

Unicode Collation Algorithm. The UNICODE CONSORTIUM, `http://unicode.org/reports/tr10/`, July 2006a. Unicode Technical Standard #10.

Unicode Normalization Forms. The UNICODE CONSORTIUM, `http://unicode.org/reports/tr15/`, October 2006b. Unicode Standard Annex #15.

The UNICODE CONSORTIUM. *The Unicode Standard Version 5.0*. Addison-Wesley, November 2006.

W3C. XSL *Transformations (*XSLT*). Version 1.0*. `http://www.w3.org/TR/1999/REC-xslt-19991116`, November 1999. W3C Recommendation. Edited by James Clark.

[8] e**X**tensible **L**anguage **S**tylesheet **T**ransformations.

[9] e**X**tensible **M**arkup **L**anguage.

[10] **W**orld **W**ide **W**eb **C**onsortium.

[11] That was the case for the version described in [Hufflen, 2005b].