Multidimensional text

John Plaice
School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
plaice (at) cse dot unsw dot edu dot au

Blanca Mancilla
School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
mancilla (at) cse dot unsw dot edu dot au

Chris Rowley
Mathematics and Statistics
The Open University in London
1-11 Hawley Crescent, Camden Town
London NW1 8NP, UK
c.a.rowley (at) open dot ac dot uk

Abstract

The standard model of text, based on XML and Unicode, assumes that documents are trees whose leaves are sequences of encoded characters. This model is too restrictive, making unnecessary assumptions about documents, text and the processing applied to these.

We propose instead that text and documents be encoded as tuples, i.e., sets of dimension-value pairs. Those dimensions used for content are split into the property dimensions, which are named by elements of an unstructured set, and the indexing dimensions, which form a structured set, often enumerated.

Using our approach allows natural solutions for a wide range of encoding requirements: encoding of documents at multiple levels of abstraction (glyph, character, word, stem-declension pair, compound word, etc.); encoding by linear, tree, DAG and multidimensional structures. Our model is upwardly compatible with existing approaches.

1 Introduction

In this article, we present a new model for manipulating documents in which every structure is encoded as a *tuple*, a set of dimension-value pairs. The simpler elements are ordinary tuples encoding basic information, while more complex elements encode mappings from structured index sets towards simpler elements.

The advantage of this new model is that it allows documents to be encoded in many different ways, taking into account logical structure, visual structure and linguistic analysis. Furthermore, the proposed model is upwardly compatible with existing practice.

This model is one result of a research project into the nature of text initiated by authors Plaice and Rowley [5, 6]. The current standard computer model of documents assumes that the structure of a text is a tree — normally encoded using XML — whose leaves are sequences of Unicode characters and where the intermediate nodes contain sets of attribute-value pairs to define properties. The conclusion of the aforementioned research was that the current model makes the assumption that text is simply something to be shuffled around, possibly chopped up for rendering purposes, but that it has no structure of its own; furthermore, the origin of the view of a document as some form of stream of bytes can be traced directly back to the near-simultaneous invention of the typewriter and the telegraph. The relatively recent distinction between "character" and "glyph", where character is an abstraction from glyph, still makes the assumption that the visual presentation of text is the key, despite the fact that, for example, English, Amharic and Chinese are normally encoded at, respectively, the letter, syllable and (sub)word levels.

As large collections of documents are brought together, so that they can be searched, it is clear that this "standard" model leaves something to be desired. Searching through a multilingual database of texts requires substantial linguistic support, and here it becomes essential that the different languages be handled at similar levels. Moreover, in linguistics, texts are often encoded using parse trees and attribute-valued matrices, often DAGs (directed acyclic graphs) or digraphs (directed graphs).

As for the structure of documents containing text, the tree structure itself is not always appropriate. Although it is true that a two-dimensional table can be encoded as a tree, this is only true by imposing ordering on its two dimensions (either by rows or by columns); they are not naturally encoded by a purely hierarchical structure but by its antithesis, a completely crossed structure. Moreover, two-dimensional tables are often used to visualize multidimensional data, whose encoding truly requires a multidimensional data structure such as those investigated in a different context by one of the authors [1], not just to capture and simplify the semantics, but to ensure an efficient and tractable storage mechanism.

Importantly, what distinguishes the electronic document from all previous forms of document is that it is recreated every time that it is read, listened to, studied or processed. As a result, with a slight change of parameters, it can be recreated differently from any previous occasion. The standard model completely breaks down for these kinds of situations: it must resort to programming the document and, in so doing, loses the possibility of having the document being properly indexed for searchability. (See [4] for further discussion.)

The tuple structure that we are proposing allows us, as shall be shown below, to define a number of different structures, including ordered streams and trees, DAGs and multidimensional structures. For example, a sentence in a document can be encoded as a sequence of characters, as a parse tree with words as leaves, or as a "feature structure" from linguistics known as an attribute-valued matrix. Indeed, it could easily be encoded as all of the above, with appropriate link structures connecting the components.

The structure of this paper is as follows. We present a brief analysis of, and show the current limitations of, various existing models for the encoding of texts and documents in §2. We follow with a presentation of our new model (§3) using an extended example. We then (§4) describe some features of its use to declined word-stem sequences, parse trees, hierarchical document structure, attribute-value matrices and tables. We conclude with a discussion of future work.

2 Existing models

We examine in this section three approaches to dealing with text. Although not exhaustive, it does provide us with indications of where a more complete model should be heading.

2.1 XML documents

An XML document is typically encoded as a tree. In some sense, this is all there is to say, but a proper understanding of XML requires examining not just the obvious tree structure, but also the structures of the nodes and the leaves in these trees.

For each element in an XML document, there is a possibly empty list of attribute-value pairs, and a possibly empty list of child elements. The leaves of the tree consist of PCDATA (Parsed Character Data) or CDATA (Character Data), in both cases sequences of characters from the Unicode or some other character set, with PCDATA being parsed by the XML parser.

Therefore XML, often presented as a simple encoding, actually requires four data structures to describe a document:

- the tree;
- the list of elements;
- the attribute-value list; and
- the character sequence.

In addition, there are arbitrary restrictions on attributes which limit their usability: attributes cannot contain multiple values, nor can they contain tree structures.

Our model includes such XML tree-based documents but it can also handle non-hierarchical structures that do not necessarily have any natural XML encoding.

2.2 Linguistics AVMs

In linguistics, it is common to model written language using structures consisting of a tree and an associated attribute-value structure [3, p. 16]. For example, here is the parse tree for the sentence "Mary seems to sleep.":

```
[ S$_1$: [ NP$_2$: Mary
        VP$_1$: [ V$_1$:  seems
              VP$_3$: [ Aux$_3$: to
                     VP$_3$:  [ V$_3$: sleep ]]]]]
```

The associated structure, called an attribute-value matrix (AVM), is presented below.

Note: The subscripts in the parse tree correspond to the nodes in the AVM. Node 2 appears twice in the AVM, being the subject of both node 1 and node 3.

$$\boxed{1}\begin{bmatrix} \text{subj} & \boxed{2}\begin{bmatrix} \text{agr} & \begin{bmatrix} \text{pers} & \text{3rd} \\ \text{num} & \text{sg} \end{bmatrix} \\ \text{pred} & \text{mary} \end{bmatrix} \\ \text{comp} & \boxed{3}\begin{bmatrix} \text{subj} & \boxed{2} \\ \text{pred} & \text{sleep} \\ \text{tense} & \text{none} \end{bmatrix} \\ \text{pred} & \text{seem} \\ \text{tense} & \text{pres} \end{bmatrix}$$

Because of such shared structures, AVMs are often considered to be DAGs. However, it is possible to have cyclical structures in an AVM: Consider the noun phrase “the man that I saw”, whose parse tree is here:

```
[ NP$_1$: [ Det: the
         N’:  [ N:  man
               S’$_2$: [ Comp: that
                     S:    [ NP: I
                             VP: [ V: saw ]]]]]]
```

In the following AVM, a cyclical structure is needed to describe the “filler-gap” dependency in the relative clause [3, p. 19]:

$$\boxed{1}\begin{bmatrix} \text{def} & + \\ \text{pred} & \text{man} \\ \text{comp} & \boxed{2}\begin{bmatrix} \text{pred} & \text{saw} \\ \text{subj} & \begin{bmatrix} \text{pred} & \text{pro} \end{bmatrix} \\ \text{obj} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Our model naturally encodes both such parse trees and these AVMs.

2.3 Tables

Xinxin Wang and Derick Wood [7] developed a general model for tables, in which a table is a mapping from a multidimensional coordinate set to sets of contiguous cells. For them, the two-dimensional format commonly used to present a table is not the internal format. Here is an example of the use of a 3-dimensional coordinate set from their paper:

	Assignments			Examinations		Grade
	Ass1	Ass2	Ass3	Midterm	Final	
1991						
Winter	85	80		60	75	75
Spring	80	65	75	60	70	70
Fall	80	85		55	80	75
1992						
Winter	85	80	70		75	75
Spring	80	80			75	75
Fall	75	70	65	60	80	70

It appears that, in 1991 alone, Assignment 3 was identical across the three terms but for 1992, we can see no such simple explanation of the larger box since it seems to amalgamate this assignment with an examination.

In this example and using their notation, the three dimensions and their value sets are as follows:

$$\begin{aligned} \text{Year} &= \{1991, 1992\} \\ \text{Term} &= \{\text{Winter}, \text{Spring}, \text{Fall}\} \\ \text{Marks} &= \left\{ \begin{array}{l} \text{Assignments} \cdot \text{Ass1}, \\ \text{Assignments} \cdot \text{Ass2}, \\ \text{Assignments} \cdot \text{Ass3}, \\ \text{Examinations} \cdot \text{Midterm}, \\ \text{Examinations} \cdot \text{Final}, \\ \text{Grade} \end{array} \right\} \end{aligned}$$

A small change of syntax would transform this into an example of our model.

3 The new model

In this section we present an extended descriptive illustration of the model, using an example, rather than a detailed formal model.

There is only one basic structure in this model: the *tuple*, which is defined as a set of *dimension-value* pairs. This is not a new data structure, and it has many different names in different formalisms: *dictionary* in PostScript, *hash-array* in Perl, *map* in C++, *association list* in Haskell, *attribute-value list* in XML, and *tuple* in Standard ML and Linda.

We begin by proposing a possible encoding for the sentence “LaTeX 2ε is neat.”

```
[ type: sentence
  numberWord: 3
  endPunctuation: [ type: unichar, code: 002E ]
  0: [ type: TeXlogo
       TeXcode: "\LaTeX\thinspace2\lower1pt
                 \hbox{\small$\varepsilon$}}"
       SimpleForm: [type: digiletters
                    unicharstring: "LaTeX2e" ]]
  1: [ type: word
       numberChar: 2
       0: [ type: unichar, code: 0069 ]
```

```
      1: [ type: unichar, code: 0073 ] ]
  2: [ type: word
      numberChar: 4
      2: [ type: unichar, code: 0061 ]
      1: [ type: unichar, code: 0065 ]
      0: [ type: unichar, code: 006E ]
      3: [ type: unichar, code: 0074 ] ] ]
```

In this example, there are 12 tuples:

- 1 sentence,
- 1 TEX logo,
- 1 character string consisting of a "word and digits",
- 2 words, and
- 7 Unicode characters.

The different dimensions in the example play different rôles. In fact, there are three categories of dimensions:

- The *typing dimensions* allow one to distinguish the different kinds of tuple. Thus all tuples must have a typing dimension. In the example, a special dimension, called `type`, provides the information. This is the standard solution, although we do not exclude the use of further special dimensions, say `subtype` or `version`, for further clarification.
- The *property dimensions* are used to store any type of information about the tuple. In the example, these dimensions are:
  - `numberWord` and `endPunctuation`, for tuples of type `sentence`;
  - `TeXcode` and `SimpleForm`, for the `TeXlogo` tuples;
  - `unicharstring` for `digiletters` tuples;[1]
  - `numberChar` for `word` tuples;
  - `code` for `unichar` tuples.
- The *indexing dimensions* are used to access the substructures of a tuple (the content of the tuple) via an indexing mechanism or structure. The set of all the indexing dimensions available to a given type of tuple can be very large, conceptually infinite, and will often carry a complex structure.

  In the following sections we will extensively develop examples of how the structure of these indexing dimensions can be used to encode complex systems. In the example, both the `sentence` and `word` tuples use the natural numbers ($\mathbb{N}$) to enumerate their content; thus the indexing dimensions available are the natural numbers, whose structure is the unique countable well-ordering.

[1] This could be replaced by a more formal tuple of type `digiword` that is like `word`: being an indexed collection of `unichar` tuples that can also contain digits.

More precisely, the property dimensions form an unstructured set, whilst the indexing dimensions form a subset of a structured set. We can, for example, write a tuple of type `typespec` that defines the sets needed for the dimensions in `sentence` tuples:

```
[ type:      typespec
  tupletype: sentence
  typedim:   {type}
  propdim:   {numberWord, endPunctuation}
  indexdim:  N
]
```

Note that, although the set of possible index dimensions $\mathbb{N}$ is the infinite set of all natural numbers, any given tuple will use only a finite number of numbers as indices. Also, it is important that, for example, the order of the characters in a word is defined by the ordering of their indexing dimensions as natural numbers, not the order in which they appear in the written form of the "`word` tuple". In this example we write the characters in the order of the numbers of their Unicode slots (such an ordering may be very efficient for certain applications).

In all of these sets, of dimensions and possible values, both equality and membership must be computable. In all usable examples, of course, equality-testing and other necessary set operations should be at worst of polynomial time complexity for reasonably sized sets.

The tuples used as values may be of any type, thus, for example, they could include strings, files, programs, and so on.

There can be other interesting interplays between dimensions and values. Consider, for example, the Unicode character tuple:

```
[ type: unichar, code: 002E ]
```

Here the value `002E` is a value used to index the Unicode character database and, as Bella [2] has shown, that database can be understood as a single tuple indexed by the natural numbers whose entries are themselves tuples containing various kinds of information about each Unicode character.

The tuples used in this model are *conceptual*: they can be implemented — both as data structures in running programs and as sequentialised files on disk — in many different ways. Depending on the exact applications, algorithms and programming languages and environments, some solutions are more appropriate than others.

4 Examples

We hope it is now clear that our tuple structure can encode any sort of simple record or entry. In this section, we indicate with a few more examples how the tuple structure can encode different kinds of data structure.

As was described in the previous section, there are typing, property and indexing dimensions in a tuple. If we restrict ourselves to the indexing dimensions, then the tuple can be considered to be a (partial) function from an index set (the structured set of indexing dimensions) to a set of values, which are in general themselves tuples, interpreting a singleton value as a type of tuple `singleton` and indexing set of size 1 {0}. If, in the sense of datatypes, these singleton values are strongly typed, then these `singleton` values will need a `datatype` property dimension.

- *Declined word-stem sequences*: To enhance document search or to effect grammatical analysis, it is common to *stem* words, separating the stem and the prefixes or suffixes of the words. We would then end up with entries such as:

```
[ type:          verb
  language:      English
  stem:          carry
  mood:          indicative
  tense:         present perfect
  voice:         active
  person:        3rd
  number:        singular
  unicharstring: "has carried" ]
```

  or

```
[ type:          noun
  language:      French
  stem:          pomme
  gender:        feminine
  number:        plural
  unicharstring: "pommes" ]
```

  Should the system not be able to parse such a word/phrase, then it will store only its `unicharstring` until it is appropriately updated.

- *Parse trees*: The result of the natural language parsing of a sentence is a richer text structure that is often encoded in a tree structure.

  Below is a possible parse tree for the sentence "Mary seems to sleep.", first presented in the section on AVMs (§2):

```
[ type: sentence
  NP:   Mary
  VP:   [ type: verbPhrase
          V:    seems
          VP:   [ type: verbPhrase
                  Aux:  to
                  VP:   [ type: verb
                          V:    sleep ]]]]
```

- *Hierarchical document structure*: The traditional book with frontmatter, chapters, sections and subsections is a typical example of a document tree. This hierarchy can be extended downwards to paragraphs, sentences, phrases, words and characters.

- *Attribute-value matrices*: As explained in §2, these contain shared structures. Below is the AVM for the last example sentence.

```
[ type: AVMentries
  numberEntries: 3
  1: [ type:  AVM
       subj:  2
       comp:  3
       pred:  seem
       tense: pres ]
  2: [ type:  AVM
       agr:   [ type: AVM
                pers: 3rd
                num:  sg ]
       pred:  mary ]
  3: [ type:  AVM
       subj:  2
       pred:  sleep
       tense: none ] ]
```

- *Tables*: The encoding of tables by Wang and Wood is our final example of the model. Here is a possible encoding, where the dimensions are allowed to range over a set of possible values in order to encode the boxed values.

```
[ type: WWtable
  [ type: WWindex
    year: 1991
    term: Winter
    mark: Assignments.Ass1
  ] : 85
  ...
  [ type: WWindex
    year: 1991
    term: Winter..Fall
    mark: Assignments.Ass3
  ] : 75
  ...
  [ type: WWindex
    year: 1992
    term: Winter..Spring
    mark: Assignments.Ass3..Midterm
  ] : 70
  ...
  [ type: WWindex
    year: 1992
    term: Fall
    mark: Grade
  ] : 70
]
```

5 Conclusions

The model introduced in this paper, based on general tuples, is as simple as possible whilst being flexible enough to encode a large range of different approaches to the study and manipulation of text in all of its forms, as well as to support the encoding and linguistic tools such as language dictionaries.

However, the real power of the model is the idea of the index, built right into the model, which provides access to any piece of data, thereby supporting the specification of algorithms and hypertext-like links between document components.

The infinite index sets correspond to iterators into containers, as used, for example, in the C++ STL (Standard Template Library) for generic programming. Furthermore, using these sets it is even possible that certain tuples are, conceptually, countable infinite (like lists in a functional programming language), with the components being evaluated in a lazy manner, on-demand.

In future papers we shall show how this unifying data model makes it easy to combine in a single system myriad ways of editing, storing, manipulating and presenting text and to manipulate all of these together.

References

[1] R. A. Bailey, Cheryl E. Praeger, C. A. Rowley, and T. P. Speed. Generalized wreath products of permutation groups. *Proc. Lond. Math. Soc.*, s3–47(1):69–82, July 1983.

[2] Gábor Bella. *Modélisation du texte numérique multilingue: vers des modèles généraux et extensibles fondés sur le concept de textème.* PhD thesis, Télécom Bretagne, Brest, France, 2008.

[3] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar.* Center for the Study of Language and Information, Stanford University, 1988.

[4] Blanca Mancilla and John Plaice. Possible worlds versioning. *Mathematics in Computer Science*, 2008. In press.

[5] John Plaice and Chris Rowley. Characters are not simply names, nor documents trees. In *Glyph and Typesetting Workshop*, East Asian Center for Informatics in Humanities, Kyoto University, 2003. `http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/009-plaice.pdf`.

[6] Chris Rowley and John Plaice. New directions in document formatting: What is text? In *Glyph and Typesetting Workshop*, East Asian Center for Informatics in Humanities, Kyoto University, 2003. `http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/001-rowley.pdf`.

[7] Xinxin Wang and Derick Wood. A conceptual model for tables. In Ethan V. Munson, Charles K. Nicholas, and Derick Wood, editors, *PODDP*, volume 1481 of *Lecture Notes in Computer Science*, pages 10–23. Springer, 1998.