An Empirical Modelling Approach to Text Processing

0821769

Abstract

This paper discusses the application of Empirical Modelling(EM) principles and techniques to text processing. The paper then presents a specification for such an environment. The corresponding model is a partial implementation of this in EDEN(Engine for Definitive Notations). Finally, the model is evaluated and further improvements are suggested.

1 Introduction and Discussion of Problem

Text processing remains one of the most widespread uses of computers in both professional and personal work. As such, there is a wide range of text processors available, with variations depending on the ultimate aim of the user.

The aspect of Empirical Modelling(EM) considered by the paper is the construction of models focused around dependencies between observables in an environment.

In this paper, I will look at how an approach based around Empirical Modelling(EM) could be used to create a syntax or piece of software that would allow for a both powerful and intuitive form of text processing. In addition to this, I would also like to consider how the software could be used to develop models for a piece of EM software, such as EDEN.

In EM, much focus is placed on 'Modelling with Definitive Scripts', necessarily leading to the need to process text in some way. This form of development can lead to problems and habits encountered to conventional programming. The user may identify dependency within the text, but be unable to act upon it without the use of specialized editors or macros. This may lead to tedious 'copy paste' behaviour or extensive editing.

The user may also want to use unrefined content in the text editor without committing it to the final document. An EM framework would allow for incomplete or redundant text to be marked as such and omitted from the final document.

N. Pope (2010) discuss how EM principles lend themselves to software development. This paper aims to embody some of these principles in a tool that would allow the user to process text in the more intuitive manner provided by EM.

2 Formulation of Approach

Fundamentally, I would the EM text framework developed will consist of two parts; the text itself and the 'script' necessary to describe the relationships and dependency in the text.

This approach allows the user to take conventional text and add structure to it, by building up a complementary script and perhaps adding 'meta characters' to the text to identify the observables as the user wishes to describe them.

The syntax must also be able to address the most common forms of variation in this context: corresponding numbers to a list of names or letters, generating desired permutations and combinations of numbers and so on.

The EM project Lines, Beynon (1991), provides an example of where this could be applied:

```
# d1213 is 1 if the crossing index of
    lines 1 and 2 differs from that of
# lines 1 and 3  and line 4 doesn't
    cross line 1 between its points
# of intersection with lines 2 and 3
d1334 = if !(x13==x34) && ((r23-r13)*(
    r23-r34)>0) then 1 else 0
d2334 = if !(x23==x34) && ((r13-r23)*(
    r13-r34)>0) then 1 else 0
d1224 = if !(x12==x24) && ((r23-r12)*(
    r23-r24)>0) then 1 else 0
d1223 = if !(x12==x23) && ((r24-r12)*(
    r24-r23)>0) then 1 else 0
d1323 = if !(x13==x23) && ((r34-r13)*(
    r34-r23)>0) then 1 else 0
```

```
d1424 = if !(x14==x24) && ((r34-r14)*(
    r34-r24)>0) then 1 else 0
d2434 = if !(x24==x34) && ((r14-r24)*(
    r14-r34)>0) then 1 else 0
d1434 = if !(x14==x34) && ((r24-r14)*(
    r24-r34)>0) then 1 else 0
d1213 = if !(x12==x13) && ((r14-r12)*(
    r14-r13)>0) then 1 else 0
d1214 = if !(x12==x14) && ((r13-r12)*(
    r13-r14)>0) then 1 else 0
d1314 = if !(x13==x14) && ((r12-r13)*(
    r12-r14)>0) then 1 else 0
d2324 = if !(x23==x24) && ((r12-r23)*(
    r12-r24)>0) then 1 else 0
```

Note that in the above section of code from the EM project, the overall structure of each line is very similar; the only difference is some parts of observable names changing. The ultimate aim of this paper is provide a notation that could represent text similar to the above code in a single line.

3 Technical Layout

At the core of the idea of the Empirical Modelling Text Notation(EMTN) is the decomposition of the source text, whether it be from a source text or directly from the user, into smaller text fragments (henceforth just 'fragments' for brevity).

Throughout the text the beginning and ends of fragments will be denoted by the characters '< ' and '>' respectively. Due to the possible applications of EMTN in programming, ideally, these start and end characters should be changeable as to not conflict with the code in cases where the text contains the characters '< ' and '>' itself. As a solution to this problem, the user should be able to escape these characters by prepending a backslash character or similar.

Once the text has been divided as such, a tree-like structure will emerge. The text that remains at the top level fragment will be at the root of the tree and fragments at lower levels will belong to a parent fragment.

Diagram 1 is a visual example of this structure.

The fragments will be linked with definitive statements, which can be modified by the user as necessary. (In the model, this will be done through EDEN statements.)

Most importantly, once the text has been divided into fragments, the user will be able to replace text fragments with 'function fragments'. The function fragments are one of a number of functions that will specify a range of values.

The functions are listed below:

- seq(A-B) - Outputs a natural number sequence from $A \in \mathbb{N}$ to $B \in \mathbb{N}$. (Inclusive)
- per(S) - Outputs all possible permutations of an input string S. Hence there will be the number of characters in S factorial outputs.
- shft(S) - Outputs all permutations of an input string S that maintain relative order. e.g. S="123" shft(S) = "123","231","312".
- cyclelist(L) - Outputs all the elements of an input list L individually.
- uchar(n) - References a function with a numeric output $n \in \mathbb{N}$ and returns the nth letter of the alphabet in upper case. (modulo 26)
- lchar(n) - References a function with a numeric output $n \in \mathbb{N}$ and returns the nth letter of the alphabet in lower case. (modulo 26)

Once the user has added these function fragments, the user can then 'compile' the final text.

Any fragment with a function fragment as a child is replicated. Each replication of this fragment will contain the text of the original fragment with one of the output values of the child function. If there are multiple instances of the same function as the children of a fragment, the output values from the function are placed in each of the occurrences. For example:

AAA<BB<seq(1-2)>CC<seq(1-2)>DD>EE

becomes:

AAABB1CC1DDBB2CC2DDEE.

Once all the function fragments have been processed, the fragments can be assembled into the final text.

4 Model Implementation

I decided to implement a partial version of the specification described above in EDEN(Engine for Definitive Notations). The definitive structure of EDEN lends itself well to the framework necessary

for the above structure. Additionally, the proposed piece of software should aid in the further development of scripts for EDEN.

EDEN was originally developed for a text editor model, Yung (1987), although this text editor worked in a more conventional style - using vi key commands, the definitive functionality was mainly used internally by the program, to track screen size and cursor location. In this model, we hope to create something more akin to a tool to help with development in an EM style, than a conventional tool developed in an EM style.

Only a simple GUI structure is necessary for the model. I will be using two text boxes and two buttons.

The first text box will contain the user's input and the second will contain the output of the model. One of the button will invoke the pre-processing that needs to be performed on the text. This pre-processing will parse the text containing the '<' and '>' characters and pass the result into an appropriate data structure.

Ideally this pre-processing would take place automatically in response to valid user input. In this model however, in order to demonstrate the principles of the proposed notation, this must be done manually.

The second button will perform the final 'compile' of the text. After the user has made changes to the structure of the code and inserted function fragments, this button will output the final result to the second text box after processing the function fragments.

The two most crucial processes in the model are the functions that are called by the buttons: the process text function and the compile text function.

Due to these functions involving significant amounts of string processing, they may have a more procedural flavour than many empirical modelling programs. This is necessary in order to create the environment in which the user of the model can interact with text as observables with dependencies.

The model may also have reduced functionality compared to the outline above. The model is more important as a proof of concept; a demonstration of the feasibility of the proposed syntax.

5 Assessment of Model

The final model has a number of shortcomings, but ultimately performs as intended when the functions that have been implemented are invoked.

5.1 Replication of text through seq

- Let the input text be

  ```
  "(code) < variable<x>=0> (more code
      )".
  ```

- Perform text pre-processing.
- Set B1f=

  ```
  "seq(1-12)".
  ```

  Previously B1f=

  ```
  "x".
  ```

- The compiled text then becomes:

  ```
  "(code) variable1=0 variable2=0
      variable3=0 variable4=0
      variable5=0 variable6=0
      variable7=0 variable8=0
      variable9=0 variable10=0
      variable11=0 variable12=0 (
      more code)"
  ```

This process replicates a line of code with the correct variation specified by a function fragment.

5.2 Comments and ignoring text

- Let the input text be

  ```
  "This is content. < This is a
      comment.>This is more content."
  ```

- Perform text pre-processing.
- Set A1f=

  ```
  "".
  ```

  Previously A1f=

  ```
  "This is a comment."
  ```

- The compiled text then becomes:

```
"This is content. This is more
    content."
```

This is perhaps a trivial use of the functionally available in EDEN, but demonstrates the ease with which unwanted sections of the text can be removed.

5.3 Multiple references to a function fragment

- Let the input text be

```
"(text)< This is line < x>. I
    repeat, line < x>.>(more text)
    ".
```

- Perform text pre-processing.
- In EDEN, set:

```
B2f is B1f;
B1f = "seq(1-3)";
```

- The compiled text then becomes:

```
"(text)This is line 1. I repeat,
    line 1.This is line 2. I repeat
    , line 2.This is line 3. I
    repeat, line 3.(more text)"
```

The model allows the user to place multiple references to a single function fragment throughout a parent fragment by using the definitive notation of EDEN.

5.4 Replication of text through a comma separated list

- Let the input text be:

```
"Hello < name>,
This is a letter.
Regards,
Robert Steele
"
```

- Perform text pre-processing.
- Set A1f=

```
"cyclelist(James, Alex, Fred, Leon)".
```

Previously A1f=

```
"name".
```

- The compiled text then becomes:

```
"Hello James,
This is a letter.
Regards,
Robert Steele

Hello Alex,
This is a letter.
Regards,
Robert Steele

Hello Fred,
This is a letter.
Regards,
Robert Steele

Hello Leon,
This is a letter.
Regards,
Robert Steele"
```

This demonstrates the usage of the model as a mail-merge tool. This can easily be adapted to a more general version of the seq function, but requiring additional input.

5.5 Shortcomings of the Model

Of the function fragments described in the previous section, only seq, shft and cyclelist have been implemented.

Additionally, the number of text fragments at each 'level' is limited. The main text may only contain 3 child fragments, each of which can only contain 3 child fragments. Furthermore, these fragments cannot contain child fragments.

The maximum fragment tree is as in diagram 2, attached at the end of the paper. In this diagram, the fragments are labelled by the index they appear in the model as.

The current model does not recognise escaped characters. Any '< ' or '>' characters will be treated as fragment markers.

A major drawback of the model is the inability to perform 'two dimensional' loops. Currently, the model lacks the ability for a fragment to have

multiple function fragments as children. This means that, for example:

A<seq(1−2)>B<seq(3−4)>C

does not output the desired:

A1B3C
A2B3C
A1B4C
A2B4C

I firmly believe that this is an achievable aspiration for the model. Further development of the model should manage this, but a fundamental restructuring may be required, which I do not have the time to implement.

6 Conclusion

In this report I have outlined a framework for an EM text processing environment. The model in EDEN included with the report demonstrates the feasibility and functionality of this.

The model has multiple shortcomings, many of which I believe could be remedied with extended development. If completed, the program described in the specification would provide a powerful text manipulation tool that both embodies the principles of EM and would aid the development of further models.

References

M. Beynon. Lines. *Empirical Modelling Archive*, 1991.

M. Beynon N. Pope. Empirical modelling as an unconventional approach to software development. *Proc. SPLASH 2010 Workshop on Flexible Modeling Tools*, 2010.

E. Yung. Unconventional text editor. *Empirical Modelling Archive*, 1987.

Diagram 1:

"(code) < variable<x>=0>(more code)<call function<y>>(even more code)"

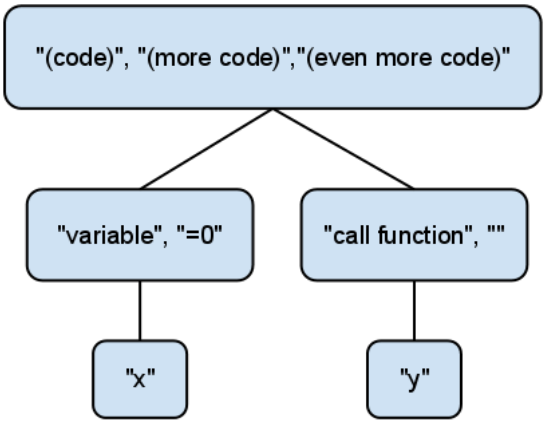

Diagram 2: