

Assignment 03 Re-specified - Program Refinement (Due: Thursday November 6, 2008)

This assignment is a re-specification and a follow-up to the Java Program Component for Automatic Singular/Plural Differentiation assignment and to Assignment 2 (the production of documentation for a prototype I created) and Assignment 3. The work submitted by each pair (for the first part) and then by each student (for the documentation part) shows interesting strengths and weaknesses. Furthermore, I found the discussions and collaboration that took place on October 16th pertaining to Assignment 3 to be very productive.

Since October 16th I have put some productive effort into this project and have produced the following results.

- I have developed the class `PluralityRule` as a component to represent rules for the formation of plural word forms given singular forms, and for likewise forming singulars from plurals. The program `TestPluralityRule` is provided to serve as an illustration of how this functions.
- I have developed the class `WordTree` as a form of "dictionary tree" along with a program `TestWordTreeR` serving as an illustration of this structure. This structure, as given, provides a means to represent words in a reasonably efficient manner and to associate each word with a single non-zero integer value. The intent is that class will be used to associate every confirmed word with the corresponding rule(s) which may be correctly used to form that words plural form.
- The `WordTree` class makes use of the `WordNode` and `WordNodeStack` classes as subcomponents.

Each of the following remains to be done:

- First and foremost the original `Plural` class needs to be reengineered so as to make use of the components I have developed. In particular the simple `List` objects are to be replaced by a single instance of a `WordTree`. File management operations need to be developed for the component allowing for the word tree to be built from a previously stored "word tree file" (`English.dct` for example). This file would be updated only by the "trainer program" discussed below.
- In addition, the component needs to read a "rules file" (`English.rls` for example) because these then serve as the primary means by which the encoded rules are known. A secondary "exceptions file" (see below) containing additional rules likewise needs to be read.
- The "trainer" program must be developed to allow a human user to confirm (or correct) the plural formations generated by the component. The essence of this program is one which prompts the user to confirm or correct each word in a "log file" (`English.log` for example). In this way, words are removed from the log file and then added to the word tree, possibly with a corresponding "exception rule" being added to a "exceptions file" (`English.exp` for example).
- Since some words do have multiple acceptable plural forms the `WordTree` class needs to be modified to allow for the "data" value associated with each word to be extended from a single `int` value to a list of `int` values. Java's `ArrayList` may be useful for this purpose. Similarly the `getPlural` method should be overloaded, or provided with some iteration mechanism to allow for the multiple plural forms to be obtained.

In summary, I envision this component as requiring each of the following data files for each natural language application it is to be used with. English is used as the example language.

`English.dct` - A text file containing every word represented in the dictionary tree associated with the rules (identified by rule number) that may be used to form that words valid plural forms.

`English.rls` - A text file containing allow of the initial known rules for the formation of plural forms from singular forms. The rules in this file must be given in the reverse order they are to be considered. Thus, the first rule is to be the most general rule, the one to be considered only when no other rule applies.

`English.exp` - A text file similar to `English.rls` that also contains known rules serving as "exceptions" to the standard rules in `English.rls`. The protocol is that these rules are considered before the rules in `English.rls`.

`English.log` - A text file which presents the singular and plural forms of unconfirmed words actually generated by the component. The intent is that each of these will then be confirmed or corrected by the trainer program so that all subsequent queries regarding the work will give confirmed results.

Let's see if we can finish this,
P.M.J.