Functional Programming
Exercise 2: Caesar Cipher

Shin-Cheng Mu

2012 Formosan Summer School on Logic, Language, and Computation
Aug 27 – Sep 7, 2012

The purpose of this exercise is to familiarise one with list processing and a programming style nicknamed "wholemeal programming", in which an aggregate data structure is treated as a whole. This exercise is adapted from Hutton [Hut07].

Many functions mentioned in the class or defined in previous exercises will be useful.

Download the file `Caesar.ml` from the course website and load it into the OCaml toplevel by the command `#use "Caesar.ml"`.

1. Complete the definition of *shift*. For some examples:

   - *shift* 2 `'b'` = `'d'`
   - *shift* 8 `'3'` = `'3'`, *shift* 8 `'A'` = `'A'` — *shift* operates only on lowercase leters.
   - *shift* 2 `'y'` = `'b'`
   - *shift* (-1) `'a'` = `'z'`

   Note the pecular (in my opinion, wrong) behaviour of the *mod* operator of OCaml: $(-2)\ mod\ 26 = -2$, not 24.

2. Define

   $$\begin{array}{lll} encode & : & int \to char\ list \to char\ list \\ encode\_str & : & int \to string \to string \end{array}$$

   that perform Ceaser ciphering.

3. Complete the definitions of functions

   $$\begin{array}{lll} count\_eq & : & 'a \to 'a\ list \to int \\ count\_lower & : & char\ list \to int \end{array}$$

   where *count_eq* x xs yields the number of occurrences of x in xs, while *count_lower* xs yields the number of lowercase letters in xs.

   You may find their definitions very similar — both are instances of a function

   $$count\ :\ ('a \to bool) \to 'a\ list \to int$$

   such that *count* p xs counts the number of elements in xs that satisfies predicate p.

4. Define

   $$histo\ :\ char\ list \to float\ list$$

that computes the percentage of each lowercase character in the input list. The result is a list of 26 floating point numbers, one for each alphabet. For example,

$$histo\ (explode\ \texttt{"aabc"});;$$

evaluates to

$$[50.; 25.; 25.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.]$$

because 50% of the characters in `"aabc"` are `'a'`, 25% are `'b'`, etc.

Note that the denominator should be the number of lowercase letters, rather than the length of the entire string. For example, *histo* (*explode* `"aabcA,A"`) yields the same result.

5. Define

   $$crack\ :\ char\ list \rightarrow int$$

   that takes an supposedly encoded string and compute the most possible offset. You will need plenty of helper functions and values including *table* (the average histogram of English alphabets), *rotate*, *index*, *map*, *filter*, etc. You may also find the following predefined functions useful:

   - *minimum* : *'a list* → *'a*, that returns the minimum element of a given list;
   - *hd* : *'a list* → *'a*, that returns the first (left-most) element of a list;
   - *chisqr* : *float list* → *float list* → *float*, such that *chisqr es os* computes the similarity between *es* and *os*. The smaller the outcome, the more similar *os* is to *es*. **Note**: the order matters: *es* is the "model", while *os* is a particular table to compare against *es*.

   One possible way to compute *crack xs* is to

   - compute the histogram of the input list. Call the result *freqs*.
   - Compute all the 26 rotations of *freqs*.
   - Find, among all the rotations of *freqs*, the position of the element that is the most similar to *table*.

   Hint: how do we find the position of the minimum element in a list, say, *ys*? Of course, it is the position of the left-most element that equals *minimum ys*.

6. Define

   $$\begin{array}{lll} decode & : & char\ list \rightarrow char\ list \\ decode\_str & : & string \rightarrow string \end{array}$$

   that deciphers an encoded string by calling *crack*.

References

[Hut07] Graham Hutton. *Programming in Haskell.* Cambridge University Press, 2007.