Squarefree words

The study of squarefree words (more generally, words avoiding certain patterns) has important applications, for example, in formal language theory, combinatorics and computational biology. In this coursework, squarefree words are used to practice programming with lists.

Some definitions and remarks.

By a word we mean a finite list. Following common practice, we denote a word by simple writing its letters next to each other, for example, \textit{abcabb}. The empty word is denoted by \(\epsilon\), and \(vw\) denotes the concatenation of the words \(v\) and \(w\).

Note that in Haskell the word \textit{abcabb} would be implemented as \([a,b,c,a,b,b]\).

The type of the letters of a word is not fixed in advance. Hence, in the word \textit{abcab} the letters \(a,b,c\) could be characters, or integers, etc. Make sure that the signatures of your Haskell functions appropriately reflect this.

A \textit{subword} of a word \(w\) is a word \(v\) such that \(w = u_1vu_2\) for some (possibly empty) words \(u_1, u_2\).

A word \(w\) is called a \textit{square} if it can be split into two identical parts, that is, \(w = vv\) for some word \(v\). For example, \(aa\) and \(abab\) are squares, but \(abba\) isn’t.

Question 1. Define a function that tests whether a word is a square. [15 marks]

A word is called \textit{squarefree} if none of its nonempty subwords is a square. For example \textit{abcab} is squarefree, but \textit{abbab} isn’t.

Question 2. Define a function that tests whether a word is squarefree. [15 marks]

Question 3. Give a rough estimate (using the big-Oh notation) of the runtime of your test for being squarefree in terms of the length of the input word. [15 marks]
In the remaining questions we consider words of integers only.

For every natural number \( n \) we define inductively a word \( w_n \) as follows:

\[
\begin{align*}
  w_0 & = \epsilon \\
  w_{n+1} & = w_n w_n
\end{align*}
\]

For example, \( w_1 = 0 \), \( w_2 = 010 \), \( w_3 = 0102010 \), e.t.c.

**Question 4.** Define a function that computes \( w_n \) for every \( n \). Confirm empirically that all \( w_n \) are squarefree.

**[15 marks]**

**Question 5.** Prove that all \( w_n \) are squarefree.

**[15 marks]**

**Question 6.** Find a formula for the length of \( w_n \) and prove it.

**[5 marks]**

An \( n \)-ary word is a word built from the letters \( 0, \ldots, n-1 \). If \( n = 2 \) respectively \( n = 3 \), one says “binary” respectively “ternary”, instead of “\( n \)-ary”. For example, \( 001101 \) is a binary word and \( 0102010 \) is a ternary word. Clearly, \( w_n \) is \( n \)-ary for every \( n \).

**Question 7.** Prove that \( w_n \) is maximally squarefree among all \( n \)-ary words. That is, if \( 0 \leq i < n \), then \( w_n i \) is not squarefree.

**[5 marks]**

**Question 8.** Compute the first 100 elements of \( w_{10000} \). Explain why Haskell is able to do this computation. Would this computation be possible in Java or Lisp?  

**[5 marks]**

In the following we consider ternary words only.

Consider the following operation \( \sigma \) on ternary words that begin with 0 or 1:

Replace each occurrence of the letter

\[
\begin{align*}
  0 & \text{ by } 0102 \\
  1 & \text{ by } 1012 \\
  2 & \text{ by } 1202 \text{ or } 0212 \text{ depending on whether the preceding letter is } 0 \text{ or } 1.
\end{align*}
\]

**Question 9.** Implement the function \( \sigma \).

**[5 marks]**

For every \( n \) we define inductively a ternary word \( s_n \) as follows:

\[
\begin{align*}
  s_0 & = 0 \\
  s_{n+1} & = \sigma(s_n)
\end{align*}
\]

Hence \( s_1 = \sigma(0) = 0102 \), \( s_2 = \sigma(0102) = 0102101201021202 \), e.t.c.

Axel Thue (Norwegian Mathematician, 1863 - 1922) showed that all \( s_n \) are squarefree.

**Question 10.** Implement \( s_n \) and confirm Thue’s result empirically.

**[5 marks]**

**Remark:** In fact, \( s_{n+1} \) always extends \( s_n \), so one obtains an infinite squarefree ternary word.

**Submission:** Please see the course web page [http://www.cs.swan.ac.uk/~csulrich/fp1.html](http://www.cs.swan.ac.uk/~csulrich/fp1.html) for instructions.