The Thue-Morse Sequence

Jan Rochel

to be handed in before 26-09

In this assignment you will compare different implementations of the Thue-Morse sequence and test some of its properties using QuickCheck.

The Thue-Morse sequence is an infinite binary sequence beginning with

0 1 10 1001 10010110 1001011001101001 . . .

It can be defined as the sequence \( t \) satisfying:

\[
\begin{align*}
    t_0 &= 0 \\
    t_{2n} &= t_n \\
    t_{2n+1} &= 1 - t_n
\end{align*}
\]

1. Define the Thue-Morse sequence as a list of Ints called \( \text{tms1} \) according to this definition.

2. What can you say about the efficiency of \( \text{tms1} \)? Explain. Hint: there is more than one aspect to discuss.

3 (Bonus). Find a nice way to amend the efficiency problem of \( \text{tms1} \) while still sticking to the above definition of \( t \). Compare the efficiency of both implementations using ghci’s command \( \text{:set +s} \). The following idea might help you with \( \text{tms2} \):

\[
\begin{align*}
    t_0 &= t_1 = 1 \\
    t_2 &= t_3 = 1 \\
    t_4 &= t_5 = 1 \\
    t_6 &= t_7 = 1 \\
    \ldots
\end{align*}
\]

The next variant relies on the fact that every prefix of \( t \) with a length that is a power of two is followed by its binary inversion.

4. Write a function \( \text{invert} \) that implements this inversion, and use it to define \( \text{tms3} \), which should construct the sequence by recursively appending to prefix \( p \) the inverse of \( p \).
Another definition relies on parities where \( t_n = 0 \) if \( n \) interpreted as a binary number has an even number of 1s, otherwise \( t_n = 1 \).

5. Define a function from \( \text{Int} \) to \([\text{Bool}]\) that converts a natural number into its binary representation. Use it in implementing \( \text{tms4} \).

A more difficult-to-understand definition is based on \( \text{entangle} \).

\[
\text{tms5} :: [\text{Int}]
\]
\[
\text{tms5} = 0 : \text{xs} \quad \text{where}
\]
\[
\text{xs} = 1 : \text{entangle} \text{xs} \text{ys}
\]
\[
\text{ys} = 0 : \text{entangle} \text{ys} \text{xs}
\]

\( \text{entangle} \) is a function that merges two lists \( l_1 \) and \( l_2 \) by taking turns in picking elements from \( l_1 \) and from \( l_2 \), thus \( \text{entangle} \text{"thue"} \text{"morse"} \) should yield \text{"tmmhoresse"}.

6. Complete this implementation by defining \( \text{entangle} \).

Here are two more implementations. The latter builds the sequence successively by substituting in each step all occurrences of 0 and 1 by 01 and 10 respectively.

\[
\text{tms6} :: [\text{Int}]
\]
\[
\text{tms6} = 0 : \text{entangle} (\text{invert} \text{tms6}) (\text{tail} \text{tms6})
\]
\[
\text{tms7} :: [\text{Int}]
\]
\[
\text{tms7} = 0 : 1 : \text{rw} (\text{tail} \text{tms7}) \quad \text{where}
\]
\[
\text{rw} (0 : \text{xs}) = 0 : 1 : \text{rw} \text{xs}
\]
\[
\text{rw} (1 : \text{xs}) = 1 : 0 : \text{rw} \text{xs}
\]

The elegance of \( \text{tms5}, \text{tms6}, \) and \( \text{tms7} \) (hint: maybe also \( \text{tms2} \)) is possible due to lazy evaluation. One might even be surprised how they are able to function given the ubiquitous auto-dependency.

7. Evaluate \( \text{tms7} \) by hand. In doing so whenever necessary you should assign labels to positions within the produced list which act as representatives for suffixes of the list. Complete the template below until you have produced the first 10 elements of the result.

\[
\begin{align*}
\text{-- tms7} & = 0 : 1 : \text{rw} (\text{tail} \text{tms7}) \\
\text{-- A} & = 0 : 1 : \text{rw} (\text{tail} A) \\
\text{-- ...}
\end{align*}
\]

Next we are going to use QuickCheck to ensure that all of the above implementations are equivalent. First, make sure that it is installed by running \texttt{ghc-pkg list} from the command line (not \texttt{ghci}). If it isn’t, install it with the \texttt{cabal update} and then \texttt{cabal install quickcheck}, also from the command line. In your module you have
to import Test.QuickCheck, which brings the required functions into scope, most importantly `quickCheck`:

```haskell
quickCheck :: Testable prop => prop -> IO ()
```

Think of `prop` as a function with an arbitrary number of parameters and a `Bool` as a result type. In ghci, try to apply `quickCheck` to `prop_isBinSeq tms4` with

```haskell
prop_isBinSeq :: [Int] -> Int -> Bool
prop_isBinSeq tms idx = let t = tms !! idx in t == 0 || t == 1
```

The property will then be tested against a large, randomly generated control sample for `idx` to acquire evidence on whether the property holds for all `Int`s. If you need to include a precondition in your tests you can do so by using material implication

```haskell
infixr 1 -->
(--> :: Bool -> Bool -> Bool
x --> y = not x || y
```

If you run into performance issues you can limit the size of the samples by using e.g. `limitedCheck limit = quickCheckWith stdArgs {maxSize = limit}`. Start with small values for `limit` to prevent ghci from crashing. Use `verboseCheck` instead of `quickCheck` to get additional information about the test runs.

8. Write a test `prop_tmsImplsEquiv` to assert that all the above variants of the Thue-Morse sequence are equivalent.

Next we are going to test various properties of the Thue-Morse sequence. Assured that the implementations are equivalent let us select a one of the variants to perform further testing on. Use ghci’s `:set +s` flag to determine which one is most efficient.

For each of the following properties you are to write a QuickCheck test to assert this property for any given infinite list of `Int`s.

9. **prop_subseq4HasSquares**. A *square* is a string of the form `xx`, i.e. a direct repetition of two non-empty strings. There is no substring of the Thue-Morse sequence of length four or more that is square-free.

10. **prop_cubeFree**. A *cube* is a string of the form `xxx`. The Thue-Morse sequence is cube-free.

11. **prop_overlapFree**. The Thue-Morse sequence is overlap-free. An *overlap* is a string of the form `axaxa` where `a` is a single letter and `x` is a string.

---

1. QuickCheck does offer more sophisticated approaches, they do however involve concepts that have not been covered in the lectures yet
2. Here, the word ‘string’ does not refer to Haskell Strings as a data type, but is used as an alias for ‘finite sequence’.
Be careful to avoid non-termination for the last two tests.

12 (prop_inverseSubstring). For all of its substrings the Thue-Morse sequence also comprises their inverse.

13 (prop_mirrorSubstring). For all of its substrings the Thue-Morse sequence also comprises their mirror image.

Remark: Do not settle for something that just works, but code that is appealing to the eye. Beautiful code is usually concise and documents itself. Be assured that for each of the exercises there are concise solutions. If you aren’t convinced that you’re code looks good, ask the assistants for suggestions. Code quality is a decisive factor for your grade.

Important: Only code that passes the type checker is accepted. If you need to leave gaps in your module use undefined. Please stick to the function names of the assignment.

References


   http://ai.uwaterloo.ca/~shallit/Papers/ubiq.ps

   http://dx.doi.org/10.1016/0304-3975(89)90013-3

   http://www.cse.chalmers.se/~rjmh/QuickCheck/manual.html