Software Specification and Testing 2015
Lab Exam - Mats Stijlaart (11152990)

Problem 1

See Assignment1.hs

Problem 2

1) Give one more example of an even function on the real numbers.

A first example would be the abs function that takes a number and converts it to its absolute ($\lambda x \mapsto |x|$). Other examples may be functions of the form $\lambda x \mapsto x^n$, where n is an even integer.

2) Explain why it is impossible to write an algorithm that checks whether a function of type Float $\mapsto$ Float is even.
This function has universal quantification. We will have to check all floats as input for this function to verify this property and this is not possible since this is an infinite set. As from the lecture: It is **not** possible to verify universal quantification, we can only falsify it.

3) Implement a test maybeEven for evenness of functions, and explain why the outcome False of this test is more reliable than the outcome True.
The code of this question is located in Assignment2.hs. As described in the previous paragraph, we can not verify a infinite set of numbers. When you run a lot of test you may start to assume that a certain property hold, but you are never sure. However when the property fails, there is a counter-example, thus a 'False' result is always more reliable.

Problem 3

See Assignment3.hs. I could have used the following snippet (then I could have removed the base case in the mergeC), but had to introduce another case for freqList to handle empty strings.

```
foldr mergeC [(last (sort s), 1)] (init (sort s))
```

Problem 4

See Assignment4.hs. The question states 'any frequency list'. A valid frequency list may be an empty list, however the given implementation of createTree is non-exhaustive, thus I have added a precondition that checks if the input is a non-empty list.

Problem 5

The buildTree function first sorts the input table based on the weight (the first element in the list will have the least occurences). This is followed by mapping all items in the sorted frequency list to Leafs (the implementation could use an uncurry). In the end it will build the tree by taking the first two items of the sorted list, merging them to a new Form and than insert this fork on the correct position (insertBy) to make sure the list is still ordered. The last step is recursively and will continued until only one element is left and returns this element. The result will be a tree with the smallest elements as deep as possible in the tree.
The test for this implementation is provided in Assignment5.hs. The same not-empty precondition as in Problem 4 must hold for this test.

Problem 6

Implementation

To test the encode message I make sure that 'fqs' contains all the characters in 'message'. To do this I've created an arbitrary instance for 'FrequencyMessage' that will make sure that 'message' is a sub list of 'fqs. Next there should be a precondition that checks that the 'fqs' is non-trivial. This is implemented in 'validFrequencyMessage'.

Properties

The first property that I identified is the property that encode end decode should be inverses from each other as long as the preconditions are met. This is tested with testEncodeDecode and testDecodeEncode. The next property is that encode should fail when 'message' contains a character that is not in 'fqs'.

Problems

The first problem was found with the testDecodeEncode. The 'decode' function was non-exhaustive for empty bit-lists. For example 'decode "abc" []. This problem could be fixed by add the following case to 'decode':

```
decode _ [] = ""
```

I have patched the LabExam.hs for this change.

Another problem was shown with a bit list where the path of ones and zeros does not match any node in the HTree. For example with the string "abc" and the bit list [1] as input for decode, the single 1 does not match any path in the tree. I have added another case for 'dcd' to handle Fork's when the bit list is empty. This case throws an exception. To actually test the right-inverse 'decode' on the bijection $encode \cdot decode$, the precondition on the input is very strong. Therefor I have added a arbitrary instance DecodeInput to generate valid input to test this inverse.

Conclusion
I've tested the output of encode by using decode and vice versa and caught some small bugs. After this I can say I am quite confident that the functions are tested enough.