

## Exercise (Tuesday): Testing with Properties

In today's exercise we will develop a variety of properties for testing the binary search tree implementation discussed in the lecture. The materials provided include a correct implementation in `BST.hs`, and the properties presented on the slides in `BSTSpec.hs`. `BSTSpec` uses Template Haskell and `quickCheckAll` to define `runTests`, which tests all the properties in the module. Make sure you can run these tests, and use this file as a starting point for your work. As you develop new properties, you should make sure they continue to pass for `BST.hs`. The files `BST1.hs` ... `BST8.hs` contain different buggy versions of `BST.hs`, with a different bug in each file. **Do not read these files**; we will use them later to practice diagnosing bugs by testing.

### Writing Properties for BST

As you write properties to solve the next exercises, make sure you test each property as soon as you have written it, either by running `quickCheck` with the property as an argument, or by using `runTests` to test all the properties in the `BSTSpec` module. Since you are testing a *correct* implementation, all the properties you write should pass. (If a property fails, you need to correct the property, not the code!) You can run more tests of a property using a command such as

```
quickCheck . withMaxSuccess 10000 $ prop_UnionPost
```

1. `BSTSpec` contains a *validity property* for **insert**. Add similar properties for **nil**, **delete**, and **union**.
2. `BSTSpec` contains *postcondition properties* for **find** and **insert**. Add postconditions for **delete** and **union**.
3. `BSTSpec` contains *metamorphic properties* for **size/insert** and **insert/insert**. Add further metamorphic properties to test **insert**, **delete**, and **union**.
4. `BSTSpec` contains a *model based property* for **insert**. Add model-based properties to test **find**, **nil**, **delete**, and **union**.

### Diagnosing Bugs with Properties

Once you are satisfied with your properties, try using them to diagnose buggy implementations. The files `BST1.hs` ... `BST8.hs` contain different buggy versions of `BST.hs`, with a different bug in each file.

**Do not read these files.** Four bugs are revealed by the original `BSTSpec.hs` provided, and four slip through. Each of these versions can be tested using `BSTSpec.hs`, by changing `import BST` to `import BST1` ... `import BST8`.

5. Run your tests on each of the buggy implementations, using `runTests`. If any implementation passes all your tests, something is wrong!
6. Which properties are most effective at revealing errors?
7. Use the counterexamples found to diagnose each bug. Which properties give the most helpful output? Once you think you know what the bug is, you may inspect the implementation to see if you are right.