Thomas Hart
CS 329
Final

Problem 1

Truth Table

A is: node != null, b is: node.left != null, c is: node.right != null

A	B	C	Outcome
T	T	*	T
T	F	T	T
T	F	F	F
F	*	*	F

Binary Tree diagram that would test all outcomes given all nodes are traversed:

1 ← first test (A and B true, C is irrelevant)

^

2 3 ← second test (A and C true, B false)

^ ^

4 Null Null 5 ← third test (A true, C and B false)

^ ^

Both Null Both Null ← fourth test (A false, B and C irrelevant)

Problem 2

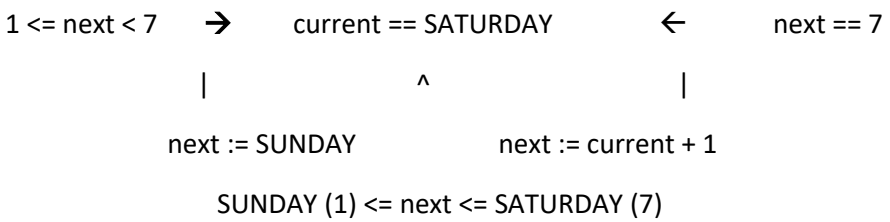
I would create a test that uses the runToRandomState function and create an identical function for a Java Set. Using the same seed, I would call the exact same function calls using the same parameters to verify the Trie and the Java Set were identical in their states each time a particular method was called on them. I'd run this many times with a random seed each time to ensure the test was checking for all kinds of situations.

The next test would be the same thing, expect this time we would test two Tries against each other instead of a Trie and a Java Set.

The third test would be an algebraic property test, where we run the random test mentioned above on two Tries (prefix), execute an algebraic property, then do the random test again (postfix). Then we check to see if both Tries are identical. An algebraic property we could test would be adding an element to the first Trie and removing the same element, which should be identical to the second Trie. Another

algebraic property we could test would be calling equals or contains on one Trie and asserting that it is still identical to the second because those two functions should change it.

Problem 4



Weakest precondition is 1 <= next <= 7

requires 1 <= next <= 7

Survey

What did you learn in this class?

I learned how to properly test programs. I didn't previously know all the possible holes in tests and how you could do it without optimal coverage, but now I understand how to cover it all. I also learned about type proofs and how those are verified so incompatible types aren't mixed. I learned dafny and how it verifies things as you code. I learned about abstract syntax trees and how they can be used to test syntax. I learned how to use the AST library used in class. I also learned how to use github more effectively and understand the branch system better. I feel more equipped to work as a team by pushing branches and merging things after they've been verified. I feel more equipped to use submodules as well. I also learned about stateful property-based testing, weakest precondition analysis, making a truth table to get full MC/DC coverage, and more.

What did you like about this class?

I like the real-world application. I image we'll always need to test code, use github-type system to work in groups, and think deeply about what could be flawed in our code and tests. I like all the resources given so I could figure things out from home if I couldn't make it to class.

What would you change in this class?

Maybe it was meant to be difficult, but the AST library we used was hard to sort out. Maybe additional resources or a cheat sheet of useful functions would be helpful. For example, it took me a while to figure out how to create a new Boolean Literal when I was first using it. I finally figured out I could get the AST from an AST node and use that to create a new literal, which didn't seem immediately intuitive. Just little things like that.