Comp 590 Final Project Update

Proving Binary Trees

Will Swinyer
Undergrad Student - CS
University of North Carolina at Chapel Hill
Chapel Hill, NC
swinyer@live.unc.edu

Kendall Miller
Undergrad Student - CS
University of North Carolina at Chapel Hill
Chapel Hill, NC
kkm@live.unc.edu

Andrew Mahn
Undergrad Student - CS
University of North Carolina at Chapel Hill
Chapel Hill, NC
mahndrew@live.unc.edu

I. Project Proposal

We will create an implementation of the Binary Tree data structure within Coq. First the data structure and functionality will be implemented, then the properties of the data structure and validity of its functions will be proven. The biggest hurdle for us will likely be figuring out ways to translate the implementations of Binary Trees that we already know into formats that are acceptable within Coq, namely operations in level order fashion. Our goal is to implement a Binary Tree data structure that creates a balanced, level order tree of nats from an input list of nats and is capable of summing all of its nodes. On a basic level, we'd like to implement functionality for insertion, deletion, and emptiness checks as well. Once all of these functionalities are implemented, our goal is to prove the correctness of all of these capabilities.

II. Goals / Timeline Update

During this project, we will aim to create and prove the correctness of the binary tree data structure within the Coq platform, as well as prove the validity of different common helper algorithms to ensure correctness when using a binary tree to store and traverse data.

Our minimum goals included defining a binary tree within Coq, as well as proving some of the basic methods and algorithms that are often used to manipulate the tree structure. Such aspects would include the basic tree format with nodes having the ability to contain data as well as point to other children within the structure. In order to manipulate our data structure, we have chosen to prove the two most utilized algorithms, *insertion* and *deletion*. These methods will allow us to handle growing and shrinking the tree in the most efficient way possible while proving the validity of said algorithms.

As we will discuss later, some of these goals proved to be more difficult than anticipated, especially when keeping in mind our standard goals. Our minimum goals will remain the same, but their functionality may change depending on if we change the invariants associated with our standard goals.

For our standard goals, we chose to include the basic creation and iteration of the tree, specifically by including the construction of a tree from a given array in level order fashion. We will then aim to prove the validity of other useful methods such as checking if the tree is full and computing the sum of all the nodes in the tree. Our final standard goal is to implement and prove the correctness of the *checking height balance* algorithm.

The level-order goal has become a roadblock for us. This goal requires we sort our tree upon creation and with each insertion and deletion. As we were developing our base functionality for the binary tree, we quickly realized that the implementation of these functions will be different and much more complicated if we uphold this invariant. Because of this, we are struggling with our minimum goals and may need to reassess our goals.

For our reach goals, we have chosen to provide the correctness for more complex methods that we have found helpful in our own projects in the past. This includes algorithms such as converting a tree into a doubly-linked list and searching for specific root to leaf paths with a given sequence. Lastly, we would like to implement and prove the correctness of an iterative postorder traversal, which is our most ambitious task of the project.

These reach goals seem like even more of a reach now. If we manage to fix our issues with the standard goals, then we will continue to work towards these reach goals. However, in the event that doesn't happen, we will have to scale these goals back as well.

In our original timeline, we planned to have our minimum goals completed or well on their way by the

progress report on March 9th. We then planned for our standard goals to be finished about a month after the progress report, on April 2nd, so we could try to achieve our reach goals by the final proof due date on April 20th. We then planned to have a week to write our final paper, which is due on April 27th, and another week to prepare and give our final presentation on May 6th. We unfortunately have not completed our minimum goals at this time due to unforeseen setbacks. Because of this, we now would like to achieve our minimum goals as well as at least one of our standard goals (especially the level-order array instantiation / balanced invariant) by April 2nd. After this, we hope to use the rest of our time to finish the rest of our standard goals and as many of our reach goals as possible by April 20th.

III. What We Have Done

Thus far, we have been able to implement only some of our minimum goals. We initially and ambitiously tried to make the binary tree polymorphic, but we kept running into errors with constructors expecting arguments of incorrect types when implementing subsequent functionality. For that reason, we scrapped polymorphism and settled for populating our binary trees with nats in order to make progress. Once we changed our approach, we were able to successfully define an Inductive binary tree type, a working insert function, and an emptiness-checking helper function.

Our remaining minimum goal to implement functionality for deletion from the tree proved much more difficult than anticipated. This is because we attempted to do so while keeping the tree balanced, which sometimes involves re-arranging the children of some nodes. This logic will be difficult to implement in Coq, and becomes even more difficult and complicated when you attempt to balance the tree. Because of this. we have yet to implement delete.

IV. Final Proof Goals

Because of our struggles to properly implement basic functionality, we have not yet begun to prove propositions about our data structure. In our proposal, we stated that we are aiming to "prove the correctness of" various aspects of our data structure. As we've begun the project and thought more about what this actually means, we've come to realize that we instead need to prove various properties of binary trees. These could include balance (every node with 0 or 1 children has a height of 1) and tree height based on node count. As long as our invariants hold, we will know exactly what the height of a tree and all of its nodes are and where any item/value will be located within the tree at all times. We're not sure if or how we can generalize this, but we believe we should be able to prove these assumptions for any concrete example we can come up with.

The first proofs we will write will be related to insertion and deletion within a binary tree. We hope to complete this relatively soon as it was included in our minimum goals. Currently, our approach will be to prove that various sequences of insertions and deletions will be equal to sample binary trees representing their expected outputs.

Later, once we implement our standard goals, we hope to prove them through similar means. However, we would also like to prove propositions about the properties of our trees. For instance, we would like to prove the height of a balanced binary tree as a function of its node count.

V. Functions / Theorems Completed

So far, we have defined the binary tree as well as some basic functions. We defined an Inductive *BT* as either Empty or a Tree with a left child, value, and right child. The function *is_empty*, which takes in a BT as a parameter, will return *true* if the input tree is empty and false otherwise. Our *insert* Fixpoint will insert a given nat to the given BT. Currently, this function does not balance the tree upon insertion, and we are working on figuring out how to accomplish this.

As we have made progress, we have been checking our work by creating various basic tests using the *Compute* keyword. These are all working as expected so far, and once we iron out the kinks in our implementations we hope to convert these into proofs.