EECS 478 Project 3 Report
Joseph Stevens
joesteve

1. Topological Sort
    a. The topological sort took a bit of time for me to wrap my head around. The recursive method mentioned in discussion didn't seem intuitive to me, so I tried to think of another method to use. The method I ended up using felt much simpler, I started by setting the Primary Inputs and Primary Outputs into their own vectors that I could iterate through later to print them, because I knew that as long as those two vectors went at the beginning and end, respectively, that the order within them didn't matter. I then put all of the Internal nodes into a map which I iteratively searched through, adding each node to an ordered vector of Internal nodes as I determined that all of the inputs for that node were either Primary Inputs or were Internal nodes that we had already seen. I used maps in addition to vectors to hold the PIs and POs so that I could use the find function to find them trivially, in addition to a map in parallel with my vector for Internal nodes that had already been seen, for the same reason. After all of that it was a simple matter of printing all of the names in order to standard output.
    b. All of this was done inside of circuit.cpp because that was where all of the necessary data was.
2. Functional Simulation
    a. Functional Simulation I quickly realized was fairly trivial once a topological sort was done. Therefore, I began the simulation by simply recreating the same data structures from my topological sort and sorting the nodes in the same way. From there, it simply followed that I had to apply the read-in values to the Primary Inputs, and cascade them down through the gates in topological order to reach the Primary Outputs at the end. I had initially presumed that the cascading part would be simple, but I quickly was reminded that I would have to penetrate the truthtable data structure in order to glean whether each gate was a NOT, OR, or AND. The method I came up with for doing this I thought was fairly clever, knowing that NOT gates only ever have 1 input, I checked the number of variables to determine if it could be a NOT gate. If there was only one, I also knew that under that circumstance OR and AND behave identically, so the only question was whether it was a NOT or as I termed it, a NOTNOT. I wasn't sure if this was going to be tested so I went ahead and assumed it would be. Outside of that, if there was more than 1 variable, I would check how many rows there were. If there was 1 row, then it was an AND gate, otherwise it was an OR, because there's only ever 1 instance where an AND gate is true. When I proceeded to parse these, NOTs flipped the incoming bit, and NOTNOTs simply propagated it. For ORs I set the outgoing bit initially to 0, and then flipped it to 1 if I found any of the inputs to be 1. Likewise, for ANDs I set it to 1, and if I found a 0 I set it to 0.