Connor Roos (cgr3mu)

October 25th, 2017

https://gist.github.com/cgr3mu/4dd59c1e2994d6b3eb3906dfceb0a5b6

Objective: Knowing that physical Logic Circuits do not operate exactly like logical formulas (due to the possibility of repeated gates, find the shortest formula that is equivalent to XOR(a,b) using NAND operations. In this case, shortest refers to the minimum number of NAND operations. A convincing answer would include a proof that no shorter formula exists.

Solution: Although Jake Smith's solution for formulas is correct, in a true logic circuit repeated gate terms refer to the same physical gate, so the 5 gate expression NAND(NAND(a, NAND(a, b)), NAND(b, NAND(a, b))) (second solution in his output) can be drawn as the logic circuit shown below. I used Java run in the Eclipse Integrated Development Environment but it should be able to be run in any Java IDE. I also used the assumption that order doesn't matter in the placement of NAND inputs, so that the program would run faster. (NAND(a,b) = NAND(b,a))

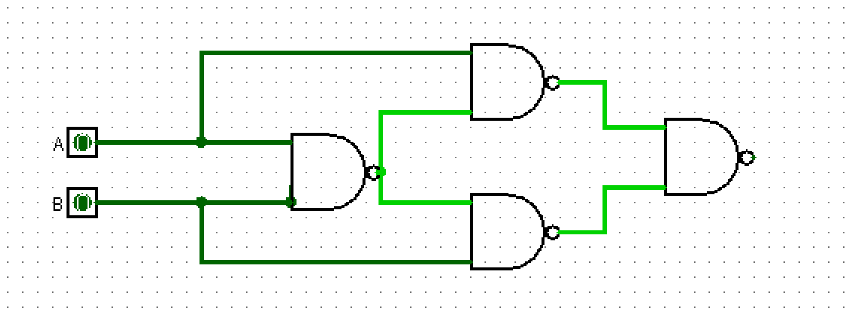

Figure 1: Logisim model of a 4-Gate NAND Solution

My solution utilizes a Java Interface I've created called Circuit that is implemented by my two classes Wire and NandCircuit. Wire represents an input to the circuit from the set {0,1,a,b}, while NandCircuit represents any circuit construction that ends with a NAND gate, as the inputs to a NandCircuit are defined as Circuit objects so they can be either wires or other NandCircuit objects. Both Wire and NandCircuit have methods for getting their inputs as Circuit objects and the name of the Circuit, which is its logical representation.

My solution also has the method numGates.The recursive method numGates works down a circuit counting every unique NandCircuit object, including the object the method is called on as all NandGate objects are defined by their final NAND gate. Each of these unique gates are stored in a Java LinkedHashSet, which prevents repetion by definition. The gates required to create the current circuit are added to the current circuit's LinkedHashSet by using the addAll method (which adds all the object in one Java Container to another, in this case a set to another set). It doesn't return the size as an int, due to how the recursion works, so I use the .size() method on the final set returned to determine if the numGates is less than the maximum gates

My solution has the TreeMap allCircuits that contains all Circuit objects and maps them using their name, which will be used later to see if a newly constructed circuit is already a member of allCircuits. The method XORCheck checks a Circuit for equivalence to XOR by using the method verifyCircuit, which returns the boolean output of a Circuit object under the given conditions set for Wires a and b. This is also accomplished recursively through the circuit, using the Wire objects as an exit case for the recursion.

Finally, for the solution itself. The method findCircuits finds all XOR equivalent of the inputed size or less. The method adds the set of Wires {0,1,a,b} to allCircuits on the first iteration so it has values to construct other circuits from. At the start of each iteration the List currentCircuits (represented as an Array in Java) is defined as all the current Circuit objects in allCircuits, this is done as Lists can be iterated over. A nested for loop with iterators i and j searches for new NandCircuit constructions such that order does not matter, so the initial j value is = i (all objects in currentCircuits are compared to every other object in currentCircuits, including themselves, only once). I ran this program with both j = i and j = 0 and have the computation time listed below. If the object is both new (found by checking if its name is already in allCircuits) and less than or equal to the maximum size, it is added to allCircuits and is checked for equavalency to XOR using XORCheck. findGates stops running when the size of allCircuits does not changed after an iteration (with the previous size of allCircuits stored as the integer oldCircuitsSize).

This Solution Yields 1 4-Gate Solution and no Solutions that have less than 4 gates exhaustively, which means that the **minimim number of NAND gates to construct a XOR gate is 4.**

Q. E. D.

Output when Maximum Size = 4:
```
Nand:(Nand:(Nand:(a, b), a), Nand:(Nand:(a, b), b))is equivalent to XOR!!! And it's 4
Gates long!
4556 milliseconds
```
Output when Maximum Size = 5:
```
Nand:(Nand:(Nand:(1, a), b), Nand:(Nand:(1, b), a))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(1, a), b), Nand:(Nand:(a, b), a))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(1, a), b), Nand:(Nand:(b, b), a))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(1, b), a), Nand:(Nand:(a, a), b))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(1, b), a), Nand:(Nand:(a, b), b))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(a, a), b), Nand:(Nand:(a, b), a))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(a, a), b), Nand:(Nand:(b, b), a))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(a, b), a), Nand:(Nand:(a, b), b))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(a, b), b), Nand:(Nand:(b, b), a))is equivalent to XOR!!! And it's 5
Gates long!
Nand:(Nand:(Nand:(1, a), b), Nand:(Nand:(Nand:(Nand:(1, a), b), b), a))is equivalent
to XOR!!! And it's 5 Gates long!
Nand:(Nand:(Nand:(1, b), a), Nand:(Nand:(Nand:(Nand:(1, b), a), a), b))is equivalent
to XOR!!! And it's 5 Gates long!
Nand:(Nand:(Nand:(Nand:(Nand:(a, a), b), b), a), Nand:(Nand:(a, a), b))is equivalent
to XOR!!! And it's 5 Gates long!
Nand:(Nand:(Nand:(Nand:(Nand:(a, b), a), a), b), Nand:(Nand:(a, b), a))is equivalent
to XOR!!! And it's 5 Gates long!
Nand:(Nand:(Nand:(Nand:(Nand:(a, b), b), b), a), Nand:(Nand:(a, b), b))is equivalent
to XOR!!! And it's 5 Gates long!
Nand:(Nand:(Nand:(Nand:(Nand:(b, b), a), a), b), Nand:(Nand:(b, b), a))is equivalent
to XOR!!! And it's 5 Gates long!
1092767 milliseconds (That’s 18 Minutes!)
```

Output when Maximum Size = 4: and j = 0 (Order of the Gate Inputs Matters)-Just for Demonstration

```
Nand:(Nand:(Nand:(a, b), a), Nand:(Nand:(a, b), b))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(a, b), a), Nand:(b, Nand:(a, b)))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(a, b), b), Nand:(Nand:(a, b), a))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(a, b), b), Nand:(a, Nand:(a, b)))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(b, a), a), Nand:(Nand:(b, a), b))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(b, a), a), Nand:(b, Nand:(b, a)))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(b, a), b), Nand:(Nand:(b, a), a))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(Nand:(b, a), b), Nand:(a, Nand:(b, a)))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(a, Nand:(a, b)), Nand:(Nand:(a, b), b))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(a, Nand:(a, b)), Nand:(b, Nand:(a, b)))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(a, Nand:(b, a)), Nand:(Nand:(b, a), b))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(a, Nand:(b, a)), Nand:(b, Nand:(b, a)))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(b, Nand:(a, b)), Nand:(Nand:(a, b), a))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(b, Nand:(a, b)), Nand:(a, Nand:(a, b)))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(b, Nand:(b, a)), Nand:(Nand:(b, a), a))is equivalent to XOR!!! And it's 4
Gates long!
Nand:(Nand:(b, Nand:(b, a)), Nand:(a, Nand:(b, a)))is equivalent to XOR!!! And it's 4
Gates long!
1128012 milliseconds (That’s almost 19 Minutes!!!)
```

Because this is far less efficient I chose to have order not matter in my solution.