Week 4: Evil EVAL

Collaboration: You should work on the problems yourself, before discussing with others, including your cohorts at your cohort meeting. By the Assessed Cohort Meeting, you and all of your cohortmates, should be prepared to present and discuss solutions to all of the assigned problems. In addition to discussing with your cohortmates, you may discuss the problems with any other current CS3102 students you want, and use any resources you want except for any materials from previous offerings of this course or complete solutions that might be available on the web, which are not permitted. **At the end of your assessed cohort meeting, your Cohort Coach will assign one of these problems as a writeup. You may not collaborate on this writeup with your cohort-mates, but you may use notes taken before or during your assessed cohort meeting.**

Problem 1: Finite vs. Infinite Functions

To fully appreciate this week's content, it is critical that you understand the difference between finite and unbounded functions. *TCS Section 1.7* defines a *finite function* to have a fixed-sized input and an *infinite function* to have unbounded input.

Check your understanding of the difference between these two by answering (with proof) the following questions:

– What is the cardinality of the set of all finite functions of the form $\{0,1\}^n \rightarrow \{0,1\}$ for a fixed natural number n (express your answer as a function of n).?

– What is the cardinality of the set of all finite functions with binary inputs?

– What is the cardinality of the set of all infinite functions with binary inputs?

Problem 2: NAND FTW!

We showed that every finite function can be computed by some NAND-Circuit. If NAND-Circuits can do so many things, why don't we use hardware that just implements NAND-Circuits, and why don't we just use the NAND-CIRC programming language?

Name some important ways that actual hardware and software behave differently from NAND-Circuits and the NAND-CIRC programming language. In other words, what are components provided by real hardware and features provided by programming languages that avoid some drawbacks of only implementing NAND-Circuits and the NAND-CIRC language?

Problem 3: Theory vs Practice

This week we saw that NAND circuits (or equivalently the NAND-CIRC programming language) could be used to compute any finite function. We also know that any real computer, since it has finite size/memory capacity, can only compute finite functions. We have a theory of computing that matches anything we can do in practice! Seems like the course should be over then and we should all go to an appropriately socially-distanced beach?

I'm happy to share with you that we still have much computing theory and more models of computing left to explore! What might be the benefits of discussing more models of computing that can be used to implement infinite functions, even though we know those can't be perfectly actualized? To restate this in a more quippy (and therefore more fun) way – *why might it be more practical to use theories that are less like practice*?

Problem 4: EVAL in Python

Download the *eval.py* program. Follow the instructions in the comments, being sure to implement every function marked with "TODO". This program walks you though the implementation of EVAL in Python.

By the time you're finished with this, you'll have built a Python interpreter for the NAND-CIRC programming language and have an interpreter powerful enough to compute any finite function!

Problem 5: Equal to Constant Function (TCS exercise 5.3 and *Defining EVAL* video)

For every $k \in \mathbb{N}$ show that there exists a NAND-CIRC straightline program of no more than $c \cdot k$ lines (where c is a constant) which computes $\texttt{EQUALS}_{x'} : \{0,1\}^k \rightarrow \{0,1\}$ where $\texttt{EQUALS}_{x'}(x) = 1$ if and only if $x = x'$. (Note that a different choice of x' requires a different $\texttt{EQUALS}_{x'}$ function, making this function different than the one in the python programming problem for this week.)

Problem 6: More Dominoes Required

So far, we have identified that the following functions have upper bounds on NAND gates required to implement:

1. n-bit adder: requires no more than $9n$ gates
2. `LOOKUP_k`: requires no more than $4 \cdot 2^k$ gates

The goal of this problem is to estimate the resources required to physically build various circuits. Towards this end we will assume that the bounds given above are exact bounds, rather than upper bounds, on the number of gates required.

The *10,000 Domino Computer* (bonus video) attempted to add two 4-bit numbers. Answer the following questions that were inspired by that video:

a) Assuming adding two n bit numbers together requires approximately $9n$ gates, estimate the average number of dominoes required to build one NAND gate.

b) Roughly how many dominoes would be required to implement `LOOKUP_k`?

c) At a quick glance, I estimate that the dimensions of the 10,000 Domino Computer are about 10×10 meters (or $100\ m^2$). The surface area of the Earth is about $510,000,000\ km^2$. What's using your estimate in the previous part, what's the largest value of k for which `LOOKUP_K` will fit on Earth?

d) Using the dominoes per gate number you derived, give an upper bound on the number of dominoes required to implement any finite function (give your answer as a function of the number of input and output bits).

