PS4: Sugar

Collaboration: You should work on the problems yourself, before discussing with others, including your cohorts at your cohort meeting. 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. Sharing and subsequently submitting any text, code, images, figures, etc. constitutes plagiarism, so make sure all submitted materials are created exclusively by members of your cohort.

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:

1. 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 in terms of n).?
2. What is the cardinality of the set of all finite functions with binary inputs?
3. What is the cardinality of the set of all infinite functions with binary inputs? (*Note: this was done in class, but we phrased the claim differently. Identifying the relationship between this question and the proof from class is sufficient here.*)

Problem 1: Full Adders (Programming)

For this problem, you will be asked to implement several straightline programs in Python for adding numbers together. You must adhere to the rules of straightline programs. In particular, make sure:

– Variables contain only bits.

– Function parameters are only the characters '`0`' and '`1`'.

– Each line may only have a single function and a variable assignment (no nested functions, no conditionals, no loops).

– Variable names may not be reused within the same function.

Unlike with the programming assignment on the previous problem set, syntactic sugar is allowed here. By this, we mean that you may use functions other than NAND, OR, AND, and NOT, so long as those other functions are themselves (eventually) defined in adders.py using just NAND, OR, AND and NOT.

We provide test cases for all functions. If you followed the above rules, and you pass all tests, you can be confident that your implementation is correct.

To complete this problem, download the adders.py program. The comments in that file guide you through the assignment, I recommend reading through all the comments linearly. Everything you must implement is marked with a comment containing "`TODO`".

Problem 3: Full Adders (written)

First, complete the programming problems. They are necessary in helping you to address the written problem below.

Show that for every n there is a NAND-Straightline program to compute ADD_n with at most $9n$ lines where $ADD_n : \{0,1\}^{2n} \rightarrow \{0,1\}^n$ is the function that outputs the sum of two input n-bit numbers (where all inputs and outputs are represented in binary).

This problem is derived from Exercise 4.5 in the textbook. You may find the other parts of Exercise 4.5 in the book helpful, but it is not necessary to solve this problem using those steps.

Problem 4: Counting Gates

In class we discussed the function XOR $: \{0,1\}^2 \rightarrow \{0,1\}$ defined such that XOR(a,b) is 1 provided exactly one of a and b is 1. For this question I have implemented an n-bit XOR function, which we denote XOR$_n$ $: \{0,1\}^n \rightarrow \{0,1\}$, which returns 1 exactly when an odd number of its inputs are 1. My implementation will use a subroutine for doing XOR with one fewer bit (i.e. it will invoke an implementation of XOR$_{n-1}$), then perform XOR on that resulting bit with the first input bit. We will have a base case of XOR$_2$, which is just XOR.

As a straightline program, XOR$_n$ will be implemented as follows:

```
def XORn(x1, ..., xn):
    rest = XORnminus1(x2, ..., xn)
    return XOR(rest, x1)
```

Suppose I have similarly implemented XOR$_{n-1}$, XOR$_{n-2}$, etc. (and all other dependent subroutines). Show using induction that the number of NAND gates needed to represent a circuit for XOR$_n$ is no more than $5n$ gates.

Problem 5: INCADD

We define the *INCADD*-straightline program model to be similar to *AON*-straightline, except the operations used are the single-bit increment and add operations defined by:

```
def INC(a):                      def ADD(a,b):
    return (a + 1) % 2               return (a + b) % 2
```

The % operator is modulo (remainder after division) and behaves as the % operator does in Java/Python. So, for example `(1 + 0) % 2 = 1` and `(1 + 1) % 2 = 0`.

Prove that the *INCADD*-straightline program model is not universal. Then suggest one operation (besides NAND or NOR) to add to the model so that it becomes universal (and prove that it is).

Problem 6: Self Reflection/Peer feedback

If you're in a cohort, complete this peer feedback form. Your responses will be aggregated, anonymized, and then shared (unedited) with the cohort-mate that it's about, so please be tactful. The feedback given here will not directly impact anyone's grade. It is intended just to facilitate conversations about duties and expectations in your cohort, to notify the course staff if something is amiss, and to document issues in case of future problems.

If you're not in a cohort, complete this self-reflection.