Exam 1 Comments

Boolean Circuits

For these questions, we assume the following logical functions with their standard meanings:

> *NOT*(a): *NOT*$(0) = 1$, *NOT*$(1) = 0$.
> *OR*(a, b): *OR*$(0, 0) = 0$, otherwise *OR*$(a, b) = 1$.
> *AND*(a, b): *AND*$(1, 1) = 1$, otherwise *AND*$(a, b) = 0$.

1. Give a simple description (which could be just the name of a well known function) of the function defined by the code below:

```
def MYSTERY(a, b):
    v1 = NAND(a, b)
    return NAND(v1, v1)
```

From this table we can see that the function returns 1 if both a and b are 1, 1s so it is the familiar AND function.

a	b	`v1`	*output*
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Another way to see this is that $NAND(x, x) = NOT(x)$, so $NOT(NAND(a, b)) = NOT(NOT(AND(a, b)))$, which simplifies to $AND(a, b)$.

2. Define *XOR* using only *NOT*, *OR* and *AND*, where *XOR*$(0, 0) =$ *XOR*$(1, 1) = 0$ and *XOR*$(0, 1) =$ *XOR*$(1, 0) = 1$.

There are many ways to do this. Here are two:

```
def XOR(a, b):
    both1 = AND(a, b)
    both0 = NOT(OR(a, b))
    both_same = OR(both1, both0))
    return NOT(both_same)
```

```
def XOR(a, b):
    dif_or_0s = NOT(AND(a, b))
    dif_or_1s = OR(a, b)
    both_dif = AND(both1, both0))
    return both_dif
```

Countability

For these problems, you may use any results that were proven in class or on a problem set in your proof (without needing to prove them).

3. Prove that the set of all fish in the sea is *countable*. (For purposes of this question, you can assume the "sea" in question is the Mediterranean Sea, and *fish* has its conventional meaning.)

This problem should remind you of the question on PS1 asking about the set of Python programs that can run on your laptop. A finite set is countable. The number of fish in the Mediterranean Sea must be finite (for lots of obvious reasons that don't need to be stated, but if you are not convinced we know that the volume of the sea is finite, and each fish has non-zero volume, so it can contain at most a finite number of fish), so must be countable.

4. Prove that the set of the even natural numbers (i.e., $\{0, 2, 4, 6, \ldots\}$) is *countably infinite*.

One way to show this is to find a bijection between $\mathbb{N}$ and the even numbers. A simple one is $g(x) = 2x$. This maps $0 \leftrightarrow 0, 1 \leftrightarrow 2, 2 \leftrightarrow 4, 3 \leftrightarrow 6, \ldots$. Each natural number maps to a unique even since there are no two $a, b \in \mathbb{N}$ such that $2a = 2b$ where $a \neq b$.

A harder approach (in this case, but usually a good idea if you are stuck finding a bijection) is to show *countable* and *infinite* separately. The evens are countable since they are a subset of $\mathbb{N}$, which we already know is countable and a any subset of a countable set is countable.

We can show the evens are infinite since there is no maximal even number. Towards a contradiction, assume there is some maximal even number k. We can create a greater even number, $k + 2$. Thus, we have a contradiction and there is no maximal even number.

5. Prove that the set of all directed graphs (as defined below) is *uncountable*.

Definition 1 (Directed Graph) A *directed graph* $G = (V, E)$ consists of a (possibly infinite) set V of vertices and a (possibly infinite) set E of edges. Every edge is an ordered pair of two distinct elements of V.

In PS1 (Problem 5) and Class 6 we proved that the set of all undirected graphs is uncountable. Since each undirected graph can be mapped to a directed graph (e.g., just make all the edges in the direction from lower $\rightarrow$ higher), we know there are at least as many directed graphs as there are undirected graphs. Since we already know the undirected graphs are uncountable, this shows the directed graphs are also uncountable.

Proofs with Definitions

Here we define the Counting Numbers, similarly to the definition of Natural Numbers you saw on Problem Set 1:

Definition 2 (Counting Numbers) We define the *Counting Numbers* as:

1. **1** is a Counting Number.
2. If n is a Counting Number, **S**(n) is a Counting Number.

6. Prove that the cardinality of the set of all *Counting Numbers* (as defined above) is *countably infinite.*

Similarly to problem 4 (in this exam), we can prove a set is countably infinite either by showing a bijection with $\mathbb{N}$, or by separately showing the set is infinite and the set is countable.

Approach 1: Bijection

We can map our Counting Numbers to $\mathbb{N}$ by simply counting the number of **S**s in the number. Accounting to the base case of the definition, **1** is a Counting Number. This has 0 **S**s in it, so maps to the natural number 0. The recursive case produces a new Counting Number by adding one **S** to a previous one, and this is the only way to create new Counting Numbers. So, we can map any Counting Number to a $\mathbb{N}$ by the following bijection: $0 \leftrightarrow \mathbf{1}, 1 \leftrightarrow \mathbf{S}(\mathbf{1}), 2 \leftrightarrow \mathbf{S}(\mathbf{S}(\mathbf{1})), \ldots$.

Approach 2: Infinite and Countable

The Counting Numbers are *infinite* since the recursive rule can always produce a new Counting Number. We can prove by contradiction that there is no last Counting Number, since if there were some last Counting Number x, we can produce a new one **S**(x).

The Counting Numbers are *countable* since we can count the number of applications of the recursive rule needed to produce any counting number x, starting from the base **1**.

Computing Models

7. Prove that Boolean circuits using the gateset $\{MAJ, NOT, ZERO\}$ is equivalent to Boolean circuits using the gateset $\{MAJ, NOT, ONE\}$. ($ZERO$ is the constant gate that outputs `0` for any input, and ONE is the constant gate that outputs `1` for any input. MAJ is the majority of three inputs gate you are familiar with from PS3.)

Since the only difference between the two gatesets if the first one includes $ZERO$ (but not ONE and the second one includes ONE but not $ZERO$, we can show they are equivalent by showing how to produce ONE using the first gateset, and produce $ZERO$ using the second gateset:

$(\Longrightarrow)$ $ONE = NOT(ZERO)$

$(\Longleftarrow)$ $ZERO = NOT(ONE)$

8. Prove that Boolean circuits using the gateset $\{MAJ, NOT\}$ is equivalent to Boolean circuits using the gateset $\{NAND, XOR\}$.

Impossible!

We know from PS3, slack, and class that $\{MAJ, NOT\}$ is ***not universal***, but since $\{NAND, XOR\}$ includes $NAND$ it clearly is universal. They cannot be equivalent since what it means to be not universal is that it cannot compute some function that a universal gate set can compute.

9. Prove that *AON-CIRC*, straightline programs composed of *AND*, *OR*, and *NOT* operations is equivalent to *MOP-CIRC*, straightline programs composed of the plus, multiply, and constant one operations defined by:

```
def PLUS(a, b):
    return (a + b) % 2


def MULT(a, b):
    return (a * b) % 2


def ONE(a, b):
    return 1
```

The `%` operator is modulo (remainder after division). So, for example `(0 + 1) % 2 = 1`, `(1 + 1) % 2 = 0`, and `(1 * 1) % 2 = 1`.

MOP-CIRC in terms of AON-CIRC:

`PLUS(a, b) = AND(OR(a,b), NOT(AND(a, b))`	`PLUS` is `XOR`
`MULT(a, b) = AND(a, b)`	`MULT` is `AND`
`ONE(a, b) = OR(OR(a, b), NOT(OR(a, b)))`	Always outputs **1** ($a \vee NOT(a)$)

Since we already know *AON-CIRC* is universal, we could just argue that *MOP-CIRC* is a regular Boolean circuit so can't be more powerful than universal. Alterantively, we could show how to define *AON-CIRC* using *MOP-CIRC*:

`AND(a, b) = MULT(a, b)`	`AND` is `MULT`
`NOT(a) = PLUS(a, ONE(a, a))`	`XOR(a, 1) = NOT(a)`
`OR(a, b) = PLUS(PLUS(a, b), MULT(a, b))`	Lots of alternatives