Self-Assessment Midterm - Solutions

1. [10 points] Better Buses

Recall the UTS buses from Week 1. To celebrate the student Covid positivity rate dropping below 1%, the displays are being upgraded. The new displays are 128 pixels wide and 32 pixels tall. Whereas before pixels could be either orange or black, now each pixel can be in one of three states: orange, green, or black.

Answer these questions about the length of binary strings necessary to represent the bus display.

a) A simple encoding scheme uses two bits to represent each state: `00` is black, `01` is orange, and `11` is green. Assuming all configurations are represented with the same number of bits, what is the number of bits required to represent all configurations using this representation? Justify your answer by demonstrating a bijection between all strings of the length you indicate, and all possible configurations of the display.

b) The above encoding is inefficient. Describe a more efficient encoding and argue that it is the most efficient encoding that can represent all possible configurations with a fixed-length bitstring.

Comments:

The display is 128×32 pixels. Since these are convenient powers of 2, we know there are $2^7 \times 2^5 = 2^{12} = 4096$ total pixels.

a) The simple encoding scheme uses two bits to represent the state of each pixel. So, to represent the full display we need 2×2^{12} bits (8192 bits). There is a simple bijection between the strings in $\mathtt{0},\mathtt{1}^{8192}$ and the display configurations. For any string that is of the form $\{\mathtt{00},\mathtt{01},\mathtt{11}\}^{4096}$ the string maps to the configuration where the i^{th} pixel has color based on the $2i$ and $2i+1$ bits in the string, mapping from `00` to black, `01` to orange, and `11` to green. If the string is not of the form $\{\mathtt{00},\mathtt{01},\mathtt{11}\}^{4096}$ it is an invalid configuration and the bus should display "Out of Service" and return to the bus center.

Grading notes: because the question stated “justify your answer by demonstrating a bijection” we would have liked to see answer that either provide a bijection (which means, mapping some of the bitstrings of the given length to invalid configurations, or explain that it is not actually necessary to show a bijection to demonstrate that all configurations can be represented, but sufficient to show a injective mapping (that is, that each valid display configuration corresponds to a unique bitstring). Since this was meant to be a first warm-up question, though, we would have given full credit for any answer with the correct number of bits.

b) This is a tricky question since we have to map the 3^{4096} possible configurations to a bitstring. The minimum length bitstring that can cover all possible configurations is $\lceil \log_2(3^{2^{12}}) \rceil$ (since the actual value of $\log_2(3^{2^{12}}) \approx 6492.006402953855719234514714410256419880199813908402423626762873 0...$ (https://www.wolframalpha.com/input/?i=log_2%283%5E%282%5E12%29%29) this means we need 6493 bits. This makes about 1.9×10^{1954} (https://www.wolframalpha.com/input/?i=2%5E6493+-+3%5E4096) of the bitstrings correspond to invalid configurations (which seems like a lot of wasted information, but without the 6493rd bit there would be some configurations that could not be represented).

The simplest way to map the bitstrings to configurations would be to just convert a base 3 representation of a configuration (e.g., 0 = black, 1 = orange, 2 = green) to a base 2 number using a standard base conversion. Then, the inverse of that conversion would convert the bitstring to a base 3 number representing that actual configuration. We talked in class (see the video in collab) about other more intuitive ways of mapping the 3-value configurations to bitstrings, and its connection to the musical circle-of-fifths.

Grading notes: This problem raises lots of interesting subtleties in binary representations, but was a bad question to ask in an "exam", especially as the first question. It was added and modified hastily in preparing the midterm, without thinking through the difficulties sufficiently, and hopefully would not have survived the quality control review we would have done if this were a graded exam. Sorry if this question cause you unnecessary distress or shattered your confidence for the rest of the exam!

Countability

2. [10 points] It's about to get odd

Prove that the set of the odd natural numbers (i.e., $\{1, 3, 5, 7, \ldots\}$) is *countably infinite*.

Comments:

Show a one-to-one mapping between the set of odd numbers and the natural numbers. For example, $0 \Leftrightarrow 1$, $1 \Leftrightarrow 3$, $2 \Leftrightarrow 5$, ... This could be expressed as $f(n) = 2n + 1$ for all $n \in \mathtt{N}$ and $f^{-1}(n) = \lfloor (n/2) \rfloor$ for all $n \in \{1, 3, 5, 7, \ldots\}$.

3. [10 points] Toes

Prove that the set of all toes on all the feet of all current UVA students is countable.

Comments:

The number of UVA students is finite, and each student has a finite number of toes. So, the set of all toes is finite, and all finite sets are countable. (As mentioned in class, we are nearly always dealing with infinite sets, but it is always a good idea to first think if a set is finite. You shouldn't be surprised if there is a question on the final exam that can also be solved easily by determining that a set is finite.)

Uncountability

4. [10 points] Fingers

The Cantorvanian creatures from the planet Cantorvania have only one hand, but it has a countably infinite number of fingers. A human glove has only 5 holes for fingers, so when a Cantorvanian wears one it will put many fingers into the same finger hole. To wear a glove Cantorvanians do not need to put their fingers into the glove's holes contiguously. For example, fingers 4 and 7 may go into hole 2, with finger 5 going into hole 5. To It is ok if some glove holes have no fingers, but all fingers must be in a hole.

Show that there is an uncountable number of ways for a Cantorvanian to wear a human glove.

Comments:

The easiest strategy for this question is to find a way to represent the Cantorvanian glove configurations using a representation we already know is uncountable. In finding this mapping, we only need to find

a one-to-many representation that maps every element of the set that is known to be uncountable to at least one Cantorvarian glove configuration. This would show that the cardinality of the set of Cantorvarian glove configurations is at least as large as the cardinality of the known uncountable set, hence it must be uncountable.

A simple mapping is to the infinite bitstrings, which we have already proved to be uncountable. Instead of considering a five-finger glove, let's consider a two-finger mitten where the "thumb" is represented as `0` and the four fingers are represented as `1`. Then, we can map any configuration of a Cantorvanian's fingers into the glove as a bitstring. For the example, fingers 4 and 7 are in hole 2 and finger 5 is in hole 5, the bits at the corresponding positions would be `1` to represent those fingers in non-thumb holes. If fingers 1, 2, and 3 are in hole 1 (the thumb), and finger 6 is in hole 23, this would be represented by `0001111`....

Thus, we can represent any configuration of the Cantorvanian's fingers as an infinite binary string. But, this isn't quite the mapping we need to show uncountability — we need to show the reverse mapping: that every infinite binary string corresponds to a different configuration of the Cantorvanian's fingers. The reverse mapping is simple: for the i^{th} finger, if the i^{th} is a `0`, the finger is in hole 1; if it is a `1`, the finger is in hole 2. Note that the statement in the problem that it is okay to have some glove holes with no fingers makes this a valid mapping (even if it may be uncomfortable for the Cantorvanian to put so many fingers in the same hole — of course, there is no mapping from the countably infinite number of fingers the Cantorvanian has to the finite number of glove holes that does not result in an infinite number of fingers in at least on of the glove holes).

Grading notes: we expect many answers that provide the mapping in the wrong direction here — just showing that every configuration of the glove can be represented by a bitstring isn't sufficient to show the configurations are uncountable, we need to show that each of the infinite bitstrings maps to a unique configuration. Since uncountablity is a "bigger than" property, though, we don't need to find a one-to-one mapping (which would be quite challenging), we just need to show a mapping that maps every infinite bitstring to a different glove configuration. Showing the mapping without this explanation, though, would be worth substantial partial credit.

Counting Gates

5. [10 points] n-bit XOR

In class we discussed the function $\texttt{XOR} : \{0,1\}^2 \rightarrow \{0,1\}$ defined such that $\texttt{XOR}(a,b)$ is 1 provided exactly one of a or b is 1. For this question we have implemented an n-bit `XOR` function, which we denote

$$\texttt{XOR}_n : \{0,1\}^n \rightarrow \{0,1\}$$

for $n \geq 2$. It returns 1 exactly when an odd number of its inputs are 1. Our implementation uses a subroutine for doing `XOR` with one fewer bit (i.e., it will invoke an implementation of $\texttt{XOR}_{n-1}$), then perform `XOR` on that resulting bit with the first input bit. We will have a base case of $\texttt{XOR}_2$, which is normal 2-bit `XOR`.

As a straightline program, $\texttt{XOR}_n$ for all $n > 2$ will be implemented as follows:

```
def XOR_n(x1, ..., xn):
    rest = XOR_n_minus_one(x2, ..., xn)
    return XOR(rest, x1)
```

Show using induction that the number of NAND gates needed to represent a circuit for $\texttt{XOR}_n$ is no more than $5n$ gates (hint: XOR requires 4 NAND gates).

Comments:

The most important insight for this problem is that XOR requires 5 NAND gates to implement, and each "iteration" of this recursive definition requires an additional XOR. From this intuition, we can see that $\texttt{XOR}_n$ requires n XOR gates, or $5n$ NAND gates. The problem requests we employ a proof by induction, so let's do that.

Theorem: let $G(n)$ represent the number of NAND gates required to implement $\texttt{XOR}_n$. We want to show: $\forall n \in \mathbb{N}.G(n) \leq 5n$.

Inductive Hypothesis: $P(n) : G(n) \leq 5n$. It is possible to implement $\texttt{XOR}_n$ using $5n$ or fewer gates.

Base Case: ($P(2)$) We select a base case of $n = 2$, which is just a standard 2-bit XOR (we select a base case of 2 since it's unclear how we would meaningfully define XOR for 0 or 1 bit of input). As we mentioned in class, a 2-bit XOR requires 5 NAND gates. Thus $G(2) = 5 \leq 5 \cdot 2 = 10$, and so our base case holds.

Inductive Step: For the inductive step, we need to show $P(n) \implies P(n+1)$ for all $n \geq 2$.

We show that if $G(k) \leq 5 \cdot k$ then $G(k+1) \leq 5(k+1)$. To do this, we note that the number of NAND gates required to implement $\texttt{XOR}_{k+1}$ is given by $G(k+1) = G(k) + 5$ since it requires computing $\texttt{XOR}_k$ followed by XOR. Therefore we know:

$$\begin{aligned} G(k+1) &= G(k) + 5 \\ G(k+1) &\leq 5k + 5 \\ G(k+1) &\leq 5(k+1) \end{aligned}$$

So our inductive step holds, and we can conclude from induction that $\texttt{XOR}_n$ requires no more than $5n$ NAND gates to implement.

Universality

6. [10 points] XOR, AND, 1

Show that the operations XOR, AND and 1 together are a universal gate set.

Comments:

For a collection of gates to be universal we need to show that we can use them to implement any finite function. The easiest strategy to to employ to achieve this is typically going to be to use this set of gates to implement another set of gates that we know to be universal (typically either AND+OR+NOT or NAND). I will show XOR, AND and 1 together are a universal gate set by using them to implement NAND.

Note that $\texttt{XOR}(a, 1) = \texttt{NOT}(a)$, since $\texttt{XOR}(0, 1) = 1$ and $\texttt{XOR}(1, 1) = 0$. Therefore we can use our gate set to implement NOT, since we have both of XOR and 1. Since our gate set also has AND, we can implement $\texttt{NAND}(a, b)$ as $\texttt{NOT}(\texttt{AND}(a, b))$. Since NAND is universal, and it can be implemented with XOR, AND and 1, those gates are universal as well.

Incremental Universality

7. [10 points] Increment and Add

Consider the *INCADD*-straightline language where programs must be straightline code using these two operations:

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

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

The `a` and `b` inputs are a single bit. The `%` operator (as in Java, Python, and Rust) is modulo (remainder after division). So, for example, `(1 + 0) % 2 = 1` and `(1 + 1) % 2 = 0`.

Either show that straightline programs using the operations `INC` and `ADD` is an equivalent computing model to `AON` straightline programs, or prove that it is not equivalent.

Comments

In this case, the gate set is not universal. To see this, first note that $\texttt{INC}(a)$ is equivalent to $\texttt{NOT}(a)$ and that $\texttt{ADD}(a, b)$ is equivalent to $\texttt{XOR}(a, b)$. This means that asking if `INC` and `ADD` are equivalent to `AON` is equivalent to asking whether `NOT` and `XOR` are universal, which is an equivalent gate set to that in Week 3 Problem 6.

To see that this is equivalent, we can implement `NOT` from $\texttt{XOR}, 0, 1$ and we can implement 0 and 1 from `NOT` and `XOR`.

To get `NOT` from $\texttt{XOR}, 0, 1$ we can implement $\texttt{NOT}(a) = \texttt{XOR}(a, 1)$.

To get 0 from `NOT` and `XOR` we can implement $0 = \texttt{XOR}(a, a)$.

To get 1 from `NOT` and `XOR` we can implement $1 = \texttt{NOT}(0)$.

Since we've now shown that $\texttt{INC}, \texttt{ADD}$ are equivalent to a non-universal set of gates $(\texttt{XOR}, 0, 1)$, they cannot be equivalent to `AON`.

Asymptotics

8. [10 points] Big Omicron

Recall these definitions of asymptotic operators from class:

Definition 1 (O) A function $f : \mathbb{N} \to \mathbb{R}_+$ is in the set $O(g(n))$, defined for any function $g : \mathbb{N} \to \mathbb{R}_+$ if and only if there exist two constants $c \in \mathbb{R}_+, n_0 \in \mathbb{N}$ such that: $\forall n > n_0 . f(n) \leq cg(n)$.

Definition 2 (Ω) A function $f : \mathbb{N} \to \mathbb{R}_+$ is in the set $\Omega(g(n))$, defined for any function $g : \mathbb{N} \to \mathbb{R}_+$ if and only if there exist two constants $c \in \mathbb{R}_+, n_0 \in \mathbb{N}$ such that: $\forall n > n_0 . f(n) \geq cg(n)$.

Definition 3 (Θ) A function $f : \mathbb{N} \to \mathbb{R}_+$ is in the set $\Theta(g(n))$, defined for any function $g : \mathbb{N} \to \mathbb{R}_+$ if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

Let $f(n) = 13n$ and $g(n) = 7n^2$, which of the following are true? Support your answer to each part with a convincing argument that uses the definitions above.

a. $f \in O(g)$

Comments

This is True. To show this we need to identify a choice of n_0 and c to satisfy $\forall n > n_0 . 13n \leq c \cdot 7n^2$.

To do this, I will arbitrarily select $c = 1$ and derive a satisfactory n_0:

$$\begin{aligned} 13n &\leq c \cdot 7n^2 \\ 13n &\leq 7n^2 \\ 13 &\leq 7n \\ \frac{13}{7} &\leq n \end{aligned}$$

This inequality holds true for any value of $n \geq \frac{13}{7}$, and so we can select $n_0 = 2$

b. $f \in \Omega(g)$

Comments

This is False. To demonstrate this we need to show that there is no choice for c and n_0 to satisfy $\forall n > n_0 . 13n \geq c \cdot 7n^2$. To do this, it's sufficient to show that for any choice of $c > 0$ we can find an arbitrarily large value of n_0 such that $13n < c \cdot 7n^2$. To demonstrate this, we will solve the following inequality for n:

$$\begin{aligned} 13n &< c \cdot 7n^2 \\ 13 &< c \cdot 7n \\ \frac{13}{7} &< cn \\ \frac{13}{7c} &< n \end{aligned}$$

This tells us that no matter what value of c we select (so long as it's positive) $f(n)$ will be less than $c \cdot g(n)$ for any value of $n > \frac{13}{7c}$, and so it is not the case that $f \in \Omega(g)$.

c. $f \in \Theta(g)$

Comments

This is False, because $f \notin \Omega(g)$

Asymptotics

9. [10 points] Omicron and Agemo

If $f \in O(g)$ is $g \in \Omega(f)$?

Use the definitions (in the previous page) to either prove that for any functions f and g, $f \in O(g) \implies g \in \Omega(f)$, or show that there is at least one case where $f \in O(g)$ but $g \notin \Omega(f)$.

Comments

To answer this question we need to determine whether being able to find c and n_0 to satisfy $\forall n > n_0.f(n) \leq c \cdot g(n)$ implies we can find c' and n'_0 to satisfy $\forall n > n'_0.g(n) \geq c' \cdot f(n)$. To begin, let's just naively start with the former inequality and check if we can derive the latter.

$$\begin{aligned} f(n) &\leq c \cdot g(n) \\ c \cdot g(n) &\geq f(n) \\ g(n) &\geq \frac{1}{c} f(n) \end{aligned}$$

The tells us that for any choices of c and n where $f(n) \leq c \cdot g(n)$ we can find a $c' = \frac{1}{c}$ such that $g(n) \geq c' \cdot f(n)$, and so if $\exists c > 0.\exists n_0 > 0.\forall n > n_0.f(n) \leq c \cdot g(n)$ we can conclude $\exists c' > 0.\exists n'_0 > 0.\forall n > n_0.g(n) \geq c \cdot f(n)$ by selecting $c' = \frac{1}{c}$ and $n'_0 = n_0$. Thus the statement is True.