CS70 Summer 2018 — Solutions to Homework 4

Sung Hyun Harvey Woo, SID 24190408, `CS70`

July 13, 2018

Collaborators: Dylan Hwang (dylanhwang@berkeley.edu), Allison Wang (awang24@berkeley.edu), Michael Hillsman (mhillsman@berkeley.edu)

Sundry

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up. — Sung Hyun Harvey Woo

1. Countability Introduction

(a) Yes.
Given all real numbers from 0 to 1 exclusive (dubbed set A), and all real numbers from 0 to infinity exclusive (dubbed set B), we can establish that they have the same cardinality if we show that there is an explicit bijection. We can take every element of the real numbers from A and use the equation $\frac{1}{A_i} - 1$ where i is all real numbers such that $0 < i < 1$ and map it to an element in B. Since we know that all real numbers in A is a unique element, all mapping to B in the form of $\frac{1}{A_i} - 1$ will be unique as well, so we know that it is one to one from A to B. And we we have a value B_i in set B, we know that the preimage of that number will also be unique since there is only one value in A such that $(B_i + 1) * k = 1$, for a k in A. Since the mapping from A to B is one to one and the mapping from B to A is one to one, by the Cantor-Bernestein theorem we know that A and B have a bijection therefore all real numbers from 0 to 1 exclusive , and all real numbers from 0 to infinity exclusive have the same cardinality.

(b) Yes.
We know that each string has a finite length and that the set of English characters is finite. I will assume that the set of English characters include a to z, A to Z, and punctuation. Regardless of the exact definition of English characters, it will be finite. When it comes to showing if the set of English strings are countable, we can do this by enumerating the strings. We can have a set of all natural numbers, N, and enumerate the strings according to N, starting at 1. We can organize each string by length and list them out, mapping them to the natural numbers. We could first list out all strings possible that is of length 1, then all strings possible that is of length 2, etc. Since we know each string has a finite length, and that the set of English characters are finite, we can enumerate them (producing a bijection between natural numbers and the set of strings), proving that the set of English strings are countable.

(c) Yes, the answer does not change, It is still countable.
Continuing from part b, we now assume that the alphabet is countably infinite (dubbed set A). The answer does not change in this case. even if we have an infinite alphabet, we know that it is countable. We can use the method from part b and prove that there will be a bijection between the set of strings and natural numbers if we can show that it can be enumerated. We can enumerate this set of strings by restricting the number of characters used in each string, and also restricting the length of the string. Let's assume that set A starts with a, b, c,... etc. Then we would first enumerate all the strings that have characters (a) and length 1, then strings that have the characters (a b) and length 2, then (a to c) with length 3, and on, we know that both criteria for enumeration is finite, we know that we can have a bijection between the set of strings and natural numbers and any given string will exist in the enumeration since alphabet is countable, and the length is finite and countable.

2. Counting Cartesian Products

(a) Countable.
We have already seen that $\mathbb{N}$ X $\mathbb{N}$ is countable in the example of the spiral in a 2 dimensional plane. We can do that same here, and assume that (a, b) is a coordinate on a 2 dimensional plane. Since each a and b value are natural numbers, we can put them on a graph and know that each pair will not repeat, since it would mean the set contains multiple instances of the same pair. A and B are both countable, therefore there is a bijection between each respective

set and natural numbers. We can take each pair in the 2D plane of $\mathbb{N}$ X $\mathbb{N}$ and map to directly to the same point in A X B since both A and B can be enumerated using natural numbers. We already know that $\mathbb{N}$ X $\mathbb{N}$ is countable, and we can enumerate it such that the set $\mathbb{N}$ X $\mathbb{N}$ is unique to each pair. We will have a bijection between $\mathbb{N}$ X $\mathbb{N}$ and A X B, therefore we know that A X B is countable.

(b) Countable. Proof by induction
Since we have a finite number of countable sets (n sets), we can use induction to show that it is countable.

Base Case : $n = 2$
Part 2.a above showed that A X B would be countable by utilizing a bijection between $\mathbb{N}$ X $\mathbb{N}$ and A X B since both sets A and B are countable. The bijection proved that A X B would be countable, thus A_1 X A_2 is countable.

Inductive Hypothesis: $n = k$
Assume that for some k, the cartesian product of $A_1, A_2, A_3, ...A_k$ is countable.

Inductive Step : $n = k + 1$
We have the cartesian product of $A_1, A_2, A_3, ..., A_{k+1}$. We know that for some k, the cartesian product of $A_1, A_2, A_3, ..., A_k$ is countable. Since we know from the 2.a, that if A and B are countable sets, the cartesian product of A and B is also countable (by mapping it to $\mathbb{N}$ X $\mathbb{N}$), we can have the cartesian product of $A_1XA_2XA_3X...XA_k$ and A_{k+1}, giving us $A_1XA_2XA_3X...XA_{k+1}$ that we know is countable.

(c) Uncountable.
We can show that the cartesian product of an infinite number of countable sets (dubbed set A), is only countable if there are a finite number of sets that have more than one element, and the rest have only one element. If a set has only one element, every element of the cartesian product of $B_1, B_2, ...$ will have to contain this one element. If there are a finite number of sets that have more than one element, we could essentially "fix" the elements of the one element sets, and produce the rest of the elements using the sets that have more than one element. We can prove this is the condition necessary for A by showing that it is uncountable if there were infinitely many sets with more than one element.
Given that there are infinitely many sets with more than one element, you can prove that set A would be uncountable by using the diagonalization proof. We assume that even with infinitely many sets with more than one element, set A is countable, which means that it can be enumerated in a list (with a bijection to natural numbers). So we have a list of elements as listed below:

$$(a_1, a_2, a_3, ...), (b_1, b_2, b_3, ...), (c_1, c_2, c_3, ...) \tag{1}$$

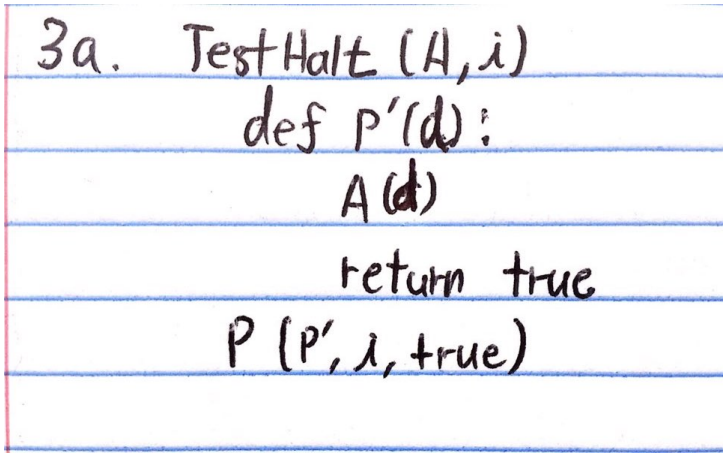
where the a_1 would be an element from set 1, a_2 from set 2, and on. We are assuming that it is countable, so we know that the enumeration is valid. Now, we assume an element of set A (dubbed A_d) that constructs a valid cartesian product, using the 1st element in the first cartesian product listed, the 2nd element in the second cartesian product listed etc. Since we know that these sets have at least one other element other than the one that it currently has, we can modify A_d to A_{di} so that for each set it contains an element other than the one that A_d has for each respective set. A_d cannot exist in this enumeration because if it did, let's say

as the k^{th} element of the list, the k^{th} value of A_d would correspond to the k^{th} value of A_{di} which we know is a contradiction since we defined A_{di} to be contain a different element than A_d. Thus, if set A has infinitely many sets with more than one element, it would be uncountable.

Therefore, the condition the cartesian product of an infinite number of countable sets to be countable is that it has a finite number of sets that have more than one element, and the rest have only one element or if there are any empty sets, since an empty set would result in set A to be empty as well.

3. Impossible Programs

(a) Does not exist.
We can show that a program P does not exist if we can use reduction and show that it is at least as hard at the Halting problem. We first assume that a program P exists, and if we can show that using program P, we can solve the Halting problem, we show that program P doesn't exist because it would in turn produce a solution to the Halting problem, which we know does not exist. Consider program P and the pseudocode below.



Given a program P, we can set it up such that if P exists, TestHalt would also exist. The TestHalt function takes in any program A and any input i, and runs P(P', i, true). The function P' takes in any input d, runs the function A using that input. If it halts, it will go onto the next step, return true, however if it loops, the return true statement will never be reached. When we run P(P', i, true), if P'(i) returned true(halts), the P function will return true, indicating that the function A(i) has halted. If A(i) loops forever, P' will never return, therefore the function P will return false, since it is a function that knows whether a function F with an input x outputs a y. Thus, using program P, we can construct a TestHalt function, which we know cannot exist. Therefore, program P cannot exist.

(b) Does not exist.
We can prove that P does not exist if we can show that there are cases where it will not output the right answer, using contradiction. We assume that a program P exists and we construct a function F(x), which will halt if P(F) returns false, and loop forever if P(F) returns true. This way, if we pass in F(F) to this program P, if P(F) returns false (that it will loop forever), then F(F) will halt, whereas if P(F) returns true(that it will halt), F(F) will simply loop forever.

This would be a contradiction similar to the Turing(Turing) proof mentioned in the lecture. Therefore this program P cannot exist, because it will never return the right answer.

(c) Does not exist.
For this problem, we can come back to the TestHalt method. We can prove that program P does not exist if we prove that we can solve the Halting program using P.

```
3c. TestHalt (A, i)
    F'(d):
        return true
    G'(d):
        A(i)
    return P(F', G')
```

If we have a program P that can show whether or not two functions halt at all the same inputs, and false if it does not, we can use it to make a TestHalt method. The F' function is defined as a function that always halts, regardless of its input, whereas G' simply runs A(i), and arbitrary program with an arbitrary input i. If we run P(F',G'), both F' and G' ignore the input, therefore if A(i) halts, for the first case, it will halt for every other input since both functions ignores the input, therefore P(F',G') returns true if A(i) halts. If A(i) were to loop forever, P(F',G') would return false, since it sees that F' will always halt regardless of the input, but G' will loop forever regardless of the input. Therefore, by setting F' as a function that halts regardless of the input, and G' such that it ignores its input, we can create a TestHalt function that will return true if A(i) halts, and false if A(i) loops forever.

4. Printing All x where M(x) Halts

(a) We can construct a program P, as shown below in pseudocode.

```
P(M):
    setM = []
    for x=1 to infinity:
        setM.append(M(x))
        run all M in setM for 1 loop.
        for all M in setM:
            if M halted:
                print n  // input of the M function M(n)
                setM.remove(M)
```

We start with a program P that takes in a program M. It creates an empty set (SetM) that is meant to contain different implementations of M(x) where x is from 1 to infinity. It starts a for loop starting at 1 and adds M(1) to SetM. Then it runs all M in setM for one loop and then checks if any M in SetM has halted. If it has halted, it prints out the input of the M for that particular M (dubbed n) and then removes that M function from the set. Any M in the set that has not halted stays inside the set. This way, the M that is kept in the set will continue to loop until it halts, printing out it's n value, or if it loops forever, the n value will never print, satisfying all the requirements of program P.

(b) It is impossible to solve part 4.a when it has to be put in lexicological order, since doing so would solve the halting problem. Given a single input k, in order for the outputs to be in lexicological order, we need to know if it is the next print statement possible or if there are any outputs that are lexicologically in front of k, but have not printed yet. In order for M(k) to print, it needs to wait until it knows that it is the next one in the lexicological list. However, let's assume that there is a M(k-1) that takes a very long time, but not loop forever. M(k) does not know if it loops forever or if it takes a long time. In order to output all outputs in lexicological order, M(k) must do one of the two things. Either it will simply output M(k) meaning that it won't be in lexicological order, or it will have knowledge of whether or not a previous lexicological output will halt or not.
In other words, Having an output in lexicological order would prove that you can solve the Halting problem. The example, M(k), can only print if it knows that it is the next output in line, therefore if there exists an M function that can output the results in a lexicological order, it means that there exists a function that can determine whether or not a function (M(k-1) for example) will halt, before it actually halts, and can determine whether or not a function will halt or loop forever, since it gives M(k) the knowledge to print at the time when it is in lexicological order. Since a solution to the Halting problem does not exist, this program would be impossible to create.

5. Counting, Counting, and More Counting

(a) $\binom{n+k}{n}$

(b) (a) $\binom{52}{13}$
(b) $\binom{48}{13}$
(c) $\binom{48}{9}$
(d) $\binom{39}{7}\binom{13}{6}$

(c) $\frac{104!}{2^{52}}$

(d) $\sum_{x=50}^{99}\binom{99}{x}$

(e) (a) $7!$
(b) $\frac{6!}{3!}$
(c) $\frac{7!}{4!}$
(d) $\frac{7!}{2!2!}$

(f) (a) $5!$

(b) $\frac{6!}{2!}$

(g) 27^9

(h) $\binom{8}{2}$

(i) $\binom{35}{9}$

(j) $\frac{20!}{(10!*2^{10})}$

(k) $\binom{n+k}{k}$

(l) $n-1$

(m) $\binom{n-1}{k}$

6. Fermat's Wristband

(a) k^p
Each bead has k possibilities. There are p beads, meaning that the total number of sequences is k^p.

(b) $k^p - k$
If it uses at least two colors, all you have to do is take out the possibilities that it only uses one color, which is k possibilities.

(c) $\frac{k^p-k}{p}$
We know that p is prime, using Fermat's little theorem we know that rotating (1 to p-1) sequences will not be an equivalent sequence. We also know that if you have a sequence of length p, it will be equivalent to other length p sequences, therefore the total number of non-equivalent patterns is $\frac{k^p-k}{p}$.

(d) Fermat's little theorem states that :

$$a^p \equiv a \pmod{\text{p}} \tag{2}$$

Using the answer from part 6.c, we have $\frac{k^p-k}{p}$, where k is co-prime to p (since p is prime). In this case this would mean that $k^p - k$ has to be divisible by p since $\frac{k^p-k}{p}$ has to be an integer. Therefore, we have :

$$k^p \equiv k \pmod{\text{p}} \tag{3}$$

which is the same as Fermat's Little Theorem.