1 Counting Cartesian Products

For two sets A and B, define the cartesian product as $A \times B = \{(a,b) : a \in A, b \in B\}$.

(a) Given two countable sets A and B, prove that $A \times B$ is countable.

(b) Given a finite number of countable sets $A_1, A_2, \ldots, A_n$, prove that

$$A_1 \times A_2 \times \cdots \times A_n$$

is countable.

Solution:

(a) As shown in lecture, $\mathbb{N} \times \mathbb{N}$ is countable by creating a zigzag map that enumerates through the pairs: $(0,0), (1,0), (0,1), (2,0), (1,1), \ldots$. Since A and B are both countable, there exists a bijection between each set and a subset of $\mathbb{N}$. Thus we know that $A \times B$ is countable because there is a bijection between a subset of $\mathbb{N} \times \mathbb{N}$ and $A \times B : f(i,j) = (A_i, B_j)$. We can enumerate the pairs (a,b) similarly.

(b) Proceed by induction.
Base Case: $n = 2$. We showed in part (a) that $A_1 \times A_2$ is countable since both A_1 and A_2 are countable.
Induction Hypothesis: Assume that for some $n \in \mathbb{N}$, $A_1 \times A_2 \times \cdots \times A_n$ is countable.
Induction Step: Consider $A_1 \times \cdots \times A_n \times A_{n+1}$. We know from our hypothesis that $A_1 \times \cdots \times A_n$ is countable, call it $C = A_1 \times \cdots \times A_n$. We proved in part (a) that since C is countable and A_{n+1} are countable, $C \times A_{n+1}$ is countable, which proves our claim.

2 Counting Functions

Are the following sets countable or uncountable? Prove your claims.

(a) The set of all functions f from $\mathbb{N}$ to $\mathbb{N}$ such that f is non-decreasing. That is, $f(x) \leq f(y)$ whenever $x \leq y$.

(b) The set of all functions f from $\mathbb{N}$ to $\mathbb{N}$ such that f is non-increasing. That is, $f(x) \geq f(y)$ whenever $x \leq y$.

Solution:

(a) Uncountable: Let us assume the contrary and proceed with a diagonalization argument. If there are countably many such function we can enumerate them as

	0	1	2	3	$\dots$
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	$\dots$
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	$\dots$
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	$\dots$
f_3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Now go along the diagonal and define f such that $f(x) > f_x(x)$ and $f(y) > f(x)$ if $y > x$, which is possible because at step k we only need to find a number $\in \mathbb{N}$ greater than all the $f_j(j)$ for $j \in \{0,\dots,k\}$. This function differs from each f_i and therefore cannot be on the list, hence the list does not exhaust all non-decreasing functions. As a result, there must be uncountably many such functions.

Alternative Solution: Look at the subset $\mathscr{S}$ of strictly increasing functions. Any such f is uniquely identified by its image which is an infinite subset of $\mathbb{N}$. But the set of infinite subsets of $\mathbb{N}$ is uncountable. This is because the set of all subsets of $\mathbb{N}$ is uncountable, and the set of all finite subsets of $\mathbb{N}$ is countable. So $\mathscr{S}$ is uncountable and hence the set of all non-decreasing functions must be too.

Alternative Solution 2: We can inject the set of infinitely long binary strings into the set of non-decreasing functions as follows. For any infinitely long binary string b, let $f(n)$ be equal to the number of 1's appearing in the first n-digits of b. It is clear that the function f so defined is non-decreasing. Also, since the function f is uniquely defined by the infinitely long binary string, the mapping from binary strings to non-decreasing functions is injective. Since the set of infinite binary strings is uncountable, and we produced an injection from that set to the set of non-decreasing functions, that set must be uncountable as well.

(b) Countable: Let D_n be the subset of non-increasing functions for which $f(0) = n$. Any such function must stop decreasing at some point (because $\mathbb{N}$ has a smallest number), so there can only be finitely many (at most n) points $X_f = \{x_1,\dots,x_k\}$ at which f decreases. Let y_i be the amount by which f decreases at x_i, then f is fully described by $\{(x_1,y_1),\dots,(x_k,y_k),$ $(-1,0),\dots,(-1,0)\} \in \mathbb{N}^n = \mathbb{N}\times\mathbb{N}\times\cdots\times\mathbb{N}$ (n times), where we padded the k values associated with f with $n-k$ $(-1,0)$s. In Lecture note 10, we have seen that $\mathbb{N}\times\mathbb{N}$ is countable by the spiral method. Using it repeatedly, we get $\mathbb{N}^{(2^l)}$ is countable for all $l \in \mathbb{N}$. This gives us that $\mathbb{N}^n$ is countably for any finite n (because $\mathbb{N}^n \subset \mathbb{N}^{(2^l)}$ where l is such that $2^l \geq n$). Hence D_n is countable. Since each set D_n is countable we can enumerate it. Map an element of D_n to (n, j) where j is the label of that element produced by the enumeration of D_n. This produces an injective map from $\cup_{n\in\mathbb{N}} D_n$ to $\mathbb{N}\times\mathbb{N}$ and we know that $\mathbb{N}\times\mathbb{N}$ is countable from Lecture note 10 (via spiral method). Now the set of all non-increasing functions is $\cup_{i\in\mathbb{N}} D_n$, and thus countable.

3 Undecided?

Let us think of a computer as a machine which can be in any of n states $\{s_1,\ldots,s_n\}$. The state of a 10 bit computer might for instance be specified by a bit string of length 10, making for a total of 2^{10} states that this computer could be in at any given point in time. An algorithm $\mathscr{A}$ then is a list of k instructions $(i_0,i_2,\ldots,i_{k-1})$, where each i_l is a function of a state c that returns another state u and a number j. Executing $\mathscr{A}(x)$ means computing

$$(c_1,j_1)=i_0(x), \qquad (c_2,j_2)=i_{j_1}(c_1), \qquad (c_3,j_3)=i_{j_2}(c_2), \qquad \ldots$$

until $j_\ell \geq k$ for some ℓ, at which point the algorithm halts and returns $c_{\ell-1}$.

(a) How many iterations can an algorithm of k instructions perform on an n-state machine (at most) without repeating any computation?

(b) Show that if the algorithm is still running after $2n^2k^2$ iterations, it will loop forever.

(c) Give an algorithm that decides whether an algorithm $\mathscr{A}$ halts on input x or not. Does your contruction contradict the undecidability of the halting problem?

Solution:

(a) Each of the k instruction can be called on at most n different states, therefore there are at most $n\cdot k$ distinct computations that can be performed during any execution. After $n\cdot k+1$ iterations we must have repeated one of these computations.

(b) Since $2n^2k^2 > n\cdot k+1$, $\mathscr{A}$ must repeat a computation $i_s(c_t)$ for some $(s,t)\in\{1,\ldots,n\}\times\{0,\ldots,k-1\}$. But we know that when $i_s(c_t)$ is performed the second time, its consective computations will be precisely the same that followed the first evaluation of $i_s(c_t)$. In particular, we will see $i_s(c_t)$ a third time, and hence a fourth, fifth time etc.

(c) From our solution to part (b) it follows that we only need to check whether after $2n^2k^2$ iterations, $\mathscr{A}(x)$ is still running or not. If it is, $\mathscr{A}(x)$ does not halt, otherwise it does. This does not contradict the undecidability of the halting problem, since it only states the inability to decide whether an *arbitrary* algorithm halts. Here we only proved the decidability for algorithms that can be run on an n-state machine, of which there are only finitely many!

4 Code Reachability

Consider triplets (M,x,L) where

```
M is a Java program
x is some input
L is an integer
```

and the question of: if we execute $M(x)$, do we ever hit line L?

Prove this problem is undecidable.

Solution:

Suppose we had a procedure that could decide the above. Consider the following program for deciding whether $M(x)$ halts:

```
main():
run M(x);
print('hello')
```

Then we ask, is `print('hello')` ever executed?

If so, it means that $M(x)$ halts. Otherwise, $M(x)$ infinite looped.