CS 70 Discrete Mathematics and Probability Theory
Spring 2026 Sinclair, Song

HW 06

1 Unions and Intersections

Note 10

Given:

- X is a countable, non-empty set. For all $i \in X$, A_i is an uncountable set.
- Y is an uncountable set. For all $i \in Y$, B_i is a countable set.

For each of the following, decide if the expression is "Always countable", "Always uncountable", "Sometimes countable, Sometimes uncountable".

For the "Always" cases, prove your claim. For the "Sometimes" case, provide two examples – one where the expression is countable, and one where the expression is uncountable.

(a) $X \cap Y$

(b) $X \cup Y$

(c) $\bigcup_{i \in X} A_i$

(d) $\bigcap_{i \in X} A_i$

(e) $\bigcup_{i \in Y} B_i$

(f) $\bigcap_{i \in Y} B_i$

Solution:

(a) Always countable. $X \cap Y$ is a subset of X, which is countable.

(b) Always uncountable. $X \cup Y$ is a superset of Y, which is uncountable.

(c) Always uncountable. Let x be any element of X. A_x is uncountable. Thus, $\bigcup_{i \in X} A_i$, a superset of A_x, is uncountable.

(d) Sometimes countable, sometimes uncountable.

Countable: When the A_i are disjoint, the intersection is empty, and thus countable. For example, let $X = \mathbb{N}$, let $A_i = \{i\} \times \mathbb{R} = \{(i,x) \mid x \in \mathbb{R}\}$. Then, $\bigcap_{i \in X} A_i = \varnothing$.

Uncountable: When the A_i are identical, the intersection is uncountable. Let $X = \mathbb{N}$, let $A_i = \mathbb{R}$ for all i. $\bigcap_{i \in X} A_i = \mathbb{R}$ is uncountable.

(e) Sometimes countable, sometimes uncountable.

Countable: Make all the B_i identical. For example, let $Y = \mathbb{R}$, and $B_i = \mathbb{N}$. Then, $\bigcup_{i \in Y} B_i = \mathbb{N}$ is countable.

Uncountable: Let $Y = \mathbb{R}$. Let $B_i = \{i\}$. Then, $\bigcup_{i \in Y} B_i = \mathbb{R}$ is uncountable.

(f) Always countable. Let y be any element of Y. B_y is countable. Thus, $\bigcap_{i \in Y} B_i$, a subset of B_y, is also countable.

2 Count It!

Note 10

For each of the following sets, determine and briefly explain whether it is finite, countably infinite (like the natural numbers), or uncountably infinite (like the reals):

(a) The integers which divide 8.

(b) The integers which 8 divides.

(c) The functions from $\mathbb{N}$ to $\mathbb{N}$.

(d) The set of strings over the English alphabet. (Note that the strings may be arbitrarily long, but each string has finite length. Also the strings need not be real English words.)

(e) The set of finite-length strings drawn from a countably infinite alphabet, $\mathscr{A}$.

(f) The set of infinite-length strings over the English alphabet.

Solution:

(a) Finite. They are $\{-8,-4,-2,-1,1,2,4,8\}$.

(b) Countably infinite. We know that there exists a bijective function $f : \mathbb{N} \to \mathbb{Z}$. Then the function $g(n) = 8f(n)$ is a bijective mapping from $\mathbb{N}$ to integers which 8 divides.

(c) Uncountably infinite. We use Cantor's Diagonalization Proof:

Let $\mathscr{F}$ be the set of all functions from $\mathbb{N}$ to $\mathbb{N}$. We can represent a function $f \in \mathscr{F}$ as an infinite sequence $(f(0), f(1), \cdots)$, where the i-th element is $f(i)$. Suppose towards a contradiction that there is a bijection from $\mathbb{N}$ to $\mathscr{F}$:

$$\begin{aligned}
0 &\longleftrightarrow (f_0(0), f_0(1), f_0(2), f_0(3), \dots) \\
1 &\longleftrightarrow (f_1(0), f_1(1), f_1(2), f_1(3), \dots) \\
2 &\longleftrightarrow (f_2(0), f_2(1), f_2(2), f_2(3), \dots) \\
3 &\longleftrightarrow (f_3(0), f_3(1), f_3(2), f_3(3), \dots) \\
&\vdots
\end{aligned}$$

Consider the function $g : \mathbb{N} \to \mathbb{N}$ where $g(i) = f_i(i) + 1$ for all $i \in \mathbb{N}$. We claim that the function g is not in our finite list of functions. Suppose for contradiction that it were, and that it was the n-th function $f_n(\cdot)$ in the list, i.e., $g(\cdot) = f_n(\cdot)$. However, $f_n(\cdot)$ and $g(\cdot)$ differ in the n-th argument, i.e. $f_n(n) \neq g(n)$, because by our construction $g(n) = f_n(n) + 1$. Contradiction!

(d) Countably infinite. The English language has a finite alphabet (52 characters if you count only lower-case and upper-case letters, or more if you count special symbols – either way, the alphabet is finite).

We will now enumerate the strings in such a way that each string appears exactly once in the list. We will use the same trick as used in Lecture note 10 to enumerate the elements of $\{0,1\}^*$ We get our bijection by setting $f(n)$ to be the n-th string in the list. List all strings of length 1 in lexicographic order, and then all strings of length 2 in lexicographic order, and then strings of length 3 in lexicographic order, and so forth. Since at each step, there are only finitely many strings of a particular length ℓ, any string of finite length appears in the list. It is also clear that each string appears exactly once in this list.

(e) Countably infinite. Let $\mathscr{A} = \{a_1, a_2, \dots\}$ denote the alphabet. (We are making use of the fact that the alphabet is countably infinite when we assume there is such an enumeration.) We will provide two solutions:

Alternative 1: We will enumerate all the strings similar to that in part (b), although the enumeration requires a little more finesse. Notice that if we tried to list all strings of length 1, we would be stuck forever, since the alphabet is infinite! On the other hand, if we try to restrict our alphabet and only print out strings containing the first character $a \in \mathscr{A}$, we would also have a similar problem: the list

$$a, aa, aaa, \dots$$

also does not end.

The idea is to restrict *both* the length of the string and the characters we are allowed to use:

1. List all strings containing only a_1 which are of length at most 1.
2. List all strings containing only characters in $\{a_1, a_2\}$ which are of length at most 2 and have not been listed before.
3. List all strings containing only characters in $\{a_1, a_2, a_3\}$ which are of length at most 3 and have not been listed before.
4. Proceed onwards.

At each step, we have restricted ourselves to a finite alphabet with a finite length, so each step is guaranteed to terminate. To show that the enumeration is complete, consider any string s of length ℓ; since the length is finite, it can contain at most ℓ distinct a_i from the alphabet. Let k denote the largest index of any a_i which appears in s. Then, s will be listed in step $\max(k, \ell)$, so it appears in the enumeration. Further, since we are listing only those strings that have not appeared before, each string appears exactly once in the listing.

Alternative 2: We will encode the strings into ternary strings. Recall that we used a similar trick in Lecture note 10 to show that the set of all polynomials with natural coefficients is countable. Suppose, for example, we have a string: $S = a_5a_2a_7a_4a_6$. Corresponding to each of the characters in this string, we can write its index as a binary string:

$(101, 10, 111, 100, 110)$. Now, we can construct a ternary string where "2" is inserted as a separator between each binary string. Thus we map the string S to a ternary string: 101210211121002110. It is clear that this mapping is injective, since the original string S can be uniquely recovered from this ternary string. Thus we have an injective map to $\{0,1,2\}^*$. From Lecture note 10, we know that the set $\{0,1,2\}^*$ is countable, and hence the set of all strings with finite length over $\mathscr{A}$ is countable.

(f) Uncountably infinite. We can use a diagonalization argument. First, for a string s, define $s[i]$ as the i-th character in the string (where the first character is position 0), where $i \in \mathbb{N}$ because the strings are infinite. Now suppose for contradiction that we have an enumeration of strings s_i for all $i \in \mathbb{N}$: then define the string s' as $s'[i] =$ (the next character in the alphabet after $s_i[i]$), where the character after z loops around back to a. Then s' differs at position i from s_i for all $i \in \mathbb{N}$, so it is not accounted for in the enumeration, which is a contradiction. Thus, the set is uncountable.

Alternative 1: The set of all infinite strings containing only as and bs is a subset of the set we're counting. We can show a bijection from this subset to the real interval $\mathbb{R}[0,1]$, which proves the uncountability of the subset and therefore entire set as well: given a string in $\{a,b\}^*$, replace the as with 0s and bs with 1s and prepend $'0.'$ to the string, which produces a unique binary number in $\mathbb{R}[0,1]$ corresponding to the string.

3 Unprogrammable Programs

Note 11

Prove whether the programs described below can exist or not.

(a) A program $P(F,x,y)$ that returns true if the program F outputs y when given x as input (i.e. $F(x) = y$) and false otherwise.

(b) A program P that takes two programs F and G as arguments, and returns true if for all inputs x, F halts on x iff G halts on x (and returns false if this equivalence is not always true).

Hint: Use P to solve the halting problem, and consider defining two subroutines to pass in to P, where one of the subroutines always loops.

Solution:

(a) P cannot exist, for otherwise we could solve the halting problem:

```
def Halt(F, x):
    def Q(x):
        F(x)
        return 0
    return P(Q, x, 0)
```

`Halt` defines a subroutine Q that first simulates F and then returns 0, that is $Q(x)$ returns 0 if $F(x)$ halts, and nothing otherwise. Knowing the output of $P(Q,x,0)$ thus tells us whether $F(x)$ halts or not.

(b) We solve the halting problem once more:

```
def Halt(F, x):
    def Q(y):
        halt
    def R(y):
        if y == x:
            F(x)
        else:
            halt
    return P(Q, R)
```

Q is a subroutine that always halts. R is a subroutine that halts on every input except x, and runs $F(x)$ on input x when handed x as an argument.

Knowing if Q and R halt on the same inputs boils down to knowing whether F loops on x (since that is the only case in which they could possibly differ). Thus, if $P(Q,R)$ returns "True", then we know they behave the same on all inputs and F must halt on x, and if $P(Q,R)$ returns "False", then we know they differ on some input (which must be x) and F must loop on x. Thus, if P exists, this algorithm solves the halting problem, which is a contradiction. Therefore, P cannot exist.

4 Computations on Programs

Note 11

(a) Is it possible to write a program that takes a natural number n as input, and finds the shortest arithmetic formula which computes n? For the purpose of this question, a formula is a sequence consisting of some valid combination of (decimal) digits, standard binary operators ($+$, $\times$, the "^" operator that raises to a power), and parentheses. We define the length of a formula as the number of characters in the formula. Specifically, each operator, decimal digit, or parentheses counts as one character.

(*Hint:* Think about whether it's possible to enumerate the set of possible arithmetic formulas. How would you know when to stop?)

(b) Now say you wish to write a program that, given a natural number input n, finds another program (e.g. in Java or C) which prints out n. The discovered program should have the minimum execution-time-plus-length of all the programs that print n. Execution time is measured by the number of CPU instructions executed, while "length" is the number of characters in the source code. Can this be done?

(*Hint:* Is it possible to tell whether a program halts on a given input within t steps? What can you say about the execution-time-plus-length of the program if you know that it does not halt within t steps?)

Solution:

(a) Yes it is possible to write such a program.

We already know one way to write a formula for n, which is to just write the number n (with no operators). Let the length of this formula in characters be l. In order to find the *shortest* formula we simply need to search among formulae that have length at most l.

Since there are a finite number of formulas of length at most l, we can write a program that iterates over all of them. For example, if we treat each character as a byte or an 8-bit number, the whole formula becomes a binary integer of length at most $8l$, so we can simply iterate over all binary numbers up to 2^{8l} and for each one check if it is a valid formula.

For each formula that we encounter we can compute its value in finite time (since there are no loop/control structures in formula). Therefore we can check whether it computes n, and then among those that do compute n we find the smallest one.

(b) Yes. Again it is possible to write such a program.

As before, given a number n, there is one program that we know can definitely write n, which is the program that prints the digits of n one by one. Let the length plus running time of this program be l. We only need to check programs that have a length of at most l and a running time of at most l, since otherwise their running time plus length would be bigger than l.

Similar to the previous part, we can iterate over all programs of length at most l (by treating each one as a large binary integer and checking each one's validity by e.g. compiling it). For each such program, we then run it for at most l steps. If it takes more time, we stop executing it and go to the next program, otherwise in at most l steps we see its output and we can check whether it is equal to n or not.

Now among all programs that have length at most l and execute for at most l steps and print n we find the one that has the shortest length plus execution time.