

CS 132 Homework 4 Solutions

February 1, 2005

10.31 In each case, determine whether the given set is countable or uncountable. Prove your answer.

f. The set of all functions from \mathcal{N} to \mathcal{N} .

We showed in the last homework that the set of functions from \mathcal{N} to $\{0, 1\}$ is uncountable. That set is a subset of this one, thus this one is uncountable also. For completeness we give a proof of this:

lemma: If A is uncountable and $A \subseteq B$ then B is uncountable.

The proof is by contradiction. Assume there exists a countable B and uncountable A such that $A \subseteq B$. Since B is countable it is listable, and since $A \subseteq B$ we can list A in the same order, hence A is also countable. But we assumed A to be uncountable, which is a contradiction.

g. The set of all non-increasing functions from \mathcal{N} to \mathcal{N} .

This set is countable. To show this we show that there is a one-to-one function which maps from non-increasing functions to finite sequences of natural numbers; then we show that there are only countably many finite sequences of natural numbers.

Consider an arbitrary non-increasing function f . The range of f has a least element x . Let n be the smallest natural number such that $f(n) = x$. Now note that since f is a non-increasing total function and x is the least element, then for all $m > n$, $f(m) = f(n) = x$. Thus each function is uniquely defined by the sequence $f(0), f(1), \dots, f(n)$.

Thus there is a one-to-one mapping from non-increasing functions $f : \mathcal{N} \rightarrow \mathcal{N}$ to finite sequences of natural numbers. This implies there is a bijection to some subset of these sequences: specifically, the non-increasing ones.

Since there are only countably many sequences of natural numbers (shown below), and a bijection from a subset of these to the non-increasing functions $f : \mathcal{N} \rightarrow \mathcal{N}$, the set of such functions is thus also countable.

Lemma: The set of finite sequences of natural numbers is countable

We show this by induction on n , the length of the finite sequence.

Base Case: There is only one sequence of length $n = 0$, the empty sequence.

Inductive Case: Assume that the set of sequences of length n , is countable. Call this sequence F_n . We will show this implies that F_{n+1} , the set of sequences of length $n + 1$, is countable too.

$$F_{n+1} = \bigcup_{i=0}^{\infty} \{i\} \times F_n$$

By the inductive hypothesis F_n is countable, so this is the countable union of countable sets, which is itself countable by Theorem 10.13.

Since F_n is thus countable for all n ,

$$\bigcup_{i=0}^{\infty} F_i$$

is countable, again by Theorem 10.13.

Note that the set of all non-decreasing functions $f : \mathcal{N} \rightarrow \mathcal{N}$ is not countable.

We show this with a diagonalization proof. Assume that this set is countable, and hence can be listed as $F = f_1, f_2, \dots$. Now consider the function f such that

$$f(i) = f_i(i) + 1.$$

By assumption F is a complete listing of all non-decreasing functions, so $f \in F$. By construction, however, for all functions $f_i \in F$ $f_i(i) \neq f(i)$ and so $f \neq f_i$. Hence both $f \in F$ and $f \notin F$, which is a contradiction. We conclude that F is not countable.

- h. *The set of all regular languages.* We know from Example 10.8 that the set of r.e. languages is countable. All regular languages are r.e. Thus this is a subset of a set known to be countable, and so must be countable itself.

10.37 *Let $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a partial function. Let $g(f)$, the graph of f , be the language $\{x\#f(x) \mid x \in \{0, 1\}^*\}$. Show that f can be computed by a Turing machine if and only if the language $g(f)$ is recursively enumerable.*

First, suppose that f is computed by some TM T_f ; we will show that there exists a TM T_g which accepts $g(f)$. On input $x\#y$, T_g saves y and simulates T_f on x . If T_f accepts, the output is compared to y ; T_g accepts if they are the same and otherwise it rejects. Thus if and only if $y = f(x)$, T_g halts and accepts. So T_g accepts exactly $g(f)$.

Now we show the opposite direction: assume there is a TM T_g such that $L(T_g) = g(f)$, we will construct a machine T_f which computes f . Our strategy will be to consider all strings y_1, y_2, \dots and simulate T_g on $x\#y_i$ for each y_i . However, since T_g does not *decide* $g(f)$ but only accepts it, we cannot run these simulations one at a time in series. Instead we will run them in parallel as was done in Theorem 10.6 and other proofs. We will consider y_1, y_2, \dots in canonical order. On each round we will begin a new simulation on the next y_i and advance all existing simulations by one transition. This will be repeated until we find $x\#y_i$ on which T_g halts. If this happens we will halt in configuration $(h_a, \underline{\Delta}y_i)$; otherwise we will continue looping. Thus for all x for which $f(x)$ is defined, T_f computes $f(x)$ and for all other inputs it is undefined.

11.6 *Show that every recursively enumerable language can be reduced to the language $Acc = \{e(T)e(w) \mid T \text{ is a TM and } T \text{ accepts input } w\}$.*

1. Let L be an arbitrary r.e. language accepted by the TM T , we will show that $L \leq Acc$. f will be the function which carries out the reduction. If x is a string over the input alphabet of T , $f(x) = e(T)e(x)$.
2. Clearly f is Turing computable—it needs only encode T and x .

3. If $x \in L$ then T must accept x and so $e(T)e(x) \in Acc$. Conversely, if $e(T)e(x) \notin Acc$ then T does not accept x and hence $x \notin L$.

11.8 Show that for any $x \in \Sigma^*$, the problem **Accepts** can be reduced to the problem: Given a TM T does T accept x ?

We show **Accepts** \leq **Accepts- x** .

1. First we define a function F that carries out this reduction. The input to **Accepts** is a TM T and an input w . We define F as $F(T, w) = T'$, where T' is defined as follows. T' erases its input, writes w to the tape and runs T on w . T' accepts if and only if T accepts w .
2. T' is a simple modification of T , and F is clearly computable.
3. If T accepts w then $L(T') = \Sigma^*$ and so T' accepts x . If w is not accepted by T then $L(T') = \emptyset$ and so x is not accepted by T' .

11.3 (Extra Credit) Show that if L_1 and L_2 are languages over Σ and L_2 is recursively enumerable and $L_1 \leq L_2$, then L_1 is recursively enumerable.

We give a proof by contradiction. Assume that $L_2 = L(T_2)$ is r.e. and $L_1 \leq L_2$, but L_1 is not r.e. Since $L_1 \leq L_2$ there exists a computable function f such that $x \in L_1$ if and only if $f(x) \in L_2$. If this is so, then there must exist a TM T_1 defined as follows: given an input x it computes $f(x)$ and then runs T_2 on $f(x)$. T_1 accepts if and only if T_2 accepts. Thus, by the construction of f , T_1 accepts the language L_1 . By assumption, however, no TM accepts L_1 . This is a contradiction; we can conclude that L_1 must be r.e.