Introduction to Complexity Theory Problem Set 1

Bayard Walsh

January 2024

1

L contains all strings $\in \{0,1\}^*$ with least one 1 symbol.

Lemma 1: Every string in L must have at least one 1 symbol

Base case: Day 1, L_1. We start with an empty collection and add 1 to our collection. Therefore we have $L_1 = \{1\}$, and every string in L_1 has at least one 1 symbol, so we have our base case.

Induction Day K, where $K > 1$: Assume that on day K we have collection L_K where every string in L_K has at least one 1 symbol. Consider if we can have any strings without any 1 symbol in our collection on day L_{K+1}. As all strings in L_K have at least one 1 symbol by assumption, the string without any 1 symbol must be generated on day $K+1$. Note that we cannot apply steps $1, 2$ because we have $K > 1$. Consider step 4 and take an arbitrary u, v from L_K and add uv to L_{K+1}. However all $u, v \in L_K$ must have at least one 1 symbol by assumption, so uv must also have at least one 1 symbol. Next consider step 3 and take an arbitrary w from L_K and add $0w$ and $w0$ to L_{K+1}. However all $w \in L_K$ must have at least one 1 symbol by assumption, so $0w$ and $w0$ must also have at least one 1 symbol. As these are the only ways to add strings to our collection on day $K + 1$, all strings in collection L_{K+1} must have at least one 1 character.

Lemma 2: Any string S with at least one 1 character is in L

Let $|S| = N$. We will do induction on the length of the string.

Base case: $N = 1$, therefore $S = 1$. By step 2 we have $1 \in L_1$ (and as we never remove strings, it is also in the limit of our process), and we have our base case.

Induction: $N = K + 1$. Assume that we have any string of length K with at least one 1 symbol in L_K, and we want to show that any string of length $K+1$ with at least one 1 symbol is in L_{K+1}. Let S be some string with at least one 1 symbol where $|S| = K+1$. Let us have x be some string where $|x| = 1$ and $x = 1$ or $x = 0$. Now, let us have S' be some string with at least one 1 symbol where $|S'| = K$ and $S'x = S$. Note that as $|S'| = K$ and S' has at least one 1 symbol we have $S' \in L_K$ by assumption.

Case 1: $x = 1$. We have $1 \in L_K$ by step 2 and $S' \in L_K$ by assumption. Therefore apply step 4 for $u = S', v = 1$. Now we have $S'1 \in L_{K+1}$, and as we have $S'1 = S$ we have $S \in L_{K+1}$

Case 2: $x = 0$. We have $S' \in L_K$ by assumption. Therefore apply step 3 for $w = S'$. Now we have $S'0 \in L_{K+1}$, and as we have $S'0 = S$ we have $S \in L_{K+1}$

Case 3: This is an edge case in this induction where S' is a string of all 0 symbols and $x = 1$ so $S = 0\cdots01$. Then S has at least one 1 symbol and $S' \notin L_K$ as it doesn't have a 1 symbol. In this case, we will define S'' and x' such that $x'S'' = S$, and S'' has some 1 symbol. Therefore $|S''| = K$ and S'' has at least one 1 symbol, so $S'' \in L_K$. Here note that $x' = 0$, meaning that we can generate S by applying step 3 with $w = S''$. Therefore in this edge case $S \in L_{K+1}$

Therefore any string S with at least one 1 character must be in L.

By *lemma 1* and *lemma 2*, we have that L contains all strings $\in \{0,1\}^*$ with at least one 1 symbol.

2

2.1 A

We define our invective function $f(u,v)$: First we have a string of 1 characters equal to the length of u then we have a string of 0 characters equal to the length of v, then u then v. Equivalently: $f(u,v) = 1_{|u|}0_{|v|}uv$

Lemma 1: For any given u, v and u', v' if $|u| \neq |u'|$ or $|v| \neq |v'|$ then $f(u,v) \neq f(u',v')$

Let us have $f(u,v) = 1_{|u|}0_{|v|}uv$ and $f(u',v') = 1_{|u'|}0_{|v'|}u'v'$. Consider if $|u| \neq |u'|$. Then we have $|u|$ 1 symbols at the front of $f(u,v)$, and $|u'|$ 1 symbols at the the front of $f(u',v')$, however as we know that $|u| \neq |u'|$ then the strings do not have the same number of 1 symbols at the beginning; therefore $f(u,v) \neq f(u',v')$.

Next, assume $|u| = |u'|$ *(if $|u| \neq |u'|$ then $f(u,v) \neq f(u',v')$ as shown previously)* and $|v| \neq |v'|$. Similarly we have $|v|$ 0 symbols after the first $|u|$ 1 symbols for $f(u,v)$, and $|v'|$ 0 symbols after the first $|u'|$ 1 symbols for $f(u',v')$. However as we know that $|v| \neq |v'|$ then the strings cannot have the same continuous length of 0 symbols after the 1 symbols at the start of the string, therefore the strings are different and $f(u,v) \neq f(u',v')$.

Therefore if $|u| \neq |u'|$ or $|v| \neq |v'|$ then $f(u,v) \neq f(u',v')$

Lemma 2: For any given (u,v) and (u',v') such that $(u,v) \neq (u',v')$ if $|u| = |u'|$ and $|v| = |v'|$ then $f(u,v) \neq f(u',v')$

If we have $|u| = |u'|$ and $|v| = |v'|$, then we know the the start of the strings are equal in all cases ($1_{|u|}0_{|v|} = 1_{|u'|}0_{|v'|}$). For simplicity we will only consider the second section of the function, uv and $u'v'$ to look for differences and consider those substrings in our proof.

Consider $u \neq u'$. As we know $|u| = |u'|$, the sub-string of u and u' is the same length for uv and $u'v'$ and if $u \neq u'$ then there must be some character in the first $|u|$ characters of uv and $u'v'$ where $u \neq u'$ so the sub strings don't agree in every position, meaning that $uv \neq u'v'$, and subsequently $f(u,v) \neq f(u',v')$.

Consider $v \neq v'$. As we know $|u| = |u'|$ and $|v| = |v'|$, the ending string section of $f(u,v)$ and $f(u',v')$ is the same length for uv and $u'v'$ and if $v \neq v'$ then there must be some character after the first $|u|$ characters of uv and $u'v'$ where $v \neq v'$, meaning the strings don't agree in every position and $uv \neq u'v'$, and subsequently $f(u,v) \neq f(u',v')$.

Therefore if $|u| = |u'|$ and $|v| = |v'|$ and either $u \neq u'$ or $v \neq v'$ then $f(u,v) \neq f(u',v')$.

From *lemma 1* we have that for any given u,v and u',v' if $|u| \neq |u'|$ or $|v| \neq |v'|$ then $f(u,v) \neq f(u',v')$. From *lemma 2* we have that for any given (u,v) and (u',v') such that $(u,v) \neq (u',v')$ if $|u| = |u'|$ and $|v| = |v'|$ then $f(u,v) \neq f(u',v')$. Therefore we have that for any given $(u,v) \neq (u',v')$ then $f(u,v) \neq f(u',v')$. Therefore $f(u,v)$ is injective, so an injective function exists for $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$

2.2 B

Function: $f(u,v) = uBTAG(|u| + |v|)v$, where $BTAG(|u| + |v|)$ is a binary representation of $|u| + |v|$, however if $BTAG(|u| + |v|)$, and u,v have only 1 symbols, we will replace the first symbol with a 0. This is achieved through the following conversion: $1 = 1, 2 = 10, 3 = 11, 4 = 100, 5 = 101 \cdots$. An example of the edge case is $f(11,1) = 11011$ instead of $f(11,1) = 11111$. Note that binary strings are unique to a given decimal number, and that binary strings do not start with a leading 0. Therefore adding the leading 0 will never cause a collision between strings of different lengths for $BTAG$. Note that a binary tag has at most $\lceil \log_2 (|u| + |v| + 1) \rceil$ positions, so we have $|f(u, v)| \le |u| + |v| + \lceil \log_2 (|u| + |v| + 1) \rceil$, which matches our problem requirements.

Lemma 1: if $|u| + |v| \neq |u'| + |v'|$ *then* $f(u, v) \neq f(u', v')$

If $|u| + |v| \neq |u'| + |v'|$, then the length of $|u| + |v|$ must be at least one greater or one less than $|u'| + |v'|$. The $BTAG$ length difference between two strings with a length difference of at least one is either equal, one greater, or one less. However the length would only decrease if $|u| + |v| < |u'| + |v'|$ and only increase if $|u| + |v| > |u'| + |v'|$, as longer and shorter binary strings contain less data. Therefore, if the $BTAG$ between $f(u, v), f(u', v')$ are different then the strings must be different length and are then not equal. If the $BTAG$ is the same length then $f(u, v) \neq f(u', v')$ because $|u| + |v| \neq |u'| + |v'|$.

Lemma 2: if $|u| = |u'|$ *and* $|v| = |v'|$ *and* $(u, v) \neq (u', v')$ *then* $f(u, v) \neq f(u', v')$

In this case, if both of the strings are the same length but one or either of them don't equal each other there must be some place in $f(u, v)$ and $f(u', v')$ where the strings diverge, meaning $f(u, v) \neq f(u', v')$.

Lemma 3: if $|u| + |v| = |u'| + |v'|$ *and* $|u| \neq |u'|$ *and* $|v| \neq |v'|$ *and* $(u, v) \neq (u', v')$ *then* $f(u, v) \neq f(u', v')$

Consider if $f(u, v) = f(u', v')$. Say $|u'| < |u|$ (though it can be applied either way (ie $|u'| > |u|$)). Then there must be some section of string from the front of $BTAG(u', v')$ that equals the end of u, so the strings align. As we know $BTAG(u, v) = BTAG(u', v')$, there must be some section at the front of v' so the strings can be aligned to match up. With this moving, the strings must be symmetrical in terms of their contents and with the $BTAG$, which is only possible with only 1 strings and an overall $BTAG$ of only 1 strings (there is no $BTAG$ for all 0). However, our algorithm adds a leading 0 for this case. Therefore if $|u| + |v| = |u'| + |v'|$ and $|u| \neq |u'|$ and $|v| \neq |v'|$ and $(u, v) \neq (u', v')$ then $f(u, v) \neq f(u', v')$.

From these 3 lemmas the given function has the appropriate bound in length and is injective.

2.3 C

We cannot encode the length information more compactly then the approach in **B**, and without the length information of strings we won't be able to distinguish between symmetrical concatted strings. Therefore, we every injective function must follow this bound. Infinitely many pairs can be created through abitrary adding any amount of 0 or 1 to either string.

3

A key initial observation for this problem is to apply the property that for any rational real number x, there must be some integers p, q such that $x = \frac{p}{q}$. This is helpful in solving the problem of determining whether a given real number is rational, because the set of real numbers is uncountable (as per theorem 4.17). Therefore, if the friend is attempting to determine whether a given real number is rational through an approach that computes over all real numbers instead of computing over a pairs of integers, the input would be uncountable, as the set of all real numbers is uncountably infinite, and is not Turing-recognizable. This is because there cannot be a one to one correspondence between real numbers and natural numbers (as shown in theorem 4.17). Therefore, there is no algorithm that can generate all real numbers, meaning there is no Turing machine that can decide on real numbers and thus, the set of real numbers is not Turing-recognizable.

However, the set of natural numbers is countable, and, as p, q are both natural numbers, these inputs are countable and Turing recognizable.

From here we can apply the "diagonalization" approach from def 4.15, which is a strategy using the sets of p, q to obtain a list of all the elements of x. This approach involves diagonally computing pairs of p, q
(say $(0,0), (-1,0), (1,0), (0,1) \cdots$) instead of computing pairs of p, q linearly (say $(0,0), (0,1), (0,2), (0,3) \cdots$). The key difference between these strategies is that the linear approach can continue infinitely without finding the pair (say approaching $(0, \infty)$), whereas the diagonalization approach will cover all potential pairs eventually. Note that some aspect of the algorithm may consider a Turing machine that doesn't halt as evidence that a number is irrational, for example, if our algorithm cannot find a pair p, q such that $\frac{p}{q} = x$, our friend might assume x must be irrational. If the diagonalization approach is not taken, the algorithm may not halt because it doesn't consider the correct pair, and our friend might wrongly conclude that x is irrational because of the non-halting behavior.

Therefore, through using this process of considering potential inputs, our algorithm has an input that is Turing recognizable. Another approach for computing all rational numbers, say one that computes across rational numbers could be Turing unrecognizable, so I would recommend to my friend to adopt the previous method as an aspect of their input algorithm.

4

Function: $f(x_1 \cdots x_k)$:

```
    Apply magical device $p(x_1 \cdots x_k)$:

        if YES:
            return $(x_1 \cdots x_k)$ as a root

        if NO:
            given $(x_1 \cdots x_k)$ calculate the NEXT set of integers $(x'_1 \cdots x'_k)$ using
            the $k$ generalized diagonalization method as described below

            Recursively call $f(x'_1 \cdots x'_k)$
```

Explanation:
Correctness: We have $p(x_1 \cdots x_k)$ returns YES if the polynomial expression evaluates to zero, and we only return $(x_1 \cdots x_k)$ if the procedure returns YES, otherwise we loop to $(x'_1 \cdots x'_k)$. Therefore if our function terminates we have a correct solution.

Termination: As given, we have the assumption that there exists integers $x_1, \cdots, x_k$ such that $p(x_1, \cdots, x_k) = 0$. Therefore there must be some sequence of integers $x_1, \cdots, x_k$, which is correct and will therefore cause f to terminate. Therefore we must show that our function will consider all possible options for $x_1, \cdots, x_k$ in order to terminate. This can be achieved through showing that every option for $x_1, \cdots, x_k$ is countable, because if the options are countable we can create a list of the options, and then check every value in the list through the NEXT function (similar to the approach in 4.15).

Generalized diagonalization to prove every $(x_1 \cdots x_k)$ sequence is countable:

$k = 1$. Here we have a list of length 1 of natural numbers, which by definition 4.14 is countable (pair the single element inside the list with itself).

$k = 2$. Following the example 4.15 we create a similar diagonalization between (x_1, x_2), where we consider negative numbers. In this case we can start with a sequence like
$(0,0), (0,-1), (0,1), (-1,0), (1,0), (1,1), (0,-2), (-1,-2) \cdots$ and gradually expand out diagonally from the origin to consider all permutations of 2 integers. This fixes the problem of attempting to list all potential x_1 before considering x_2, and gives us a list of all elements of (x_1, x_2) which we can then pair the elements ordered in the list at positions $1 \cdots N$ with the natural numbers $1 \cdots N$, meaning these permutations are countable and Turing recognizable.

General $k > 2$: Assume that for some way of computing all permutations of integers in list form $(x_1 \cdots x_k)$ we have correspondence so that these numbers are countable and we want to show that when computing all permutations of integers in list form $(x_1 \cdots x_k, x_{k+1})$ we have a correspondence to count the list. Let us have our countable list of all possible permutations of $(x_1 \cdots x_k)$ be $l_1, l_2, \cdots l_i$. From here, we will diagonalize with integers to add x_{k+1} to the integer list, forming sequences like $(l_1, 0)(l_1, -1)(l_1, 1), (l_2, 0), (l_2, -1) \cdots$ in order to eventually consider all possible permutations. Note that the tuple of form $(l_1, 0)$ can be consider as adding the number 0 to the end of the list l_1. Therefore, through the diagonalization correspondence, we can compute all permutations of integers that form $(x_1 \cdots x_k, x_{k+1})$, and therefore this form is countable. As we can count the list, we next assign numbers $1 \cdots N$ to the list. Therefore the input $(x_1 \cdots x_{k+1})$ is countable, meaning this algorithm will consider all possible permutations of $(x_1 \cdots x_{k+1})$, and as a solution must exists (as given), our algorithm will eventually find the solution, proving the correctness of the function.