CS687 Assignment 1

Divyanshu Bhardwaj
180253

15 February 2020

1 Question

Let $\mathbb{R}^{\mathbb{N}}$ be the set of infinite length sequences $(r_0, r_1, ...)$ of reals. (The indices of each suchsequence are natural numbers.) Prove or disprove: $\mathbb{R}^{\mathbb{N}}$ has the same cardinality as $\mathbb{R}$.

1.1 Solution

Consider the following bijection, A, between $\mathbb{R}$ and $(0, 1)$

$$A(x) = \frac{1}{2} + \frac{1}{\pi}\tan^{-1}(x)$$

A number in $(0, 1)$ can be represented as a binary number, which can be treated as an infinite binary sequence. Define $B : (0, 1) \times (0, 1) \to (0, 1)$

$$B(0.a_1a_2a_3a_4...., 0.b_1b_2b_3b_4....) = 0.a_1b_1a_2b_2a_3b_3a_4b_4....$$

We have a problem in above definition, in case of repeating trailing 1's e.g.

$$0.5 = 0.1000000.....$$

$$0.5 = 0.0111111.....$$

We avoid this situation by fixing the representation by using the former representation for all numbers having two binary sequence.

If we have $x_1, x_2, x_3, x_4, y \in (0, 1)$ such that $B(x_1, x_2) = B(x_3, x_4) = y$ then since x_1 and x_3 have odd place bits of y in it's binary sequence and x_2 and x_4 have even place bits of y in it's binary sequence, we have $x_1 = x_3$ and $x_2 = x_4$. **So B is injective.**

Above idea can be generalized to yield a function $C : (0, 1)^{\mathbb{N}} \to (0, 1)$, by using the idea of dovetailing. On similar arguments, C **is an injection**.

Since C is an injection, $A^{-1} \circ C \circ A^{\mathbb{N}}$: $\mathbb{R}^{\mathbb{N}} \to \mathbb{R}$ is also an injection where for $x \in \mathbb{R}^{\mathbb{N}}$,

$$A^{\mathbb{N}}(x) = (A(x_1), A(x_2), ...)$$

Therefore, we have an injection from $\mathbb{R}^{\mathbb{N}}$ to $\mathbb{R}$.

We can construct $g : \mathbb{R} \to \mathbb{R}^{\mathbb{N}}$ in the following way:

$$g(x) = (x, 0, 0, 0, 0, 0....)$$

g is an injective function from $\mathbb{R}$ to $\mathbb{R}^{\mathbb{N}}$. By Cantor-Bernstein-Schroeder theorem, there is a bijection from $\mathbb{R}$ to $\mathbb{R}^{\mathbb{N}}$. Hence, both have the same cardinality.

2 Question

Show that the intersection of a computably enumerable collection of computably enumerable languages is computably enumerable.

2.1 Solution

We know that a language is computably enumerable if there is a Turing Machine M that accepts it.

Let $M_0, M_1, M_2, M_3...$ be a set of computably enumerable Turing Machines and let L(M_i) be computably enumerable language where $i \in A$(A is the index set enumerating the Turing Machines). We will construct a Turing Machine M that accepts:

$$L = \{x : \forall i \in A,\, x \in L(M_i)\}$$

$$L = \cap_{i \in A} L(M_i)$$

Turing Machine M:
1. Input x
2. for $i \in A$ do 4 and 5 steps :
4. Run M_i on x.
5. If M_i then reject and halt.
6. Accept x and halt.

Since for any $x \in L$, it is accepted by all Turing Machine M_i, M will accept x in some steps. If any TM M_i does not accept x then $x \notin L$ and M will run forever or will reject.
Hence we have a turing machine M that accepts L. Therefore L is computably enumerable.

3 Question

Show that the union of a computably enumerable collection of computably enumerable languages is computably enumerable.

3.1 Solution

We know that a language is computably enumerable if there is a Turing Machine M that accepts it.

Let $M_0, M_1, M_2, M_3...$ be a set of computably enumerable Turing Machines and let L(M_i) be computably enumerable language where $i \in A$(A is the index set enumerating Turing Machines). We will construct a Turing Machine M that accepts:

$$L = \{x : \exists i \in A \, s.t. \, x \in L(M_i)\}$$

$$L = \cup_{i \in A} L(M_i)$$

Turing Machine M:
1. Input x
2. for f = 1 to ∞ :
3. for i = 0 to f do 4 and 5 :
4. Run M_i on x for f steps.
5. If M_i accepts then accept.

If the collection of Turing Machines is finite then in 3rd step of M we will run i to cover all Turing Machines. Since for any $x \in L$, it is accepted by some Turing Machine M_i, M will accept x in some steps. If no TM M_i accepts x then $x \notin L$ so M will run forever.
Hence we have a turing machine M that accepts L. Therefore L is computably enumerable or **union of a computably enumerable collection of computably enumerable languages is computably enumerable.**

4 Question

Show that every infinite computably enumerable language has an infinite decidable subset.

4.1 Solution

Let L be an infinite acceptable language, which is accepted by a turing machine T. We know that if a language is acceptable then the it computably enumerable i.e there is a computable bijection $f : \mathbb{N} \to L$. Consider the language accepted by the following code:

1. Input x
2. Initialize i to 0
3. While(True)
4. If $f(i) == x$, accept x and halt
5. If $f(i) > x$, reject x and halt
6. $i = i + 1$

Since f is a computable bijection, it will keep on enumerating **distinct** strings. Hence, the code will always terminate, either by accepting or rejecting the string. Therefore, the language accepted by the above code is decidable. Since f enumerated L, the language accepted is a subset of L. Hence proved.

5 Question

Lossless data compression has to be invertible - for every compressed word, there should be a unique decompressed word. Consider the binary alphabet $\sum = \{0, 1\}$. Let $C : \Sigma^* \to \Sigma^*$ be a compressor. Then the above constraint forces it to be one-to-one (an injection).

Prove that lossless data compression is ineffective: for all long enough lengths n and constant $c \in N$, the number of strings of length n which have compressed words less than length $n - c$, is at most 2^{n-c} . Discuss why it still makes sense to gzip a file.

5.1 Solution

The number of words of length less than $n - c$ is

$$\sum_{i=0}^{n-c-1} 2^i = 2^{n-c} - 1 < 2^{n-c}$$

Therefore, the number of strings of length n which can have compressed words less than length $n-c$, is at most 2^{n-c} .

No lossless compression algorithm can efficiently compress all possible data, because of the injection constraint but when it comes to data for human consumption, it tends to be very redundant. Computers are no better when it comes to demanding redundancy. Computer programs repeat the same instructions over and over again, and program source code is verbose. That is why even though lossless data compression techniques like gzip are ineffective for random strings, it performs good enough on data we use everyday.

6 Question

If $b_0b_1...b_n$ is a finite string, then is it true that all its "circular left shifts "$b_i...b_{n-1}b_0...b_{i-1}$, where $0 \leq i \leq n$ have the same complexity, up to an additive constant? Prove your claim.

6.1 Solution

Let $x = b_0b_1...b_n$ and $x_m = b_m...b_nb_0...b_{m-1}$ for $0 \leq m \leq n$. Suppose p is the shortest program to print x and q is the shortest program to print x_m. Then we know that $K(x) = |p|$ and $K(x_m) = |q|$. Now we write another program a for printing x_m:
Program a:
1. Run p to obtain x.
2. Left rotate x by m to get x_m.
3. Print x_m.

Similarly we will write a program b to print x:
Program b:
1. Run q to obtain x.
2. Right rotate x_m by m to get x.
3. Print x.

Now since program a prints x we have:

$$K(x_m) \leq |a| \tag{1}$$
$$K(x_m) \leq |p| + 2|m| + c_1 \tag{2}$$
$$k(x_m) \leq K(x) + 2|m| + c_1 \tag{3}$$

Similary for program b we have:

$$K(x) \leq |b| \tag{4}$$
$$K(x) \leq |q| + 2|m| + c_2 \tag{5}$$
$$K(x) \leq K(x_m) + 2|m| + c_2 \tag{6}$$

From both the equations we can conclude that $K(x)$ and $K(x_m)$ differ only by additive constant since both the strings can be derived from one another. Hence $K(x)$ and $K(x_m)$ have the same complexity up to additive constant.

7 Question

(Data Processing Inequality) Suppose ϕ: $\sum^* \to \sum^*$ be a total computable function. Then show that there is a constant, c_ϕ depending on ϕ, such that for all $x \in \sum^*$,

$$K(\phi(x)) \leq K(x) + c_\phi \qquad C(\phi(x)) \leq C(x) + c_\phi.$$

This inequality shows that no computable process can increase the complexity of a string,even though it can decrease the complexity.

7.1 Solution

Suppose there is a program p that produces string x and $|p| = K(x) = C(x|\lambda) = C(x)$. Let the longest primitive statement in our programming language be L. The program p will have a last line "print x". We will construct a new program p' in the following way:
1. Run p to obtain x.
2. Compute $\phi(x)$.
3. Print $\phi(x)$

We will remove the last statement from p i.e. "print x" and instead compute $\phi(x)$ and then print it. The program prints $\phi(x)$ and terminate. Therefore we will have:

$$K(\phi(x)) \leq (|p| - L) + c_\phi + L \tag{7}$$
$$K(\phi(x) \leq |p| + c_\phi \tag{8}$$
$$K(\phi(x)) \leq K(x) + c_\phi \tag{9}$$

where c_ϕ is the space required for storing instructions for computing ϕ.

8 Question

Consider finite prefixes of the characteristic sequence of the Halting problem, i.e. $\forall n \in \mathbf{N}$, $h_n = b_0 b_1 ... b_{n-1}$ where the $b_i = 1$ if $s_i \in H$, and $b_i = 0$ if $s_i \notin H$. Show that for all large n,

$$C(h_n) \leq log_2 n + O(1)$$

. This shows that there are uncomputable languages which have very low complexity.

8.1 Solution

We know that $H = \{x\text{: } T_x \text{ halts on } x\}$. We define a function ϕ in the following way: Given x, simulate T_x on x. $\phi(x) = 1$ if it halts else $\phi(x) = 0$. ϕ is partial computable by the definition and enumerates $i \in H$. Now we are given $n \in \mathbf{N}$ and we have to generate h_n.
Let us suppose k are the number of bits out of n that are 1 in h_n. If we know k we can enumerate H by the computations of $\phi(0), \phi(1), \phi(2)...$ in the order $\phi(i)$ terminates and define $Y = \{i\text{: } \phi(i)$ terminates and $i \leq n\}$. We stop as soon as $|Y| = k$. Now from Y we can construct all bits in h_n which are 1 and the rest bits are 0. So we have the whole sequence.
Since we need description of ϕ and $m (m \leq n \Rightarrow logm \leq n)$ to produce h_n, we have:

$$K(h_n) \leq log_2 m + c_\phi \tag{10}$$
$$K(h_n) \leq log_2 n + c_\phi \tag{11}$$
$$K(h_n) \leq log_2 n + O(1) \tag{12}$$

9 Question

A class lasts N lecture days, with a 'suprise test' - test whose date is not preannounced - in one of the lectures. Can there be such a surprise test?
1. If the test is on the N^{th} lecture day, then it is not a surprise because the students know that there is a test, and it has not been conducted yet. There is only one lecture remaining, so it must have the test.
2. Similarly, the surprise test cannot be in the $(N-1)^{st}$ lecture - the surprise test cannot be in the N^{th} lecture, there were no tests till the $(N-2)^{st}$ lecture, hence the $(N-1)^{st}$ lecture must have a test. Therefore, that test will not be a surprise.
Arguing inductively, we claim that there is no surprise test.
Examine this argument from the light of Kolmogorov complexity - can there be a surprise test?

9.1 Solution

{Referred to The Surprise Examination Paradox and the Second Incompleteness Theorem by Shira Kritchman, Ran Raz}
This is the surprise test paradox which can be understood in terms of Chaitin's proof for uncomputability of $K(x) \geq L$ for some $x \in \sum^*$ and large L. As we have studied that the statement

$K(x) \geq L$ cannot be determined hence is unprovable in any consistent mathematics theory.

Let L be any large integer guaranteed by Chaitin's proof such that for any $x \in \sum^*$ the statement $K(x) > L$ cannot be proved. We know that we can always prove $K(x) \leq L$ by running all the programs of length atmost L(which are 2^{L+1}) and verifying if any of them produces x. We also know that there are atmost 2^{L+1} programs of length L so there has to be an integer $0 \leq x \leq 2^{L+1}$ for which $K(x) > L$. Let the number of such integers be m and $m \geq 1$.

Assume $m = 1$. So there is a single integer $x \in \{0, 1, 2, ...2^{L+1}\}$. This means for all other integers y we have $K(y) \leq L$. Since we can prove that $K(y) \leq L$, the only x for which we didn't proved this statement is x such that $K(x) > L$. Thus if m = 1 we can prove that $K(x) > L$ by proving $K(y) \leq L$ for other integers. But this is a contradiction hence $m \geq 2$. Continuing like this we can prove that $m \geq i + 1$ for all $i \leq 2^{L+1} + 1$. Thus $m > 2^{L+1} + 1$ which is contradiction.

From the uncomputability of $K(x)$ we can conclude that there cannot be a surprise test. The resolution of this paradox lies in the conversion of teacher's statement to mathematical language. In this conversion the notion of knowledge is replaced by the notion of provability.