Assignmnet 2

Rohan (241110057)

1. **Recall that in the notes, we defined a language A to be decidable if A and A^c are Turing acceptable. Show that this is equivalent to saying that language A is decidable if and only if there is a Turing machine that accepts every string in A and halts, and rejects every string not in A and halts.**

   Solution:

   A language A is defined to be decidable if and only if:

   (a) (Definition 1 - Given in Notes) A and its complement A^c are both Turing acceptable (i.e., recursively enumerable).

   (b) (Definition 2 - To Prove Equivalence) There exists a Turing machine M such that:
   - If $x \in A$, then $M(x)$ accepts and halts.
   - If $x \notin A$, then $M(x)$ rejects and halts.

   Definition 2 $\Rightarrow$ Definition 1

   Suppose A is decidable in the sense of Definition 2. That means there is a Turing machine M that correctly decides A, i.e., it always halts and gives the correct decision for every input.

   Constructing two separate Turing machines:

   - M_A: Accepts x if $x \in A$ and runs forever if $x \notin A$.
   - M_{A^c}: Accepts x if $x \notin A$ and runs forever if $x \in A$.

   Since M_A and M_{A^c} accept A and A^c respectively, both are recursively enumerable. Thus, A and A^c are Turing acceptable, proving Definition 1.

   Definition 1 $\Rightarrow$ Definition 2

   Let A and A^c are both recursively enumerable, meaning there exist Turing machines:

- M_A that enumerates and accepts strings in A.
- M_{A^c} that enumerates and accepts strings in A^c.

Using the technique of dovetailing to construct a Turing machine that decides A:

(a) Run both M_A and M_{A^c} in parallel on input x, alternating steps between them.

(b) Since x belongs to either A or A^c, one of these machines must eventually halt and accept.

(c) If M_A halts first, accept x.

(d) If M_{A^c} halts first, reject x.

Since this machine halts on all inputs and correctly classifies membership in A, A is decidable in the sense of Definition 2.

2. **Prove that every infinite computably enumerable language contains an infinite decidable sub-set.**

   Solution:

   Let A be an infinite computably enumerable (c.e.) language. This means there exists a Turing machine M_A that enumerates A, i.e., M_A outputs an infinite sequence $a_1, a_2, a_3, \ldots$ such that $A = \{a_1, a_2, a_3, \ldots\}$.

   We define S by selecting elements of A in strictly increasing lexicographical order.

   (a) Initialize S as empty.

   (b) Let s_0 be the first string output by M_A.

   (c) For $i \geq 1$: Let s_i be the next string output by M_A that is lexicographically larger than all previously selected strings in S.

   (d) Set $S = \{s_0, s_1, s_3, \ldots\}$.

   Since A is infinite and the set of all finite strings s well-ordered lexicographically, there are infinitely many distinct strings in A of increasing lengths. Thus, S is finite.

   For deciding membership in S, construct a Turing machine M_S that operates as follows:

(a) Input: A string x.

(b) Enumerate S by running M_A, selecting only strings in lexicographically increasing order.

(c) At each step i:

- If x matches the i-th string s_i in S, accept.
- If the enumerated string from M_A is lexicographically larger than x, reject (since x cannot appear later in S).

(d) Halt: Since S is strictly increasing and x has finite length, M_S will halt on all inputs.

By selecting elements of A in strictly increasing lexicographical order, we have constructed an infinite decidable subset $S \subseteq A$.

3. **(Data Processing Inequality) Show that if x is any finite string, and $f : \sum^* \to \sum^*$ is any total computable function, then $C(f(x)) \leq C(x) + O(1)$. This says that you can never *increase* information through computation, you can either preserve it or decrease it.**

Solution:

Let p be the shortest program that generates x:

$$C(x) = |p|$$

where $U(p) = x$ for a universal Turing machine U.

Since f is a total computable function, there exists a Turing machine M_f such that:

$$M_f(x) = f(x)$$

Now, constructing a new program p' as follows:

(a) Use p to compute x via U.

(b) Apply M_f to x to get $f(x)$.

The length of this new program is:

$$|p'| = |p| + |M_f| = C(x) + O(1)$$

Since $C(f(x))$ is the length of the shortest program generating $f(x)$:

$$C(f(x)) \leq |p'| = C(x) + O(1)$$

Conclusion

$$C(f(x)) \leq C(x) + O(1)$$

This proves the Data Processing Inequality, indicating that applying a total computable function cannot increase the information content of a string.

4. **(Upsetting the apple cart) Imagine that you have a hard-drive filled with useful data. You place it on a stick of dynamite and light the fuse. Making the stick of dynamite, lighting the fuse etc. are all computable processes. Yet, in a few seconds, your "disorder" in the data will increase. How does this relate to the data processing inequality?**

   Solution:

   The Data Processing Inequality states that applying a total computable function f to a string x cannot increase its Kolmogorov Complexity:

$$C(f(x)) \leq C(x) + O(1)$$

   This means that computation cannot increase the amount of information, it can only preserve or decrease it.

   Destruction as a Computable Process

   Let x be the data on the hard drive, and let f represent the process of blowing up the hard drive. The final state $f(x)$ (random debris) contains no structured information. Since the explosion is a computable transformation, it is clear that:

$$C(f(x)) \leq C(x) + O(1)$$

   Loss of Structure

   Since a well-organized database has a highly structured Kolmogorov Complexity, while the debris is essentially random noise, therefore:

$$C(f(x)) \ll C(x)$$

   This shows that information content is irreversibly lost, aligning with the Data Processing Inequality.

   Conclusion

   Blowing up a hard drive is an example of a computable process that destroys structured information, reducing its Kolmogorov Complexity. This follows directly from the Data Processing Inequality, confirming that computation cannot create new information, it can only preserve or destroy it.

5. **Let x be an arbitrary finite binary string. Do all permutations of x have the same plain Kolmogorov complexity of x? If so, prove your claim. Otherwise, construct a counterex-ample, and prove that your string has some permutation which has a significantly different Kolmogorov complexity.**

   Solution:

   Not all permutations of x have the same Kolmogorov complexity.

   Counterexample 1: All ones vs. Random Shuffle

   Consider the binary string:

$$x = 1111\ldots1111 \quad \text{(all ones, length } n\text{)}$$

   This string has a simple structure and can be described concisely by:

$$\text{Print } n \text{ ones}$$

   Thus, its Kolmogorov complexity is:

$$C(x) = O(\log n)$$

   since only $O(logn)$ bits are needed to store n and describe the pattern.

   Now, considering a random permutation of x, producing:

   How is x' a permutation of x ?

$$x' = 101100111010001011\ldots$$

   If x' is algorithmically random, then:

$$C(x') \approx n$$

   Thus:

$$C(x') \gg C(x)$$

   showing that permutations of x can have vastly different complexities.

   Counterexample 2: Alternating Sequence vs. Random Shuffle

   Consider another structured string:

$$x = 101010101010\ldots \quad \text{(length } n\text{)}$$

This string follows a clear pattern and can be generated by the simple program:

Print alternating 0s and 1s up to lenngth n.

Thus, its complexity is also small:

$$C(x) = O(\log n)$$

What is a "random" shuffle?

However, if we randomly shuffle the bits of x, we obtain a new string x' with complexity:

$$C(x') \approx n$$

How ?

C(x') <= n/2.

Thus, permutations exist that significantly increase complexity, proving that not all permutations have the same Kolmogorov complexity.

Conclusion

Since there exist structured strings x such that some permutations x' are incompressible ($C(x') \approx n$), we conclude:

Not all permutations of x have the same Kolmogorov complexity.

6. **(Characteristic sequence of the halting problem) Let χ_H be the infinite binary sequence defined as follows. For any number i, the i^{th} bit of χ_H is 1 if the i^{th} Turing machine halts, and 0 otherwise. Since the Halting problem is uncomputable, χ_H is uncomputable. However, show that prefixes of χ_H are compressible: for all sufficiently large n, show that $C(\chi_H[0 \ldots n-1]) \leq logn + O(1)$, where $\chi[0 \ldots n-1]$ denotes the n-length prefix of χ_H. (This shows that even when a string is uncomputable, it may be highly compressible.)**

Solution:

The sequence χ_H contains mostly 0s, with only a few 1s at positions corresponding to halting Turing machines. Instead of storing all n bits explicitly, we store only the indices where 1s appear, leading to compression.

Encoding the Sequence Efficiently

2

(a) Each bit in χ_H represents the halting status of a Turing machine.

(b) Since at most n machines exist in the first n positions, at most n indices could contain 1s.

(c) Instead of storing all bits, we store only the indices where the bit is 1.

(d) Each index is a number between 1 and n, requiring at most $\log n$ bits per index.

(e) The total storage required is therefore at most $O(n \log n)$ in the worst case.

(f) However, a more efficient encoding exists: If we only store the list of halting indices (rather than an explicit bit sequence), then we need at most $O(logn)$ bits to describe the set of halting indices

How ?

Thus, the Kolmogorov complexity satisfies:

$$C(\chi_H[0...n-1]) \leq \log n + O(1)$$

Conclusion

Although χ_H is uncomputable, its prefixes are highly compressible because they can be represented concisely using only the halting indices. This shows that uncomputability does not necessarily imply incompressibility.

7. **(Chaitin's Omega, modified version) Let P be a prefix-free set of strings, which form the domain of the universal prefix-free machine M. Let ω be an infinite binary sequence defined by**

$$\omega = \sum_{\substack{p \in P, \\ M(p)\downarrow}} \frac{1}{2^{|p|}}.$$

Show that prefixes of ω are incompressible: for all sufficiently large $n, C(\omega[0 \ldots n-1]) \geq n - O(1)$. (The purpose of this question is to show that there are uncomputable strings which are also incompressible, in contrast to the previous question.)

Solution:

Chaitin's Omega ω is an algorithmically random sequence representing the halting probability of a universal Turing machine. Unlike χ_H, which was compressible, ω is highly incompressible, meaning that the Kolmogorov complexity of its prefixes is nearly maximal.

(a) ω encodes halting probabilities, meaning it contains dense and unpredictable halting information.

(b) The values of ω are highly random, following from the properties of a universal prefix-free machine.

(c) Since the encoding of ω follows Kraft's inequality:

$$\sum_{p \in P} 2^{-|p|} \leq 1,$$

this ensures that the sequence behaves like true randomness.

(d) Suppose, for contradiction, that $C(\omega[0...n-1]) < n - O(1)$.

(e) This would imply that there is a short description of $\omega[0...n-1]$, meaning the sequence could be compressed significantly.

(f) However, by the properties of algorithmic randomness, no such compression is possible. This was the question.

(g) Specifically, if $C(\omega[0 \ldots n-1]) < n - O(1)$, this would contradict the definition of algorithmic randomness, which states that no program shorter than $n - O(1)$ can generate a truly random sequence.

(h) The Kolmogorov complexity of a truly random sequence satisfies:

$$C(\omega[0...n-1]) \geq n - O(1)$$

Why ω is Different from χ_H

Sequence	Uncomputability	Compressibility	Reason
χ_H	Yes	Yes ($O(\log n)$)	Just the halting indices can be stored, making it compressible.
ω	Yes	No ($\geq n - O(1)$)	The sequence is algorithmically random, so compression not possible.

Conclusion

Unlike χ_H, which was structured and compressible, ω represents true randomness and is therefore incompressible. Since $C(\omega[0...n-1]) \geq n - O(1)$, it is concluded that some uncomputable sequences are also incompressible, proving the required bound.

8. **Show, using an argument using plain Kolmogorov complexity, that for all sufficiently large n, most strings of length n will <u>not</u> have a contiguous stretch of more than $log^2 n$ zeroes anywhere in the string. (This shows how you can prove that some event happens with very high probability, using the theory of plain Kolmogorov complexity.)**

Solution:

Counting Argument

6

- The total number of binary strings of length n is 2^n.
- The number of strings that contain a long contiguous stretch (more than $log^2 n$ zeros is much smaller.
- If such a string exists, we can compress it by describing:
  (a) The position of the long zero block (at most $logn$ bits).
  (b) The length of the zero block (at most $log^2 n$ bits).
  (c) The remaining string (at most $n - log^2 n$ bits).

Compressing the String

- If a string contains a long stretch of zeros, we can describe it with:

$$C(x) \leq (n - log^2 n) + O(logn) + O(log^2 n)$$

- This results in a description that is significantly shorter than n for large n.

Incompressibility and Randomness

- For a randomly chosen string, Kolmogorov complexity is approximately n.
- The fraction of strings with low complexity is small compared to 2^n.
- If a string were highly compressible due to a large zero block, it would be part of a small subset of non-random strings.

Conclusion

- The number of strings without long zero blocks vastly outnumbers those with long zero blocks.
- Thus, with overwhelming probability, a randomly chosen string does not contain a zero block of length $log^2 n$.