Some Easy Theorems in Kolmogorov Theory
Exposition by William Gasarch (gasarch@cs.umd.edu)

1 Introduction

Intuitively the string

00000000000000000000000000000000000000000000000000000000000000000000

is not random. Note that you could write a program of length $O(\log n)$ that print out $0^n$.

Intuitively the string

0110100011000000111010101000110011010010010101001010110101010111110000

is random. The shortest program to print it out might just be

\[ \text{print}(0110100011000000111010101000110011010010010101001010110101010111110000) \]

which is of length roughly the length of the string.

With this in mind Kolmogorov defined the following notion of complexity.

Definition 1.1 The Kolmogorov complexity of a string $x$, denoted $C(x)$, is the length of the shortest program that prints out $x$. (To make this formal you would need to define (1) define a model of computation such as Turing Machines, and (2) prove that the complexity only differs from a constant depending on which model you are using. We will not bother with that.)

Note 1.2 We often call algorithms that print out a string $x$ a description of $x$.

Lemma 1.3 For almost all $n$ there is a string $x \in \{0, 1\}^n$ such that $C(x) \geq n$.

Proof: Assume, by way of contradiction, that for all $x \in \{0, 1\}^n$ $C(x) < n$. Map each $x \in \{0, 1\}^n$ to the program that prints it. Note that this map is 1-1. There are $2^n$ elements in the domain and $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ in the range. Hence the map cannot be 1-1. Contradiction.

2 Classic proof that $C$ is Not Computable

Theorem 2.1 $C$ is not computable.

Proof: Assume, by way of contradiction, that $C$ is computable. Assume also that the program for $C$ is of size $s$. Consider the following program (Where $a$ is a constant to be named later.)

For each $x \in \Sigma^a$s compute $C(x)$. When you find an $x$ such that $C(x) \geq as$ print out that $x$.

This program is of size $s + \lg(as) + O(1)$. Its output is a string of length $as$. Pick a large enough so that

$s + \lg(as) + O(1) < as.$

But now the output is a string whose shortest description is of length $as$. Contradiction.
3 Easy Known Proof that $C \leq_T K$

**Theorem 3.1** $C \leq_T K$.

**Proof:**

1. Input $x$. We want to know $C(x)$.

2. For all Turing machines $M$ of length $\leq |x|$ ask Does $M(0)$ halt and output $x$? using the oracle for $HALT$.

3. Output the length of the shortest $M$ such that $M(0) \downarrow = x$

4 Main Point

The proof that $C$ is undecidable is unusual in that we do not use $HALT$. That is, the proof is not a reduction. Note also that $C \leq_T HALT$.

My students sometimes ask me *Will there be a problem on the exam where we need to prove something is undeniable, but a reduction to $HALT$ won’t work?* which is a stupid way to ask the smart question: *Is there a set $A$ such that $\emptyset <_T A <_T HALT$.* The usual answer I give is that there are no natural such sets so they should not worry about it. However, the two results about $C$ above suggest a natural set. We have $C$ is undecidable but the proof did not show $HALT \leq_T C$ and we also have that $C \leq_T HALT$.

Hence this raises the question: Could $C$ be that elusive natural intermediary degree- not decidable but not equivalent to $HALT$. Alas, this is not the case. There are two proof that this is not the case.

1. If there was a natural intermediary Turing degree then I would know about it.

2. In the next section we prove that $HALT \leq_T C$. Hence $HALT \equiv_T C$.

5 $HALT \leq_T C$

**Definition 5.1** Let $C_s(x)$ be the shortest program that prints out $x$ within $s$ steps. Note that this is computable: write a simple PRINT($x$) program, and look at all programs that are shorter than it.

**Theorem 5.2** $HALT \leq_T C$.

**Proof:**

Here is the algorithm for $HALT$ that uses $C$ as an oracle. The constant $a$ will be determined later.

1. Input($x$) (we want to know if $M_x(x)$ halts). Let $|x| = n$. 

2. Find \( s_0 \) such that, for all \( y \in \{0, 1\}^\text{an} \) \( C_{x,s_0}(y) = C(y) \). (This step uses the oracle for \( C \).)

3. Run \( M_x(x) \) for \( s_0 \) steps. If it halts then output YES. If not then output NO. (We still need to prove that this is correct.)

We need to show that if \( M_x(x) \) does not halt within \( s_0 \) steps then it never halts. Assume, by way of contradiction, that \( M_x(x) \) halts in \( s \geq s_0 \) steps. Then the following algorithm will be a short description of a string that has no short description.

1. Run \( M_x(x) \). Let \( s \) be the number of steps it took to halt.
2. For all \( y \in \{0, 1\}^\text{an} \) computer \( C_s(y) \).
3. Let \( y \) be a string of length \( an \) such that \( C_s(y) \geq |y| \).
4. Output \( y \).

The above algorithm can be described with

\[
|x| + \lg(a) + O(1)
\]

bits. Hence \( C(y) \leq |x| + \lg(a) + O(1) \).

By the definition of \( s \) we have

\[
C(y) = C_s(y) \geq |y|.
\]

Pick \( a \) such that

\[
|x| + \lg(a) + O(1) < a|x|.
\]

This yields a contradiction.

}\]