COMPSCI 750: Computation Complexity
Second Assignment

Florian Suess

November 12, 2022

Computable Transformation

We dive into the inards of the *reverse string* function $\boldsymbol{R}$. Consider the very nifty recursive definition;

$$\text{for } l \in \{0,1\} \text{ then } \boldsymbol{R}(l) = l$$
$$\text{let } w \in \{0,1\}^* \text{ then } \boldsymbol{R}(lw) = \boldsymbol{R}(w)l$$

Preliminaries; for $w \in \{0,1\}^*$, $w_i \in w$ refers to the letter in the i^{th} position of w (necessarily $0 < i \leq |w|$).

Lemma 1: $a, b \in \{0,1\}^*$ **we have** $\boldsymbol{R}(ab) = \boldsymbol{R}(b)\boldsymbol{R}(a)$**.**

We prove this inductively by the length of a.

Trivially if $|a| = 1$, $a = l \in \{0,1\}$, then $\boldsymbol{R}(ab) = \boldsymbol{R}(lb) = \boldsymbol{R}(b)l = \boldsymbol{R}(b)\boldsymbol{R}(y)$.

Assume that the claim is true for all a of length n, and that $a = la'$ with $l \in \{0,1\}$ and $|a'| = n$ is of length $n+1$. Then we have

$$\boldsymbol{R}(ab) = \boldsymbol{R}\big(l(a'b)\big) = \boldsymbol{R}(a'b)l$$

Then by assumption...

$$= \boldsymbol{R}(b)\boldsymbol{R}(a')l = \boldsymbol{R}(b)\boldsymbol{R}(la') = \boldsymbol{R}(b)\boldsymbol{R}(a)$$

By induction, we're finished.

Lemma 2: $\boldsymbol{R}(\boldsymbol{R}(w)) = w$ **for all** $w \in \{0,1\}^*$**.**

We prove this inductively yet again on the length of $|w|$.

Suppose $|w| = 1$ then $w = l \in \{0,1\}$ by definition we have $\boldsymbol{R}(l) = l$.

Assume that for any $v \in \{0,1\}^*$ such that $|v| < |w|$, we have $\boldsymbol{R}(\boldsymbol{R}(v)) = v$. Now for this w, it must be the case that $w = lv$ for some other $l \in \{0,1\}$ and $v \in \{0,1\}^*$. Clearly $|v| < |w|$.

$$\boldsymbol{R}(\boldsymbol{R}(w)) = \boldsymbol{R}(\boldsymbol{R}(lv)) = \boldsymbol{R}(\boldsymbol{R}(v)l)$$

... then crucially by **lemma 1** we continue to see;

$$= \boldsymbol{R}(l)\boldsymbol{R}\big(\boldsymbol{R}(v)\big) = lv = w$$

$\boldsymbol{R}$ is a bijection

For **injectivity**, let's assume that for some $a, b \in 0, 1^*$ that $\boldsymbol{R}(a) = \boldsymbol{R}(b)$. By lemma 2, applying $\boldsymbol{R}$ both sides yields $a = b$. For **surjectivity**, given some $a \in \{0,1\}^*$, we can construct $\boldsymbol{R}(a) \in \{0,1\}^*$ such that $\boldsymbol{R}(\boldsymbol{R}(a)) = a$ by lemma 2.

Computability

Although obvious, we can write a brief algorithm that shows by force the computable nature of $\boldsymbol{R}$.

Require: $W = w_1 w_2 w_3 ... w_n$
1: **if** W is *Empty* **then**
2: **return** Empty
3: **end if**
4: $W' = Empty$
5: $i = |W| - 1$
6: **while** $i \geq 0$ **do**
7: $W' = W'W[i]$
8: $i = i - 1$
9: **end while**
10: **return** W'

Clearly this algorithm halts nicely. As there is only one potential loop, which is administered via the variable i which is monotonically decreasing to 0.

Lemma 3: $w_i \in w$, $w'_k \in \boldsymbol{R}(w)$, $k = |w| - i + 1 \Rightarrow w_i = w'_k$

We prove this inductively on the length of w.

If $|w| = 1$, then $w = l$ for $l \in \{0,1\}$ and by definition $\boldsymbol{R}(l) = l$. Given $w_i \in w$, set $k = |w| - i + 1 = 1$.

$$w_i = w_1 = l = R(w) = w_1 = w_k$$

Now when $|v| < |w|$ we assume for v_i we can set $k = |v| - i + 1$ such that $v_i = v'_k \in R(v)$.

We can say $w = lv$ for some $l \in \{0,1\}$ and $v \in \{0,1\}^*$. Note: $|w| = |v| + 1$. Consider $w_i \in w$.

- if $i = 1$,
  $\Rightarrow w_i = l$, and since $R(vl) = lR(v) = w'$, $k = |w| - i + 1 = 1$ and $w'_k = w_i$

- if $i \neq 1$,
  $\Rightarrow w_i \in v$, set $v' = R(v)$, by assumption, $w_i = v'_t$ if $t = |v| - i + 1$
  $\Rightarrow$ if $v'_t \in lR(v) = R(vl) = R(w)$, $w_i = v'_k$ if
  $k = t + 1 = |v| - i + 1 + 1 = |w| - i + 1$.

By induction, we're finished.

$\boldsymbol{R}(w)$ and w have equal number of Monochromatic Arithmetic Progressions

Suppose we find a monochromatic progression in w (where $|w| = n$) of length k. That is the substring;

$$w_i w_{i+t} w_{i+2t} ... w_{i+(k-1)t}$$
$$\text{where } 1 \leq i \text{ and } i + (k-1)t \leq n$$

Such that all letters here are equal. Using **lemma 3** in great strides, we can systematically assemble the monochromatic progression in $w' = \boldsymbol{R}(w)$ of length k.

$$w'_{n-(i+(k-1)t)+1} w'_{n-(i+(k-2)t)+1} ... w'_{n-(i+t)+1} w'_{n-i+1}$$

Where all letters are nessecarily equal by **lemma 3**. We rely on **lemma 2** to apply the same proof for the converse where for every monochromatic progression in $R(w)$ we can build one in w. And so this sufficiently shows that there is an equal number of monochromatic progressions in w and $R(w)$.

Lemma 4: $|\boldsymbol{R}(w)| = |w|$

We prove this inductively on the length of w.
If $|w| = 1$, then $w = l$ for $l \in \{0, 1\}$ and by definition $\boldsymbol{R}(l) = l, |w| = |\boldsymbol{R}(w)|$.
Assume for $|v| < |w|$ that $|v| = |\boldsymbol{R}(v)|$.
$w = lv$ for some $v \in \{0, 1\}^*$ and $l \in \{0, 1\}$. Then...

$$|\boldsymbol{R}(w)| = |\boldsymbol{R}(lv)| = |\boldsymbol{R}(v)l| = |\boldsymbol{R}(v)| + 1 = |v| + 1 = |w|$$

By induction, we're finished.

There exists a $\mathbf{c > 0}$, $\forall w \in \{0, 1\}^*, |K(w) - K(\boldsymbol{R}(w))| \leq c$

The choice of $\boldsymbol{R}$ as a self-inverse function makes this rather straight forward. Choose any $c > 0$. Note; by lemma 2, $K(w) \leq c_p + |\boldsymbol{R}(w)|$ where c_p is some program encoding constant (can use encoding of the algorithm I supplied if desired). By lemma 4 we see, $c_p + |\boldsymbol{R}(w)| = c_p + |w| \geq K(\boldsymbol{R}(w))$. Hence;

$$|K(w) - K(\boldsymbol{R}(w))| \leq |(c_p + |w|) - (c_p + |w|)| = 0 < c.$$

As required.

Research Question

We consider the question *can programs simulate the human mind?*. We begin with the formal arguments presented, for the negative, in Ahbinav Muraleedharan's provocative paper[1].

Negative

Ahbinav constructs a mathematical argument - based around a very simple decision problem; "is your memory finite" in the context of both a Computing machine (effectively a Turing machine with finite memory) and Turing machine. He proves that no program can compute a correct answer to this decision problem without examining the memory tape of a arbitrary computing machine.[**ahbinav**] Leaving us, in the case of arbitrarily big computing machines, undecided as to when to halt the program and declare the memory to be finite. This leads us to a key contrast between a human mathematician, the designer of the computing machine, of whom by definition can solve this problem. This is used to show the existence of a statement about a mathematical object that is incomputable for a Turing Machine but computable for a human mathematician.

Negative

Ahbinav constructs a separate mathematical thought experiment, building an argument over the question; *can a humanoid robot perform all humanly doable tasks within any humanely operable environment.* He shows the sets of all possible objects are countably infinite by assumption that each object can be described by finitely many parameters. Godel numbering technique is used to drive an isomorphism between the set of objects and the natural numbers forcing the set of objects to be infinite, but countable.

Ahbinav defines the sets of environments to be uncountably infinite by describing these to be collections of objects forcing an isomorphism between the set of environments to the power set of objects. Then we finally define a task as an action sequence inside an environment.

Ahbinav's primary argument here is a contrast between the necessary generation of "large" and unique action sequences of an uncountably infinite set of environments, representing all humanely doable tasks, and the "relatively small" set of action sequences that can be generated by a robot with a Turing Machine as a brain. This argument is used to highlight the limitations to the powers of discrete-state machines and demonstrates a Turing machine's incomprehensibility of the human brain.

[1] "Turing Machines Cannot Simulate The Human Mind"

Foreword to the affirmative

Ahbinav doesn't spare much thought to the construction of arguments for the affirmative, so we instead focus on the subject of disapproval, the Church-Turing thesis argument.

Affirmative

For all practical purposes, it remains that, the fundamental building block of our brain is the so called neuron. With the recent digitizing of the C. elegans worm, with a nervous powered a set of 302 neurons [**worm**], we demonstrated a simulation of a collection of neurons. Afterall, a neuron on it's own is a simple mechanism, describable as a function onto the natural numbers. Many neurons merely being many functions onto the natural numbers. The Church-Turing thesis asserts that any function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine[**church-turing**], and so we can reasonably build up to the quantity of neurons of the human brain, and thus in turn mimic the function of the human brain that itself is known to simulate the mind hence a Turing machine can simulate the human mind.

Affirmative

The concrete definition of the human mind is ambiguous and leaves to the rise of interesting side argument, narrative beautifully driven by the Chinese Room [**chinese˙room**] argument, that perhaps it is instead more sought after to find a criterion of "human mind" and objectively leave the concept of the "human mind" unobtainable. The Turing test, an empirical definition of intelligence; if a machine's behavior cannot be differentiated from that of an intelligent human, it must be conceded that the machine possesses the same faculty as the human [**turing˙test**]. One then can build a program, that effectively lists a rulebook for interaction, that is worked on iteratively towards the objective pass of the Turing test. If we find the program to be indistinguishable from a human, who are you to say it fails to posses an ability a human has?

Mathematical modelling

In order to formalise the statement; "There exists a form of quantum randomness that is provable better than pseudo-randomness". We must break it in parts.

Pseudo-Randomness

The widely accepted mechanism for a pseudo-randomn number generator today is a two stage function.

- Accepts a *seed*.
- Deterministically generates a sequence based on this seed.

Without loss of generality, we shall live in the space of bit strings $\{0,1\}$. We can now formalise this pseudorandom mechanism via the function that returns a function;

$$\boldsymbol{PR} : \{0,1\}^* \to (\boldsymbol{G} : \mathbb{N} \to \{0,1\})$$

When we speak of pseudo-randomness, we think of sequences of the form $(S_n)_{n\in\mathbb{N}}$ where $S_n = G(s)(n)$ for some fixed seed s.

Quantum-Randomness

Quantum-computing is fundamentally supported by the qubit; that is $\varphi = \alpha|0\rangle + \beta|1\rangle$. Which represents a superposition state, represented by the linear combination of the basis states $|0\rangle$ and $|1\rangle$. Where α and β are complex numbers, restricted to $\alpha^2 + \beta^2 = 1$.

When we speak of quantum-randomness, we think in particular a sequence of the form $(Q_n)_{n\in\mathbb{N}}$ where Q_n is the measurement of some qubit.

Provably Better

As per lecture; randomness is best known symptomatically through the following properties[2]

- Unpredictability, that is given a n expanded sequence (where n is finite), $S_n = s_1 s_2 s_3 ... s_n$, there is no $f : \{0,1\}^n \times N \to \{0,1\}$ such that this function can emulate forever the sequence hence fourth; $S_{n+1} = f(s_1, s_2, s_3 ... s_n, 1), S_{n+2} = f(s_1, s_2, s_3 ... s_n, 2), ...$
- Incompressibility, the impossibility of a n expanded sequence (where n can $\to \infty$), to be compressed.
- Typicality, random sequences passing every statistical test of randomness.

If we find a form of randomness, that is the sequence R, exerting more of a symptom of randomness than another form of randomness, say R', for the same symptom. We say R is a "provably better" form of randomness than R'.

Then the statement in can be formalised in the following way

$$\exists (Q_n)_{n\in\mathbb{N}} \text{ exhibits a symptom of randomness stronger than } (Sn)_{n\in\mathbb{N}}$$

Proof of this statement

By definition $(Sn)_{n\in\mathbb{N}}$ is defined by a seed $k \in \{0,1\}^*$, $\boldsymbol{PR} : \{0,1\}^* \to \big(\boldsymbol{G} : \mathbb{N} \to \{0,1\}\big)$ would yield the pseudo-random generating function $\boldsymbol{G}_k : \mathbb{N} \to \{0,1\}$. The existence of this function shows that this sequence is clearly predictable, as it is a computable function (clearly by the existence of pseudo-random number generators today).

We construct the qubit with α, β amplitudes that are equal. ie, suppose we have $\alpha = \beta = \frac{1}{\sqrt{2}}$ then we construct a quantum random generator sequence $(Q_n)_{n\in\mathbb{N}}$ that on each n collapses the qubit $\varphi = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. By definition the probability that this sequences returns a 1, or 0, are both 0.5. Due to the fundamental nature of a qubit, it is indeterministic and incomputable.

And so we show that quantum-randomness is less predictable than any form of pseudo-randomness, hence it's provably better.

[2] I spend effort formalising the first property, and relax on the others as we won't be dwelling on the other properties too much for the purposes of this assignment question