Jeffrey Shallit
School of Computer Science
University of Waterloo
shallit@uwaterloo.ca
https://cs.uwaterloo.ca/~shallit
Further takeaways

The same argument we used to prove the existence of at least one random string of each length can be used to prove results about “almost random” strings.

These are strings $x$ with $C(x) \geq |x| - a$. Call such a string $a$-random, so that a random string is 0-random.

**Theorem.** At least a fraction of $1 - 2^{-a}$ strings of length $n$ are $a$-random.

**Proof.** By the same argument, there are at most $1 + 2 + \cdots + 2^{n-a-1} = 2^{n-a} - 1$ compressions of length-$n$ strings into descriptions of size $0, 1, \ldots, n - a - 1$. So there are at least $2^n - 2^{n-a} + 1 \geq 2^n(1 - 2^{-a})$ leftover strings with $C(x) \geq n - a$.

Note: at least 50% of all strings are 1-random, 75% of all strings are 2-random, etc.
More generally we have

**Proposition.** Let $S$ be a set of binary strings of cardinality $n$. Then some string $x \in S$ has $C(x) \geq \log_2(n + 1) - 1$.

**Proof.** Suppose $t$ is the largest Kolmogorov complexity of a string in $S$.

Then all strings in $S$ have Kolmogorov complexity $\leq t$.

Then there can be at most $1 + 2 + \cdots + 2^t = 2^{t+1} - 1$ different descriptions of strings in $S$.

So $n \leq 2^{t+1} - 1$.

In other words, $2^{t+1} \geq n + 1$, or $t + 1 \geq \log_2(n + 1)$, or $t \geq \log_2(n + 1) - 1$. 
Unfortunately, Kolmogorov complexity is uncomputable, so perfect compression is unattainable.

**Theorem.** The quantity $C(x)$ is uncomputable.

**Proof.** Assume $C(x)$ is computable by a Turing machine $T$ that takes $x$ as input.

Create a new Turing machine $T'$ that (using $T$ as a subroutine) on input $\ell$, examines all strings of size $\ell$ in lexicographic order until it finds a string $y$ with $C(y) \geq |y| = \ell$.

Such a string exists by our theorem above. Then $T'$ outputs $y$. 

Now let’s compute the Kolmogorov complexity of $y$.

On the one hand, we have $C(y) \geq \ell$.

On the other hand, the string $y$ is completely determined by $T'$ and $\ell$, so $C(y) \leq |e(T')| + (\log_2 \ell) + 1$. Now

$$\ell \leq C(y) \leq |e(T)| + (\log_2 \ell) + c$$

(1)

for a constant $c$.

Choose $\ell$ sufficiently large so that $\ell > |e(T)| + (\log_2 \ell) + c$. This inequality contradicts Eq. (1).
The basic idea in this method is that “most” strings cannot be compressed very much. Generally speaking, a proof works by selecting a typical instance and arguing about its properties. In the incompressibility method, we pick a random “incompressible” string and argue about it.

Example.

Let $\pi(x)$ denote the number of primes $\leq x$.

A celebrated theorem known as the prime number theorem states that $\pi(x) \sim \frac{x}{\log x}$.

Using the incompressibility method, however, we can prove the weaker inequality $\pi(n) > cn/(\log n)^2$ for infinitely many $n$. 
The incompressibility method

Consider the ordinary binary representation of the non-negative integers, so that, for example, 43 is represented by 101011.

If \( n \geq 1 \) is represented by a string \( x \), then it is easy to see that \( |x| = \lfloor \log_2 n \rfloor + 1 \).

Unfortunately, there are also other possible representations for 43, such as 0101011.

To avoid the “leading zeroes” problem, we can define a 1–1 mapping between the natural numbers and elements of \( \{0,1\}^* \), as follows:

\( e(n) \) is defined to be the string obtained by taking the ordinary base-2 expansion of \( n + 1 \), and then dropping the leading bit 1.
For example, the representations of the first 8 natural numbers are
given in the following table.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$e(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>000</td>
</tr>
</tbody>
</table>
Encodings of integers

Note that $|e(n)| = \lfloor \log_2(n + 1) \rfloor + 1 - 1 = \lfloor \log_2(n + 1) \rfloor$.

Now $t + 1 \leq 2t$ for $t \geq 1$, so

$$\log_2(t + 1) \leq \log_2(2t) \leq (\log_2 t) + 1.$$  \hspace{1cm} (2)

It follows that

$$(\log_2 n) - 1 \leq |e(n)| \leq (\log_2 n) + 1.$$  \hspace{1cm} (3)
What’s wrong with the following argument?

Claim: $C(xy) \leq C(x) + C(y) + O(1)$.

*Flawed Proof.* Given program-input pair $(P, i)$ computing $x$ and $(Q, j)$ computing $y$, make a program $R$ that takes $e(P)ie(Q)j$ as input and runs $P$ on $i$ outputting $x$ and $Q$ on $j$ outputting $y$ and concatenates them to output $xy$. Hence the inequality follows.

???
**Flawed Proof.** Given program-input pair \((P, i)\) computing \(x\) and \((Q, j)\) computing \(y\), make a program \(R\) that takes \(e(P)ie(Q)j\) as input and runs \(P\) on \(i\) outputting \(x\) and \(Q\) on \(j\) outputting \(y\) and concatenates them to output \(xy\). Hence the inequality follows.

Problem: \(i\) is just a binary string, so how do we tell where \(i\) ends and \(e(Q)\) begins?

We can’t.

So we need to encode \(i\) in some way so it can be *uniquely decoded*.

Hence the need for a *prefix-free encoding* of strings and numbers.
We can encode the string $x$ as follows: $E(x) = 0^{|x|}1x$.

To decode $E(x)$ we first read the number of 0’s at the front until the first 1. This gives us $|x|$, and the next $|x|$ symbols gives $x$. We have $|E(x)| = 2|x| + 1$.

So this lets us prove $C(xy) \leq 2C(x) + C(y) + O(1)$.

But we can do better. We can encode $x$ with $E_2(x) = 0^{e(|x|)}1e(|x|)x$. To decode we count the number of 0’s at the front until the first 1, which gives us $f = e(|x|)$. Then the next $f$ symbols gives us $e(|x|)$, from which we get $|x|$. Then the next $|x|$ symbols are $x$.

Then $E_2(x) = 2\log_2 |x| + |x| + O(1)$, and the better bound $C(xy) \leq C(x) + C(y) + 2\log_2 C(x) + O(1)$. 

---

Prefix-free encoding
The incompressibility method

Previously we defined a binary string $x$ to be random if $C(x) \geq |x|$. Since we now have a bijection between binary strings and natural numbers, we can define a natural number $N$ to be random if $C(e(N)) \geq |e(N)|$.

By our theorem above, there exist infinitely many random integers.
We can use the incompressibility method to formal languages, specifically, to proving that certain languages are not regular.

Example. Let $L = \{0^k1^k : k \geq 1\}$. We prove that $L$ is not regular. Suppose it were.

Then it would be accepted by a DFA $M = (Q, \Sigma, \delta, q_0, F)$. We could then encode each integer $n$ by providing a description of $M$ (in $O(1)$ bits) and $q = \delta(q_0, 0^n)$ (in $O(1)$ bits), because then $n$ is uniquely specified as the least $i$ with $\delta(q, 1^i) \in F$.

Hence $C(e(n))) = O(1)$. But there exist infinitely many $n$ with $C(e(n)) \geq \log_2 n + O(1)$, a contradiction.
The incompressibility method

We can generalize the previous example, as follows:

**Lemma.** Let $L \subseteq \Sigma^*$ be regular, and define $L_x = \{y : xy \in L\}$. Then there exists a constant $c$ such that for each $x$, if $z$ is the $n$’th string in $L_x$ in lexicographic order, then $C(z) \leq C(e(n)) + c$.

**Proof.** The string $z$ can be encoded by the DFA for $L$ (in $O(1)$ bits), plus the state of the DFA after processing $x$ (in $O(1)$ bits), and the encoding $e(n)$. 
The incompressibility method

We now consider some applications of this lemma.

Example.

Let us prove that \( L = \{1^p : p \text{ prime}\} \) is not regular.

Let \( x = 1^{p_k} \), where \( p_k \) is the \( k \)'th prime.

Then the second element of \( L_x \) is \( y = 1^{p_{k+1}-p_k} \).

But as \( k \to \infty \), the difference \( p_{k+1} - p_k \) is unbounded (because, for example, the \( n - 1 \) consecutive numbers \( n! + 2, n! + 3, n! + 4, \ldots, n! + n \) are all composite for \( n \geq 2 \)). Hence \( C(1^{p_{k+1}-p_k}) \) is unbounded.

However, by our Lemma, we have \( C(y) \leq C(e(2)) + c = O(1) \), a contradiction.
Example.

Let us prove that \( L = \{xx^Rw : x, w \in \{0, 1\}^+\} \) is not regular.

Let \( x = (01)^m \) where \( m \) is random (i.e., 
\( C(e(m)) \geq |e(m)| \geq \log_2 m - 1 \)).

Then the lexicographically first element of \( L_x \) is \( y = (10)^m0 \).
Hence \( C(y) = O(1) \).

But \( C(y) \geq \log_2 m + O(1) \), a contradiction.
The incompressibility method

Example.

Let us prove that $L = \{0^i 1^j : \gcd(i, j) = 1\}$ is not regular.

Let $x = 0^{(p-1)!}1$ where $p$ is a prime, and $|e(p)| = n$.

Then the second word in $L_x$ is $y = 1^{p-1}$, which gives $C(y) = O(1)$.

But $C(e(p)) \leq C(y) + O(1)$, and there are infinitely many primes, so $C(e(p)) = O(1)$ for infinitely many primes $p$, a contradiction.
An open problem

Can you make the incompressibility method work for context-free languages?

This would be a big breakthrough!

The difficulty is handling the stack.

Oliver Glier (2003) has gotten it to work for DCFL’s.
Now we can solve the probability paradox we talked about on Wednesday.

Instead of the uniform distribution on strings, use the *universal probability distribution* instead: the probability of a string $x$ is $2^{-K(x)}$.

Here $K(x)$ is just like $C(x)$, but with a small twist: we demand that the set of all encodings is *prefix-free*: no encoding is the prefix of any other. This means that if we concatenate several encodings, we can uniquely disentangle them.

Kraft's theorem implies that $\sum_{x \in \{0,1\}^*} 2^{-K(x)} \leq 1$. So the probabilities don’t sum to 1 necessarily, but will never exceed 1.
Solution to the paradox

Here is the bet: if Alice wants to convince Bob that a binary sequence $x$ of length $n$ was formed by the flips of a fair coin, she offers to pay Bob $2^n - |y|$ dollars, where $y$ is any compression of $x$, against one dollar bet by Bob.

A truly random sequence will have $K(x) \geq |x|$, or at least very close to $|x|$. So then Alice pays $1$ or less.

But if Alice cheated somehow, and generated $x$ by some sort of short program, then Bob can discover a short encoding $y$ and make a good return on his investment.

Notice that Bob doesn’t have to find the optimal encoding; any one shorter than $|x|$ will do.

For more about this, see the article of Kirchherr et al. at https://link.springer.com/article/10.1007/BF03024407.