CSE3305 Formal Methods II

Lecture 12: Randomness and Complexity

Dr. Kevin Korb
School of Info Tech
and Software Eng.
Building 63, Room 205.
kevin.korb@infotech.monash.edu.au
Contents

• Randomness
• Codes
• Kolmogorov Complexity
• Algorithmic randomness
• Non-computability
• Diagonalization

Recommended Reading: Cover & Thomas, Chap 7
Description Length

What is:

The largest number describable in nine words?
Description Length

What is:

The largest number describable in nine words, plus one?
Randomness: What is it?

A common intuition: “random” means uniformly distributed (i.e., maximizing entropy). The common examples:

- Gambling devices (coins, dice, . . .)

Nevertheless, the intuition must literally be false: the outcomes of Poisson processes are random; stock markets are said to be random.

So, a possible definition: An event is random iff it is produced by an indeterministic process.

But then

- either everything is random
- or nothing is.

A concept that refuses to discriminate is useless!
Randomness

So far, we have the following possible accounts:

1. Uniformly distributed; maximizing entropy
2. Generated by an indeterministic process
3. Richard von Mises’ answer: unpredictability. (beyond the limit frequency in the collective)

As with probability, Kolmogorov has an alternative (which, again, won!): Kolmogorov complexity = incompressibility.

Informally, \([\text{Kolmogorov complexity}]\) measures the complexity of a string by the length of the shortest program that produces it.

- Intuition: The more compressible a string is, the less random.

⇒ This ties in nicely with the theory of computation.

Since the medium is (bit) strings, I will now introduce codes
Definition 1 Code
A code \( C \) for a source alphabet \( S \) into a target alphabet \( T \) is a function \( C : S \to T^* \)

- Each word \( w_i \in T^* \) is a finite sequence of symbols with length \( l_i = |w_i| \).
- So, \( T^* = \) the set of all strings definable by concatenating elements of \( T \) (including the empty string).

We can easily extend \( C : S \to T^* \) to \( C' : S^* \to T^* \) by concatenation.

Examples:
- Ascii
- Most data structures in computers
- Julius Caesar’s cypher (“decoder rings”)
- Genetic coding for proteins
We will only be interested in prefix codes.

**Definition 2 (Prefix codes)** A code has the prefix property (is a prefix code) if no codeword is a prefix of any other codeword.

I.e. if $t_1t_2\ldots t_m$ is a codeword, then

$$t_1t_2\ldots t_m * \ast \ast \ast \ast$$

is not a codeword – for any non-null replacement for the *'s.

To check if a code has the prefix property we need to compare each word of length $n$ with all words of length $\geq n$.

- Prefix codes: \{0, 10, 110\}; (almost) all computer programs (e.g., END-OF-PGM statements).
- Non-prefix code: \{0, 01, 11\}; Morse code.
# Morse Code

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.-</td>
</tr>
<tr>
<td>B</td>
<td>-...</td>
</tr>
<tr>
<td>C</td>
<td>-.-.</td>
</tr>
<tr>
<td>D</td>
<td>--.</td>
</tr>
<tr>
<td>E</td>
<td>.</td>
</tr>
<tr>
<td>F</td>
<td>..-.</td>
</tr>
<tr>
<td>G</td>
<td>--.</td>
</tr>
<tr>
<td>H</td>
<td>....</td>
</tr>
<tr>
<td>I</td>
<td>.</td>
</tr>
<tr>
<td>J</td>
<td>.----</td>
</tr>
<tr>
<td>K</td>
<td>-.</td>
</tr>
<tr>
<td>L</td>
<td>..-.</td>
</tr>
<tr>
<td>M</td>
<td>--</td>
</tr>
<tr>
<td>N</td>
<td>-.</td>
</tr>
<tr>
<td>O</td>
<td>---</td>
</tr>
<tr>
<td>P</td>
<td>.--.</td>
</tr>
<tr>
<td>Q</td>
<td>--.--</td>
</tr>
<tr>
<td>R</td>
<td>.-</td>
</tr>
<tr>
<td>S</td>
<td>...</td>
</tr>
<tr>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>U</td>
<td>..-</td>
</tr>
<tr>
<td>V</td>
<td>.----</td>
</tr>
<tr>
<td>W</td>
<td>.--</td>
</tr>
<tr>
<td>X</td>
<td>.-.</td>
</tr>
<tr>
<td>Y</td>
<td>-.--</td>
</tr>
<tr>
<td>Z</td>
<td>--..</td>
</tr>
</tbody>
</table>
Kolmogorov Complexity

\[ K_U(x) = n \]

where

- \( U \) is some \([Universal Turing Machine]\)\( W \)
  (Theorem 1 shows it doesn’t matter which one)
- \( x \) is an arbitrary bit string (we’ll use base 2)
  – or else an arbitrary number, program, etc.
- \( n \) is an integer; greater means more complex

Let \( p \) range over programs for \( U \), described as bit strings.

- E.g., \( p = 001101 \ldots 11000 \)
- We will assume all legitimate \( p \)'s have the prefix property (e.g., an END-OF-PGM statement).

Let \( U(p) = \) bit string output of \( U \) given \( p \)

**Definition 3** Kolmogorov Complexity (K-Complexity)

\[ K_U(x) = \min_{p:U(p)=x} |p| \]
Universality

**Theorem 1** Universality of K-Complexity

If $U$ and $U'$ are UTMs, then there is a constant $c \in \mathbb{N}$ s.t. for any bit string $x$

$$K_{U'}(x) \leq K_U(x) + c$$

**Proof**

Suppose $p$ is the minimal length program for $U$ printing $x$. Since $U'$ is universal, we can prepend to $p$ a program $p_U$ to emulate $U$ on $U'$. Hence, the inequality is satisfied with $c = |p_U|$. \qed

Formal Methods II
Universality

It is important to note \( U \) can be ignored

- theoretically
- but not necessarily \textit{practically}

The constant \( c \) may be \textbf{enormous}

- still, swamped in the limit; it is unaffected by the bit string \( x \)
Algorithmic Randomness

— Chaitin, Kolmogorov, Solomonoff

**Definition 4** Algorithmic randomness
A bit string $x$ is algorithmically random iff

$$K_U(x) \geq |x|$$

If $K_U(x) \geq |x|$ then it is uncompressible

There is no computer program shorter than $x$ itself which can output $x$. In essence, the best we can do is use the program

PRINT “X”;

**Definition 5** Algorithmically random number
A positive integer $n$ is algorithmically random iff

$$K_U(x) \geq \lceil \log_2 n \rceil$$

where $x$ is the binary representation of $n$. 
Algorithmic Randomness

Examples... which are compressible?

- 000...000
- 0
- 0101...0101
- 0101000001101100101110011... (\pi/10 in binary)
**Algorithmic Randomness**

Computable numbers (at large expansions) are by definition algorithmically non-random!

Non-computable numbers are by definition algorithmically random.

There are **far** more non-computable than computable numbers.
Non-computable Numbers

What is a non-computable number?

- Almost all numbers you know are computable, even though they are outnumbered!
  - $e$, $\pi$, $\sqrt{2}$, ...
- It’s impossible to enumerate a non-computable number
- It’s impossible to describe one algorithmically
- Nevertheless, we can describe one non-algorithmically!

First, let’s prove that there is indeed at least one non-computable number. In order to do so, I will introduce diagonal arguments.
We will use Cantor’s diagonal argument — something every computer scientist should know.

**Theorem 2** \( \aleph_0 = |\mathbb{N}| \) is not the only infinity. In particular, \( \aleph_1 = |[0, 1]| > \aleph_0 \).

**Proof**
Suppose, on the contrary, \( \aleph_1 = \aleph_0 \). Then we can put \([0, 1]\) into a one-to-one map with \( \mathbb{N} \); that is, we can enumerate them thus (omitting the decimal point on the left; where \( b_{ij} \) is the \( j \)-th bit in the binary enumeration of the \( i \)-th number):

\[
\begin{array}{cccc}
1 & b_{11} b_{12} & \ldots b_{1n} & \ldots \\
2 & b_{21} b_{22} & \ldots b_{2n} & \ldots \\
\vdots & \vdots & \ddots & \vdots \\
n & b_{n1} b_{n2} & \ldots b_{nn} & \ldots \\
\vdots & & & \\
\end{array}
\]

Construct \( \omega = \overline{b_{11}} \overline{b_{22}} \ldots \overline{b_{nn}} \ldots \), where each of these bits is flipped. Then for every \( n \in \mathbb{N}, \omega \) is not the \( n \)-th expansion. But clearly \( \omega \in [0, 1] \). We have a contradiction, so \( \aleph_1 \neq \aleph_0 \). \( \square \)
Turing On Computable Numbers

[Alan Turing] “On Computable Numbers” (1937):

**Theorem 3**
There are non-computable numbers.

Side-effect of Turing’s proof: invention of “Turing” machines and so computer science.

**Proof**
Suppose all $x \in [0, 1]$ are computable; then for any $x$ there is a $p$ s.t. $U(p) = x$.

**Lemma 1** The set of all computer programs, $\{p_i\}$, is enumerable.

**Proof.**
Simply enumerate all possible bit strings, eliminating those which are not programs.
Turing On Computable Numbers

Let’s enumerate all possible programs and their outputs:

\[
\begin{align*}
   p_1 & \quad b_{11}b_{12} \ldots b_{1n} \ldots \\
   p_2 & \quad b_{21}b_{22} \ldots b_{2n} \ldots \\
   \vdots & \quad \ldots \ldots \ldots \\
   p_n & \quad b_{n1}b_{n2} \ldots b_{nn} \ldots \\
   \vdots & 
\end{align*}
\]

By convention, if \( U(p_j) \) stops printing at \( b_{jm} \), then \( \forall k \geq 1 \) let \( b_{j(m+k)} = 0 \).

To be sure \( U(p_j) \) may not halt at all! So this “enumeration” isn’t a real decision procedure (algorithm)… unless the Halting Problem is solvable.

\[ \text{[Halting Problem]}_{\mathcal{W}}: \text{Find a decision procedure for whether } U(p) \text{ halts.} \]

Construct \( x_\omega = \overline{b_{11}} \overline{b_{22}} \ldots \overline{b_{nn}} \ldots \)

Clearly, \( \neg \exists p_j \text{ s.t. } U(p_j) = x_\omega \). I.e., \( x_\omega \) is not computable.

\[ \square \]
Turing On Computable Numbers

The “construction” of $x_\omega$ is purely fictional.

- May depend on values we could never observe
- So, this is not an algorithm, but still an informal proof.

Suppose this were an algorithm; then we could produce $p_{\text{super}}$ which could compute $x_\omega$; so $x_\omega$ would be in the list, which is a contradiction.

Corollary
The Halting Problem is unsolvable.

Proof
If the Halting Problem were solvable, the above would be an algorithm.
Consider $x_\omega$’s binary representation:

- infinite
- uncomputable
- uncompressible

Every finite subsequence $s$ must appear infinitely often

- Indeed with limit frequency of $2^{-|s|}$ (Borel)
- Including, e.g., Shakespeare’s plays, etc.

These may be true of some computables also

(irrationals), e.g., $\pi$, $e$, $\sqrt{2}$