COMPSCI750: Computational Complexity

Cristian S. Calude

Semester 2, 2022

The Halting Problem and descriptive complexity with applications

Textbook

- M. Sipser. *Introduction to the Theory of Computation*, PWS, 2013.

The Halting Problem

The Halting Problem is historically significant because it was one of the first problems to be proved undecidable, i.e., not solvable by ‘classical’ computers, the type of device you have on desks, smartphones, tablets, cars, etc.[1]

[1] A ‘classical’ computer is not an unconventional one, like quantum, biological and other types of computers. [4]. Classical computers are modelled by deterministic Turing machines [13].

History

The Halting Problem was named and first proved by Martin Davis [11, p. 70–71]. However, the formulation and undecidability proof of the Halting Problem are usually wrongly referred to Turing's seminal paper [32]. In contrast, more recent books and articles like [1, 14] cite correctly the origin of the problem. Lucas [21] presents a comprehensive analysis of the authorship of the definition and proof of unsolvability of the Halting Problem.

Pseudocode programs

Pseudocode programs use typical instructions in standard programming languages like assignments $(=)$, comparisons $(=, \neq, \leq, \geq, <, >)$, arithmetic operations $(+, -, \times, /)$, logical operations $(\vee, \wedge)$, programming keywords (**read**, **print**, **do**, **for**, **stop**), conditionals (**if-then-else**, **case**), iteration (**while**), wrapping (**begin**, **end**) and functions $P(N)$).

We are not going to define formally the set of pseudocode programs but we are requiring that *each program ends with the unique instruction* **stop**; the reason will appear clear later on.

An example

The (pseudocode) program

```
PROGRAM(1)
$i = 1$
$s = 1$
while $i \leq 100$ do $s = s + i$
$i = i + 1$
end while
print $s$
stop
```

calculates the sum $1 + 2 + \cdots + 100$: its output is the number $5,050$ and **halts**.

Non halting programs

An example of such a program can be obtained by smodifying PROGRAM(1):

```
PROGRAM(2)
1  $i = 1$
2  while $i \geq 1$ do $i = i + 1$
3  print $i$
4  end while
5  stop
```

Deciding whether PROGRAM(2) halts is simple.

Formulation of the Halting Problem

The Halting Problem (for programs) requires a given program, called **halting decider**, which can determine, given any pair (program, input), whether the program *will* eventually *halt* when run with that input.[2]

In this formulation, there are no resource limitations on the amount of memory or time required for the deciding program's execution. The **halting decider** should decide in *finite time* whether an arbitrary program will ever halt on arbitrary input, and to give the *correct answer for all* possible pairs (program, input).

[2]The future tense indicates that the decision has to be made 'in advance'.

The difficulty of the Halting Problem

Confirming that a stopping program eventually halts can be done just by running the program as there is no time limit for reaching the decision.

What about the case when the program does not stop?

How long should we run the program to decide that it will never stop?

Intuitively it seems that this time could be very (arbitrarily?) large. But, is it always *finite*?

Loops and non-halting programs

PROGRAM(2) never stops because the *loop* formed with instructions on lines 2, 3 and 4: the execution of the program can be described by the following *infinite* sequence of instructions:

$$1, 2, 3, 4, 2, 3, 4, 2, 3, 4, \ldots 2, 3, 4, \ldots$$

A *loop* is a set of instructions that is indefinitely repeated. For example a program that executes the instructions 1,2,3,4,5,6 in the following order

$$1, 2, 3, 4, 3, 4, 3, 4, 3, 4, \ldots$$

has a loop $(3, 4)$ and, consequently, it never stops.

Loops and non-halting programs

A program containing a loop never halts and deciding if a program contains a loop is easy.

Challenge 1: why?

Does this suggest that we are close to a solution to the Halting Problem?

To prove that testing for loops is enough for solving the Halting Problem we need to prove that *any non halting program contains a loop*. Is this true?

Loops and non-halting programs

The answer is *negative* and PROGRAM(3) is an example.

```
PROGRAM(3)
$i = 1$
while $i \geq 1$ do
    $j = 1$
    while $j \leq i$ do
        print 0
        $j = j + 1$
    end while
$i = i + 1$
end while
stop
```

Challenge 2: Proof?

Infinitely many decidable cases

The Halting Problem is decidable for various infinite classes of programs and inputs. For example, every program which contains no loop instruction halts and there are infinitely many such programs.

Infinitely many decidable cases

What about the following program?

```
PROGRAM(4)
1 read $N$
2 $i = 1$
3 $s = 1$
4 while $i \leq N$ do
5       $s = i + s$
6       $i = i + 1$
7 end while
8 print $s$
9 stop
```

Infinitely many decidable cases

What about the following program?

```
PROGRAM(5)
read $N$
$i = N$
while $i \geq N$ do
    $i = i + 1$
    print $i$
end while
stop
```

Infinitely many decidable cases

Challenge 3: What is the largest decidable set for the Halting Problem?

Length of programs

Programs are written using a finite alphabet. For example, the C programming language uses 256 characters. *The number of characters used in a program is its length.* For example, PROGRAM(1) uses 46 characters including a space after each instruction, but not the meta-characters for line numbers and the name of the program.

> **Decidability Criterion.** *Fix a program P and an $n > 0$. If we know the number of inputs v of length less than or equal to n for which $P(v)$ stops, then the Halting Problem for the set $\{(P, v) \mid \mathrm{length}(v) \leq n\}$ is decidable.*

Proof and comments

First, the number of programs of length at most a fixed n is finite because the number of strings on a finite alphabet of bounded length is finite.

Second, if we know the number N of programs $\mathrm{length}(v) \leq n$ for which $P(v)$ stops, then we can run in parallel $P(v)$ for all programs $\mathrm{length}(v) \leq n$ till the N halting inputs v show up and stop; all other programs of length less than or equal to n never stop on P.

Note that the Decidability Criterion requires that the bound n is fixed and known: it cannot be replaced by the weaker condition "there exists an $n > 0$".

Challenge 4: why?

As the argument above depends on a parameter n, we again have an infinite class of decidable instances of the Halting Problem provided we have the required extra information.

An infinite decider

The following program decides the halting of any pair (**P**, N):

```
PROGRAM(6)
1 read P, N
2 P(N)
3 print YES
4 stop
```

PROGRAM(6) reads an arbitrary pair (**P**, N) and executes **P**(N): if the computation halts, then PROGRAM(6) stops and returns YES and otherwise it continues forever.

Challenge 5: Does this program solves the Halting Problem?

The Halting Theorem

The Halting Theorem. *There is no program solving correctly and in finite time the Halting Problem for every input pair.*

We prove by absurdity and assume that there exists a **halting decider** solving the Halting Problem for all programs with variables for positive integers, a strict sub-class of all programs. If we show that this assumption leads to a contradiction, then we have a proof of the undecidability of the Halting Problem.

Restricting the class of programs does not diminish the generality of the proof.

Challenge 6: why?

The Halting Theorem

```
PROGRAM(7)
1 read $N$
2 generate all programs $P$ with length$(P) \leq N$
3 run halting decider on all programs above
4 delete all non halting programs
5 run the remaining halting programs till all stop
6 compute 1 plus the biggest value $v$ output by the halting programs
7 print $v$
8 stop
```

The Halting Theorem

An analysis of PROGRAM(7) shows that it *stops* on each input N and for large enough N the length of PROGRAM(7) – which depends only on N – is smaller than N. This means that PROGRAM(7) generates itself (line 2) and, on one hand it outputs a positive integer which is less than v, the maximum output produced by the halting programs of length less than N, on the other hand it outputs v, a contradiction. The existence of the **halting decider** leads to a contradiction, so the Halting Theorem was proved.

Challenge 7: Fill in the proof details.

Turing completeness

Pseudocode programs have a powerful property: *Turing completeness*.

A programing language is called *Turing complete* if it can simulate any Turing machine.

In detail, for every Turing machine M there exists a program P such that for every input x, $P(x) = M(x)$. This means that if $M(x)$ halts, then $P(x)$ halts and $P(x) = M(x)$; otherwise, $P(x)$ does not halt.

We always assume that the program M has enough time and memory to run on input x.

Universality

Despite their simplicity, pseudocode programs are capable of running very complicated computations.

A *universal pseudocode program* (U) – first defined and constructed by Turing [32] – is a pseudocode program that can simulate an arbitrary Turing machine on every arbitrary input.

This is achieved by reading both the code/description of the machine to be simulated as well as the input.

Challenge 8: Write a universal pseudo-code program U.

Challenge 9: Use U to give another proof of the Halting Theorem.

Programming languages with decidable Halting Problem

Virtually all programming languages today (e.g. Java, JavaScript, Perl, etc.) are Turing because they each implement all the features required to run pseudocode programs.

However, not all programming languages are Turing complete. In the late 19th century, Kronecker formulated the first notions of computability, including primitive recursion [19] and a class of functions based on it. Primitive recursive functions can be computed by programs with bounded recursion only, i.e. by programs using only loops for which an upper bound of the number of iterations can be determined before entering the loop.

Programming languages with decidable Halting Problem

Every programming language computing exactly the class of primitive recursive functions is not Turing complete because its instructions do not allow infinite computations, particularly, infinite loops.

This fact also shows that all primitive functions are total, they are defined on every input.

Early versions of the GLSL and HLSL programming languages (used in GPU's for rendering graphics) prohibit unbounded loops and enforced those rules in their respective compilers, those made impossible to write a program that didn't halt.

The Halting Problem is trivial for quantum programs as these programs always stop.

Approximate solutions to Halting Problem

Can the Halting Problem be approximately solved?

Answers have been proposed by various authors [22, 9, 23, 5, 2, 6, 7].

An anytime algorithm [17] trades execution time for quality of results; the algorithm returns not only a result but also a “quality measure” which evaluates how close the obtained result is to the result that would be returned if the algorithm ran until completion (which may be prohibitively long or even infinite as proposed by Manin in [23]).

To improve the quality of the solution, the anytime algorithm can be continued after it has halted, if the output is not considered acceptable.

Approximate solutions to Halting Problem

The idea of the anytime algorithm solving the Halting Problem is the following.

We fix a rational ε in $(0, 1)$ in terms of which we define the "quality measure" of the decision. Then we construct a program which stops on every program p (as input) and outputs one of the following decisions:

a) *p halts*, and in this case the result is provably correct, or
b) *p does not halt*, and in this case the probability (properly defined) that the result is wrong is less than ε.

Approximate solutions to Halting Problem

Running times play an important role in this problem because halting programs are not uniformly distributed, see [35, 34, 29] for experimental work and [18, 5] for theoretical results – in particular the result that every program either stops "quickly" or never stops [9].

The statistical anytime algorithm has two parts:

- the pre-processing which is done only once (when the parameters of the quality of solutions are fixed) and
- the main part which is run for any input program.

Approximate solutions to Halting Problem

Using an appropriate statistical framework we construct an anytime statistical algorithm which uses three parameters:

1. the probability of an erroneous decision,
2. the precision of and
3. the confidence level in the estimation.

The input program – tested for termination – is run for up to the largest number of steps made by any of the sampled programs.

If the computation does not terminate by this time threshold (cut-off), then the (possible wrong) decision is that the program will never stop.

With a confidence level as large as required, the anytime statistical algorithm produces correct decisions with a probability as large as required.

The algorithm operates with three parameters:

a) a *bound* $\varepsilon \in (0,1)$ for the decision error,
b) a *precision* parameter $1 < \lambda < \varepsilon$, and
c) a *confidence* parameter $1 - \delta \in (0,1)$ which is a probabilistic bound on the confidence in the precision parameter.

We sample N independent halting programs $x_1, \ldots, x_N \in \mathrm{dom}(\mathrm{U})$ and, by running them on U till they stop, calculate their respective running times $t_1(\mathrm{x}), \ldots, t_N(\mathrm{x})$. Let $\mathrm{x} = (x_1, \ldots, x_N)$ and $\mathrm{t}(\mathrm{x}) = (t_1(\mathrm{x}), \ldots, t_N(\mathrm{x}))$. Randomisation is done according to the probability distribution induced by an injective computable enumeration of the halting programs.

Challenge 10: How would you "randomise"?

Approximate solutions to Halting Problem

Pre-processing.
Fix three rational numbers $\varepsilon, \lambda, \delta \in (0,1)$ *with* $\lambda < \varepsilon$.
Compute $N = N(\lambda, \delta) = \lceil \frac{1}{2\lambda^2} \cdot \ln \frac{1}{\delta} \rceil$.
Sample N *independent programs* $\mathsf{x} = (x_1, \ldots, x_N) \in \mathrm{dom}(\mathsf{U})^N$ *and calculate their respective running times* $\mathsf{t}(\mathsf{x}) = (t_1(\mathsf{x}), \ldots, t_N(\mathsf{x}))$.
Increasingly order the observed running times t_i *and compute the threshold* $\mathsf{T} = t_{(\lceil N(1-\varepsilon+\lambda)\rceil)}(\mathsf{x})$.
Main part.
Let z *be an arbitrary program for* U.
If the computation $\mathsf{U}(z)$ *does not stop in time less than or equal to* T, *then declare that* $\mathsf{U}(z) = \infty$.

The axiomatic method

For over two thousand years, Euclid's *Elements* had provided the model of mathematical rigour and the link between the geometric constructions in mathematical spaces – modelled from the world of senses and actions – and the physical space, at any scale.

The theorems proved in the mathematical space model well phenomena in the physical space, like the relationships between stars and between Democritus's atoms. Euclid's model transgresses mathematics: for example, B. Spinoza presented moral and religious arguments in the Euclidean style in his book *Ethics, Demonstrated in Geometrical Order* published posthumously in 1677.

The axiomatic method

Euclid was probably the first mathematician to use the *axiomatic method* (in his geometry).

In modern era this method consists in

- a formal language constructed according to well defined rules, together with some basic notions, and statements – called *axioms or postulates* and
- *deduction rules*.

The axiomatic method

An axiom or a postulate is a statement that asserts a certain property of a basic notion, or that a certain relation holds between some basic notions. The axioms are accepted with no proof, only justification.

An example is the Euclid axiom of parallels:

> *If a line segment intersects two straight lines forming two interior angles on the same side that sum to less than two right angles, then the two lines, if extended indefinitely, meet on that side on which the angles sum to less than two right angles.*

A common deduction rule is *modus ponens*, Latin for "affirming affirms", first used by Theophrastus (c. 371–c. 287 BC): "P implies Q and P is asserted to be true, therefore Q must be true".

The axiomatic method

A *proof* is a finite sequence of statements each of which is either an axiom or can be deduced by some inference rule from previous statements in the sequence.

A statement for which there exists a proof is called a *theorem*.

An important requirement is that *the set of axioms is either finite or can be listed by a program*, that is, they form a *Turing recognisable or computably enumerable set.*

The axiomatic method

The definition of a proof and the above assumption allow that each step of the purported proof can be algorithmically checked for correctness by a *proof-checking program* (Hilbert's requirement), i.e. the set of proofs is decidable.

Furthermore, they also imply that *the set of theorems can also be listed by a program.* The propositional logic, the predicate logic and Peano (axiomatic) arithmetic are examples of axiomatic (or formal) systems.

The axiomatic method

The goal in choosing an axiomatic system – a problem in mathematical modelling – is to be able to (rigorously) *prove as many correct results as possible in a particular field of interest, without proving any incorrect results*.

The axiomatic method

For example, an axiomatic system for arithmetic should prove as many, ideally all, true statements about natural numbers, without proving any incorrect results.

But, how do we know that a statement about natural numbers is true?

Well, such a statement is true if it has been proved. . .

Is this not a circularity? This is a delicate issue. Most true statements in arithmetic are proved in an ad-hoc manner, not in an axiomatic way.

Euclid's fifth postulate

This postulate is equivalent to the more known statement that the sum of internal angles of any triangle is exactly two right angles, that is, 180°.

Late in the nineteenth-century doubts arose as to whether axioms are always "evident". A particularly interesting challenge affected right to Euclid's axiomatic treatment of geometry.
For Galileo, Kepler, Newton and Kant, the physical universe was faithfully described by the language of Euclidean circles, triangles and straight lines. But this "belief" was shaken by the German mathematician B. Riemann, who in his 1854 *Habilitation Thesis* proposed a new geometry – later called non-Euclidean – in which the sum of the internal angles of a triangle of stellar orders of magnitude can be more than 180°, in stark contradiction with the Euclid's fifth postulate.

Doubts

Locally, the Euclidean geometry works well for planes of null curvature; however, on the scale of the Universe, it is precisely the non-null curvature which enables the unification between gravitation and inertia, the crux of Einstein's relativity.

This “delirium” forced Riemann and his followers to abandon the Cartesian space, so intuitive in daily life, in favour of non-Euclidean geometries that can be obtained by the negation of the parallel postulate.

Doubts

This was a stupendous shock:

> *the axiomatic method could not be unconditionally trusted anymore because long term experience and intuition could be misleading.*

To be trusted the axiomatic method itself had to be studied and based on solid foundations [28, 20].

The axiomatic method under microscope

The *consistency* of the axiomatic system, that is, the impossibility of proving (in the system) a statement and its negation, is of outmost importance for the axiomatic treatment.[3] In Hilbert's words (from the lecture "Axiomatic Thought" presented to the Swiss Mathematical Society (1917)):

> *The chief requirement of the theory of axioms must go farther [than merely avoiding known paradoxes[4]], namely, to show that within every field of knowledge contradictions based on the underlying axiom-system are absolutely impossible.*

[3] In most, but not all, axiomatic systems, *contradictions entail everything.*

[4] Like those discovered in G. Cantor's naive set theory by B. Russell and many others which provoked a mathematical foundational crisis (in German, Grundlagenkrise der Mathematik). Russell's paradox showed the inconsistency of "the set of all sets that are not members of themselves".

The axiomatic method under microscope

To have a solid foundation for a more rigorous notion of proof an adequate framework was required: this was provided by the new *mathematical logic* developed since mid-nineteenth century by the efforts of English mathematician G. Boole, German philosopher, logician, and mathematician G. Frege, Italian mathematician G. Peano, German mathematicians R. Dedekind and D. Hilbert, English mathematician and logician B. Russell, Austrian logician, mathematician and analytic philosopher K. Gödel and English mathematician and computer scientist A. Turing, and a few others [24].

The axiomatic method under microscope

D. Hilbert, one of the greatest mathematicians of the nineteenth century, believed that an axiomatic theory could be developed independently of any need for intuition and will bring clarity and rigour to the extent that the verification of proof could be done by a mechanical device.

Hilbert's program proposed in 1920–1921 the following “solution”:

> *Ground all existing mathematical theories on a finite, complete and consistent axiomatic system.*

Hilbert's program

- a formalisation of mathematics, that is, a precise formal language, based on well-defined rules, in which all mathematical statements are written,
- a finite set of axioms and rules of inference,
- a proof that all *true* mathematical statements can be proved in the adopted axiomatic system *(semantic completeness)*,
- a proof – preferably using only finite mathematical objects – that no contradiction can be obtained in the axiomatic system *(consistency)*,
- an algorithm for deciding the truth or falsity of every mathematical statement (*decidability*).

Hilbert's program

The consistency of more complicated systems could be proven in terms of simpler systems and, ultimately, the consistency of all of mathematics will be reduced to that of arithmetic.

The new mathematical logic and the impressive three volumes of Russell and Whitehead *Principia Mathematica* [25] provided the logical basis for the development of Hilbert's program which by 1920 enlisted the works of important mathematicians and logicians such as P. Bernays, W. Ackermann, J. von Neumann and J. Herbrand.

Hilbert's convinction

In his address to the International Congress of Mathematics held in Paris in 1900, Hilbert expressed the deep conviction in the solvability of all mathematical problems, phrased as an *axiom*:

> *Is the axiom of solvability of every problem a peculiar characteristic of mathematical thought alone, or is it possibly a general law inherent in the nature of the mind, that all questions which it asks must be answerable? . . . This conviction of the solvability of every mathematical problem is a powerful incentive to the worker. We hear within us the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no* igorabimus*.*

From Hilbert to Gödel

Hilbert's program motivated the exceptional work of the young mathematician and logician K. Gödel. *His Incompleteness Theorem* says that

> *In every axiomatic system satisfying certain conditions, there exist statements which themselves as well as their negations are unprovable.*

The conditions require [27] that the axiomatic system is

(CEN) *computably enumerable*, i.e. the set of axioms (hence theorems) can be enumerated by a program,

(CON) *consistent*, i.e. the system cannot prove both a statement and its negation, and

(CAR) *rich enough*, i.e. the system contains a modicum of arithmetic of natural numbers.

Incompleteness

If (CEN), (CON) and (CAR) are satisfied, then there exists (and can be effectively constructed) a statement G such that both G and its negation non-G are unprovable in the axiomatic system, so the condition

(SCO) the axiomatic system is *syntactic complete*, that is, for any statement, either the statement or its negation is provable in the system,

is false:

the axiomatic system is syntactic incomplete.

Gödel's Incompleteness Theorem and its proof [16] are purely syntactic, in the sense that they do not use the semantic notion of "truth".

If the axiomatic system has a "sound semantics", then the Incompleteness Theorem can be re-phrased in a more intuitive form:

> *In every computably enumerable, rich enough and consistent axiomatic system, there exists a true, but unprovable sentence, that is, the axiomatic system is semantically incomplete.*

From the Halting Theorem to the Semantic Incompleteness Theorem

To illustrate this relation between the Halting Theorem and a form of Semantic Incompleteness Theorem we prove, following [12], a form of the Incompleteness Theorem using the Halting Theorem. By $N(P, v)$ we denote the statement

the program P will never halt on input v.

For any particular program P and input v, $N(P, v)$ is a perfectly definite statement which is either true (if P never halts) or false (if P eventually halts).

From the Halting Theorem to the Semantic Incompleteness Theorem

The falsity of $N(P, v)$ can always be demonstrated by running P on v. As we know, no amount of computation will suffice to demonstrate that $N(P, v)$ is true: we may still be able to prove by a logical analysis of some P's behaviour that $N(P, v)$ is true, but, because of the Halting Theorem, no such method can work correctly in *all* cases.

From the Halting Theorem to the Semantic Incompleteness Theorem

Suppose now that we choose an axiomatic system satisfying (CEN) and a variant of (CON) requiring that the statements of the form $N(P, v)$ can be formalised in the language of the system, (CAR). As expected, some statements $N(P, v)$ will be provable while others will not be provable.

Can we choose an *axiomatic system in which all true statements* $N(P, v)$ *and only them are provable?*

From the Halting Theorem to the Semantic Incompleteness Theorem

We will assume that the axiomatic system satisfies the following three conditions:

1. (CEN): The axiomatic system can formalise the statements $N(P, v)$, where P is a program and v is an input for P.
2. **Sound:** For all P, v, if there is a proof Π for $N(P, v)$, then P never halts on input v.
3. **Semantic complete:** For all P, v, if P never halts on input v, then there is a proof Π for $N(P, v)$.

From the Halting Theorem to the Semantic Incompleteness Theorem

Soundness means that the system can only prove “true” statements $N(P, v)$. Semantic completeness states the converse: every true statement $N(P, v)$ is provable. The question above can be re-phrased in a more precise: *Can we find a rich enough axiomatic system which is both sound and complete?*

From the Halting Theorem to the Semantic Incompleteness Theorem

Incompleteness Theorem. (Incomputability Semantic Variant) *No axiomatic system in which statements of the form $N(P, v)$ are formalised can be both sound and semantic complete.*

Here is a proof. Because of (CEN) and the definition of a proof in an axiomatic system, we can construct a *proof-checking program* that can determine whether an alleged proof Π for "$N(P, v)$" is a correct proof in the system.

From the Halting Theorem to the Semantic Incompleteness Theorem

Suppose, by absurdity, we had found a *sound and syntactically complete axiomatic system*. Suppose that the proofs in this system are particular strings of symbols on the specific finite alphabet of the system and let $\Pi_1, \Pi_2, \Pi_3, \ldots$ be a computable enumeration of all finite strings on this alphabet. This enumeration includes all possible proofs in the system, as well as a lot of other strings (including a high percentage of total non-sense). Most importantly, hidden between non-sense, *we have all possible proofs*.

From the Halting Theorem to the Semantic Incompleteness Theorem

Next we show how we can use our axiomatic system to solve the Halting Problem – an impossibility. Suppose we are given a program P and an input v and have to test whether or not P eventually halts on v. To answer this question we run in parallel the following two computations:

(A) run P on v and stop when the computation stops,

(B) generate systematically the sequence $\Pi_1, \Pi_2, \Pi_3, \ldots$ of all possible strings and use the proof-checking algorithm to determine whether each Π_i is a proof of $N(P, v)$; stop when the first proof Π_j for $N(P, v)$ was found.

From the Halting Theorem to the Semantic Incompleteness Theorem

We will prove that the above algorithm always stops and depending on which branch (A) or (B) it stops, the output is "stop" or "does not stop".

The algorithm stops when either (A) stops, i.e. when $P(v)$ stops, or in (B), i.e. when a proof Π_i for $N(P,v)$ is validated as correct. In the first case the answer is "P stops on v" and in the second case the answer is "P does not stop on v". To finish the proof we need to show the following three facts.

From the Halting Theorem to the Semantic Incompleteness Theorem

First we prove that the algorithm cannot stop simultaneously on both (A) and (B). Indeed, if the algorithm stops through (B), then a proof Π_i for $N(P, v)$ is found, so by soundness, the computation (A) cannot stop; if, on the contrary, the computation (A) stops, then no valid proof for $N(P, v)$ can exist because otherwise by soundness, hence P will never halt on input v.

From the Halting Theorem to the Semantic Incompleteness Theorem

Second we prove that the algorithm stops either on (A) or on (B). Indeed, if the algorithm does not stop on (A), then P will never halt on input v, so by completeness the algorithm will stop on (B). If the algorithm does not stop on (B), then there is no valid proof for $N(P, v)$, so again by completeness, P will stop on v, hence the algorithm will stop through (A).

From the Halting Theorem to the Semantic Incompleteness Theorem

Finally, we prove the correctness of the algorithm. If P will eventually halt on v, then the computation (A) will eventually stop and the algorithm will give the correct answer: "P stops on v". If P never halts on v, (A) will be of no use: however, because of completeness, there will be a valid proof Π_i of $N(P, v)$ which will be eventually *discovered* by the computation (B). Having obtained this Π_i we will be sure (because of soundness) that P will indeed never halt.
Thus, we have described an algorithm for solving the Halting Problem, a contradiction!

Questions

Challenge 11: Is the popular formulation of the Gödel Incompleteness Theorem – *in every axiomatic system satisfying certain conditions there exist true and unprovable statements* – correct?

Can a solution to the Halting Problem be used to construct a sound and complete axiomatic system for proving the statements $N(P, v)$?

From the Halting Theorem to the Semantic Incompleteness Theorem

We prove that a solution to the Halting Problem can be used to construct a sound and complete axiomatic system for the statements $N(P, v)$.

To this aim we consider the language of statements $N(P, v)$ and we assume that a **halting decider** solves *correctly and in finite time the Halting Problem for every pair* (P, v).

Then we construct an axiomatic system with no axioms and one inference rule:

> *Run the* **halting decider** *on* (P, v) *and return YES (if* $P(v)$ *never stops) or NO (otherwise).*

From the Halting Theorem to the Semantic Incompleteness Theorem

A proof in the axiomatic system is a finite sequence of instructions executed by the **halting decider** *on input pair* (P, v) *which ends in* YES.

Explicitly, proofs in this axiomatic system are *finite sequences of instructions* $i_1, i_2, \ldots, i_n$ executed by the **halting decider** on (P, v) which end with YES – recall that the **halting decider** stops always in finite time on any input.

From the Halting Theorem to the Semantic Incompleteness Theorem

If we have a proof for $N(P, v)$, then $P(v)$ does not stop by virtue of the correctness of **halting decider**, so the axiomatic system is sound.

If $P(v)$ does not stop , i.e. $N(P, v)$ is true, then there is a proof in the system for $N(P, v)$ because **halting decider** solves correctly the Halting Problem.

More about undecidable problems

Note that *undecidable problems*, like the Halting Problem, depend on the adopted mathematical model of computation and *unprovable statements*, like consistency, depend on a fixed axiomatic system: they are both *relative*.

For example, Gödel's Incompleteness Theorem *does not show the existence of absolutely unprovable statements (in arithmetic or anywhere else), but the existence of statements which cannot be proved* in a fixed axiomatic system with some properties.

Thinking about them as "absolutes" is incorrect and may lead to lunatic consequences [15].

Minimal programs

Consider the

> *the first natural number that cannot be named in less than fourteen words.*

Why is this number so special to deserve the honour to be discussed?

On one hand, as only finitely many natural numbers can be named with less than fourteen words, the number exists.

On the other hand, as the above definition has thirteen words, the number does not exist.

This paradox was discovered by B. Russell who called it *Berry's paradox*.[5]

[5] G. G. Berry, a librarian at the Bodleian library of the Oxford University, suggested it to him, see more in [10, p. 8].

Minimal programs

The expression “be named” in the above formulation needs to be made precise in order to be able to analyse it mathematically.

One possible way is to use programs as names for natural numbers. In this approach the *length* of a program, i.e. the number of characters of the program, will be used to define the shortest programs.

For example PROGRAM(1) has 64 characters (including blanks), so it length is 64. PROGRAM(4) needs a special attention as its size depends also on the value read into N, so the length of this program is bounded by the sum between $\log_2 N$ – the number of bits necessary to code N – and the lengths of its instructions.

Minimal programs

A *(length-) minimal* – sometimes called elegant – program is the shortest program that produces a given output, i.e. a natural number.

Explicitly: a program P is *minimal* if no program shorter than P produces the same output; if more than one program satisfy the above property, then the minimal program is the first lexicographical such program.

Minimal programs

Chaitin Minimality Theorem. [10, Chapter 5] *There is no program deciding whether an arbitrary program is minimal.*

Assume by the absurdity that there exists a **minimality-decider** which accepts as input a program P and returns in finite time YES or NOT according to whether P is minimal or not.

If **minimality-decider** returns on P YES, then P eventually stops, its output is a natural number N and no program shorter than P produces N.

We recall that $\mathrm{length}(P)$ is the length of the program P and $\mathrm{output}(P)$ is the output produced by P if it stops.

Minimal programs

For every natural N the program

```
$\textsc{Program}(P_N)$
1  read $N$
2  print $N$
3  stop
```

has $\mathrm{length}(P_N) \leq \log_2 N + 23$ and $\mathrm{output}(P_N) = N$.

Minimal programs

For every natural N there exists a minimal program P_N^{min} such that $\mathrm{output}(P_N^{min}) = N$ because the number of programs of length less than or equal to $\log_2 N + 23$ producing N is finite, so one of them is minimal.

With a similar argument we can show that for every naturals $M > N$ there exists a minimal program whose output is M.

Minimal programs

We next construct the program

```
PROGRAM($T_N$)
1 read $N$
2 generate all $P$ with length($P$) $> N$ till
3 minimality-decider finds the first minimal program $P_M^{min}$ with $M > N$
4 run $P_M^{min}$
```

Minimal programs

By instructions 3 and 4 above the program T_N stops in finite time,

$$\mathrm{output}(T_N) = \mathrm{output}(P_M^{min}) = M \text{ and}$$

the program P_N^{min} is minimal.

We have arrived to a contradiction because for large enough N, by instruction 2,

$$\mathrm{length}(T_N) \leq \log_2 N + \text{ constant } < N < \mathrm{length}(P_N^{min}).$$

Program naming

A *name* for a string x is a pair

$$\langle [P], w \rangle,$$

where $[P]$ is a *code*[6] of a program P and w is a string such that

$$P(w) = x,$$

that is, the computation $P(w)$ stops and produces as output x. For simplicity, we write P also for $[P]$.

[6] We can algorithmically assign binary strings to programs, see for example [12].

Prefix-freeness

Let $B = \{0, 1\}$. A set S of strings is prefix-free if no string in S is a proper prefix of a string in S.

- The sets $\{x \in B^* \mid |x| = n\}$, $\{1^n0 \mid n \geq 1\}$ are prefix-free.
- The set $\{x_1x_1x_2x_2 \ldots x_nx_n01 \mid x_1x_2 \ldots x_n \in L\}$ is prefix-free for every $L \subseteq B^*$.
- The set $\{x_10x_20 \ldots x_n1 \mid x_1x_2 \ldots x_n \in L\}$ is prefix-free for every $L \subseteq B^*$.
- B^* is not prefix-free.

Program naming

The length of $\langle P, w \rangle$ is the size of the name. How can we define this "length"? We use a prefix-free coding: if $P = p_1 p_2 \ldots p_n$, then we encode $\langle P, w \rangle$ by the string

$$p_1 0 p_2 0 \ldots p_{n-1} 0 p_n 1 w. \tag{1}$$

A simple algorithm can test whether a string x is of the form (1) and, in the affirmative, can compute the unique strings $p_1 p_2 \ldots p_n$ and w such that $x = p_1 0 p_2 0 \ldots p_{n-1} 0 p_n 1 w$.

Furthermore, $\mathrm{length}(\langle P, w \rangle) = 2\,\mathrm{length}(P) + \mathrm{length}(w)$.

Program naming

We consider **a** shortest (*minimal*) name $\langle M, w \rangle$ for x (several such strings may exist); its (unique) length is denoted by $K(x)$ and is called the *algorithmic complexity or Kolmogorov-Chaitin complexity* of x.

We can easily name a string x by placing a copy of itself directly into an analogue of PROGRAM(P_N): $P_x = x$.

The length of this name is $2\,\mathrm{length}(P) + \mathrm{length}(x)$.

This name says very little about the “structure” of x; a significantly shorter name will tell us that the information in x *can be compressed*, so its amount of information cannot be very large.

Program naming

This suggests that the algorithmic complexity is a measure of the amount of information in strings. As the PROGRAM(P_x) computes the identity function $P_x = x$, we deduce the following inequality: there exists a constant c such that for all strings x we have

$$K(x) \leq 2\,\mathrm{length}(P) + \mathrm{length}(x) = c + \mathrm{length}(x), \qquad (2)$$

where $c = 2\,\mathrm{length}(P)$; the inequality above follows from the minimality of K.

Program naming

Formula (2) shows that algorithmic complexity is never much larger than the length of the string itself. Of course, the algorithmic complexity of *some* strings can be much shorter than the lengths of the strings.

For example, the information in the strings

$$A{=}0101010101010101010101010101010101010101010101010101010101010101,$$

$$B{=}0101101010100101010110101010010101011010101001010101101010100101,$$

appears to be shorter than their lengths.

Program naming

But, is this case for all strings? Each name is a binary string, so the number of names of length less than n is at most the sum of the numbers of strings length $0, 1, \ldots, n-1$:

$$1 + 2 + 4 + \cdots + 2^{n-1} = 2^n - 1.$$

The number of minimal names is less than the number of strings of length n – which is 2^n, so at least one string of length n has to have $K(x) \geq n = |x|$.

We say that a string x is c-compressible ($1 \leq c < |x|$) if $K(x) \leq |x| - c$. If x is not c-compressible, then x is incompressible by c.

Program naming

As a corollary we show that at least $2^n - 2^{n-c+1} + 1$ strings of length n are incompressible by c. Indeed, at most $2^{n-c+1} - 1$ strings of length n are c-compressible (because at most that many descriptions of length at most $n - c$ exist), hence the remaining

$$2^n - (2^{n-c+1} - 1)$$

strings of length n are incompressible by c.

The end of science

In June 2008, C. Anderson, former editor-in-chief of *Wired Magazine*, wrote a provocative essay titled "The End of Theory: The Data Deluge Makes the Scientific Method Obsolete" in which he states that

with enough data, the numbers speak for themselves.

And, he continued:

Correlation supersedes causation, and science can advance even without coherent models, and unified theories.

Along this path, Anderson's followers have also stressed the role of big data in replacing the scientific method.

From causation to correlation

Sufficiently powerful algorithms can now explore huge databases and can find therein correlations and regularities.

The strength and generality of this method relies on the size of data: the larger the data, the more powerful and effective is the method grounded on computationally discovered correlations.

There is no need to theorise, understand, criticise . . . the sense of the discovered correlations: No semantic or causal analysis is required. See [30].

The end of understanding

Big data algorithms like deep learning are notoriously opaque: they don't give insight and understanding which are essential in maths (and science).

Yet, **mathematician** S. Strogatz wrote in New York Times 2018

> *Maybe eventually our lack of insight would no longer bother us. After all, AlphaInfinity [a hypothetical descendent of AlphaZero] could cure all our diseases, solve all our scientific problems and make all our other intellectual trains run on time. We did pretty well without much insight for the first 300,000 years or so of our existence as Homo sapiens. And we'll have no shortage of memory: we will recall with pride the golden era of human insight, this glorious interlude, a few thousand years long, between our uncomprehending past and our incomprehensible future.*

Data deluge

From scarcity and difficulty to find data (and information) we now have a deluge of data.

Would you be surprised that more than 90% of all the data throughout history was generated in the past two years?

Data deluge

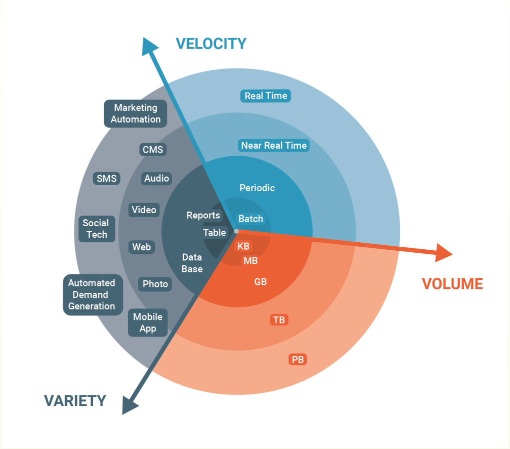

Figure: 3 V's of big data: volume, velocity, and variety

Data deluge

- **V**eracity: poor.

A. Navathe (University of Pennsylvania) (quoted in M. L. Page. Meet your digital doctor, *New Scientist* 20 (2019)) said that

> *"... you can't just feed a whole data to an AI and expect it to produce brilliant insights, because of the inherent biases in data sets. ... The data that they are seeing is filtered through the eyes and judgements of humans."*

This is why IBM Watson for Oncology incorrect and unsafe decisions are determining doctors to abandon it. See [31] and G. Smith. An Epic Failure: Overstated ai Claims in Medicine, August 2021.

Politics is dirty. So is big data. Cleansing it is complex, expensive and hard to automatise.

Data deluge

- **V**elocity: *growing data rate is quickly outpacing the ability to decipher it.*
- **U**nderstanding: ?

Correlations

A correlation is a relationship between two variables taking (numeric) values: if one variable value increases, then the other one also increases (or decreases) "in the same way"

"as the temperature goes up, ice cream sales also go up",
"as temperature decreases, the velocity of molecules decreases".

Variables whose values cannot be expressed in the same units like

per capita income and cholera incidence

can be correlated, and their relationship is measured with a *correlation coefficient* with values in $[-1, +1]$.

Correlations are useful because of their potential predictive power:

> *Use or act on the value of one variable to predict or modify the value of the other.*

Correlations

If A is correlated to B, then

- A could cause B, or
- B could cause A, or
- C could cause both A and B, or
- hidden variables could have been overlooked, or
- data could be defective, or
- it could be a coincidence, or
- ?

Correlations

Since the 10th century AD, Chinese doctors remarked that mild smallpox infection could prevent more severe illnesses. They were able to confirm this correlation by successfully experimenting smallpox inoculation on large numbers of individuals.

Well before our era, the regularities of occurrences of Halley's comet where also observed in China. They allowed ancient Chinese astronomers to predict future occurrences of the comet and even to establish a correlation between occasional delays and the presence of large planets along its path.

Correlations

These were extraordinary observations of non-obvious correlations and their practical relevance. Yet, further scientific knowledge has been added to them since:

1. the understanding of microbial infections and vaccines in the 19th century,
2. Clairaut's mathematical derivation of the orbit of Halley's comet, and its delays, on the grounds of Newton's laws and equations, in the 18th century.

These scientific works could frame these correlations into relatively robust theories that gave them scientific meaning.

This method could also detect spurious correlations – like the relation between the orientation of the comet's tail and the Emperor's chances of a military victory – also claimed by the same ancient masters.

The myth of correlation

Pitfalls of exaggerating the value of prediction in case of correlated observables have been discussed in the literature for many years.

The example

> *In 2012, the* New England Journal of Medicine *published a paper claiming that chocolate consumption could enhance cognitive function. The basis for this conclusion was that the number of Nobel Prize laureates in each country was strongly correlated with the per capita consumption of chocolate in that country.*

was discussed in [33, 26].

The myth of correlation

A 2010 study conducted by Harvard economists Carmen Reinhart and Kenneth Rogoff reported a correlation between a country's economic growth and its debt-to-GDP ratio. In countries where public debt is over 90 percent of GDP, economic growth is slower, they found. The study gathered a lot of attention. Google Scholar listed 1218 citations at the time of writing, and the statistic was used in US economic policy debates in 2011 and 2012.

The research didn't hold up on replication: . . . certain years had been excluded from the analysis, and that a spreadsheet error had a dramatic influence on the results. Correcting the errors actually seemed to reverse the main finding and alter many others. But by the time that became clear, the original study had already attracted widespread attention and affected policy decisions (in Europe, for example).

The myth of correlation

Spurious high correlations can be hilarious ...

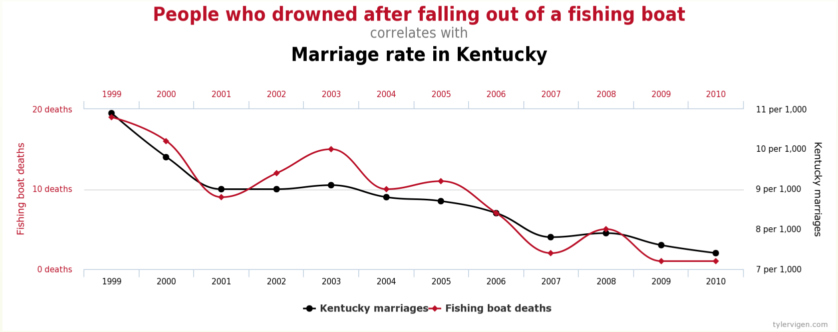

Figure: Spurious correlation with $r = 0.952407$

The myth of correlation

The success of the website Spurious Correlations maintained by military intelligence analyst and (former) Harvard Law student evolved into the “ridiculous book”

Tyler Vigen. *Spurious Correlations*, May 2015

which illustrates – through hilarious graphs – the well-known fact that correlation does not imply causation.

Does causation imply correlation?

The answer is no.

Think about the relation between pressing a car's accelerator and its velocity.

A skilled driver can drive a powerful car with constant speed uphill and downhill: a passenger observing the accelerator pedal (going up or down) and the car going uphill or downhill, but knowing nothing else about driving, may conclude falsely that the position of the pedal and the slope have no effect on the car's velocity.

A typology of correlations

“[S]ciences are essentially an exercise in observing correlations, then proposing explanations for them”.

They operate with the following types of correlations:

1. two types of *local* correlations:
   1.1 one can be accounted for as an influence of one event on another (“if it is rainy, then the ground is wet”)
   1.2 another one can be accounted to common local causes (“gum disease, oral cancer, loss of taste, mouth sores and bad breath are likely to affect youths with smoking habits”)
2. *non-local* correlations (like quantum correlations of entangled qubits)
3. ?

Ramsey theory

Ramsey theory, named after the British mathematician and philosopher

Figure: Frank P. Ramsey (1903–1930)

studies the conditions under which order must appear.

Ramsey theory

Ramsey theory typically asks questions of the form:

> *Are there binary (finite) strings with no patterns/correlations?*

Or, to say it in a dual way:

> *How many elements of some structure must there be to guarantee that a* given regularity *will hold?*

Ramsey theory answers in the negative the first question by providing precise answers to the second:

> *Complete disorder is an impossibility. Every large set of numbers, points or objects necessarily contains a highly regular pattern.*

Ramsey theory: an example

Let $s_1 \cdots s_n$ be a binary string. A monochromatic arithmetic progression of length k is a substring

$$s_i s_{i+t} s_{i+2t} \cdots s_{i+(k-1)t},$$

for $1 \leq i$ and $i + (k-1)t \leq n$ with all characters equal for some $t > 0$.

The string (of length 8)

$$01100110$$

contains no monochromatic arithmetic progression of length 3 because the positions 1, 4, 5, 8 (for 0) and 2, 3, 6, 7 (for 1) do not contain a monochromatic arithmetic progression of length 3.

Ramsey theory: an example

The strings

011001100

and

011001101

have each a monochromatic arithmetic progression of length 3: 1, 5, 9 for 0 and 3, 6, 9 for 1.

In fact, one can prove that

> *all 512 binary strings of length 9 have a monochromatic arithmetic progression of length 3.*

Van der Waerden correlations

Finite Van der Waerden theorem. *For any positive integers k and c there is a positive integer γ such that every string, made out of c digits or colours, of length more than γ contains a monochromatic arithmetic progression of length k.*

The theorem states that all sufficiently long strings of digits/colours, taken from a finite set, have long enough arithmetic progressions of the same digit/colour, i.e.

> *a sequence of equi-distributed positions of the same digit/colour (monochromatic) of a pre-given length.*

Van der Waerden correlations

Van der Waerden theorem shows that in any coding of a large enough arbitrary database into a string of digits, there will be correlations of a pre-determined arbitrary length: the same digit sitting on a long sequence of equi-distributed positions.

These equi-distributed positions of the same digit are not due to the specific information coded in the data nor to a particular encoding: they are determined only by the large size of data.

Yet, a program may see such pre-chosen correlations as a “law” or as sufficient information to replace a law.

What is a spurious correlation?

According to Oxford Dictionary, *spurious* means

> *Not being what it purports to be; false or fake. False, although seeming to be genuine. Based on false ideas or ways of thinking.'*

The (dictionary) definition of the word "spurious" is semantic, that is, it depends on an assumed theory: one correlation can be spurious according to one theory, but meaningful with respect to another one.

A stronger definition of spurious correlation

Can we give a definition of “spurious correlation” which is independent of *any* theory?

Following [8]

> *a correlation is spurious if it appears in a randomly generated string/sequence.*

Obviously, such a spurious correlation is “meaningless” according to any reasonable interpretation because, by construction, all data in the string/sequence is generated at random.

The deluge of spurious correlations

- Do spurious correlations in the above sense exist?

  The answer is affirmative: Van der Waerden (more generally, Ramsey-type) correlations are spurious.

- Is the size of the set of spurious correlations "small", meaning that only insignificantly many correlations are spurious?

  The answer is negative and one way to prove it is to use algorithmic information theory [3].

  Informally, for large *n*, very few strings of length *n* are compressible or, dually, most strings are random, if one takes incompressibility as a criterium for randomness.

The deluge of spurious correlations: AIT analysis

Fix n, so we have 2^n n-bit-strings.

1. A k-correlation in an n-bit-string x is a substring of length k of the string x. There are $n - k + 1$ possible correlations in x.
2. A correlation is spurious if it appears in **an** incompressible n-bit-string, hence a non-spurious correlation should not appear in **any** incompressible n-bit-string.
3. A string x is c-compressible $(1 \leq c < |x|)$ if $K(x) \leq |x| - c$.
4. At least $2^n - 2^{n-c+1} + 1$ strings of length n are incompressible by c.

The deluge of spurious correlations: AIT analysis

The probability that a string x of length n is c-compressible is

$$\#\{|x| = n : K(x) \leq n - c\} \cdot 2^{-n} \leq (2^{n-c+1} - 1) \cdot 2^{-n} < 2^{1-c},$$

so the probability that a string x of length n is not c-compressible is

$$\#\{|x| = n : K(x) > n - c\} \cdot 2^{-n} \geq 1 - 2^{1-c}.$$

The deluge of spurious correlations: AIT analysis

Take now $c = n/2$, a mild compression degree.

$$\begin{aligned} \#\{|x| = n \mid K(x) \leq n - n/2\} \cdot 2^{-n} &= \#\{|x| = n \mid K(x) \leq n/2\} \cdot 2^{-n} \\ &\leq 2^{1-n/2} \longrightarrow_{n\to\infty} 0, \end{aligned}$$

while the probability that a string x of length n is not $n/2$-compressible is

$$\begin{aligned} \#\{|x| = n \mid K(x) \geq n - n/2\} \cdot 2^{-n} &= \#\{|x| = n \mid K(x) \geq n/2\} \cdot 2^{-n} \\ &\geq 1 - 2^{1-n/2} \longrightarrow_{n\to\infty} 1. \end{aligned}$$

The deluge of spurious correlations: AIT analysis

Accordingly, the probability of a spurious correlation in an n-bit-string is greater than

$$(n-k+1)\times(1-2^{1-n/2}) \longrightarrow_{n\to\infty} \infty,$$

and the probability of a non-spurious correlation in an n-bit-string is less than

$$(n-k+1)\times 2^{1-n/2} \longrightarrow_{n\to\infty} 0.$$

The deluge of spurious correlations

Here is an illustration. Recall that the best average rate of algorithmic compressibility is about 86.4% (i.e. a string x is compressed into Zip(x) whose length is about 86.4% of the length of x). The probability that a binary string x of length 2048 (relatively short) is reduced by 13.6% is smaller than

$$2^{-\frac{136}{1000}\cdot 2048} < 10^{-82},$$

a very, very small number ($10^{82} \approx$ the number of hydrogen atoms in the universe), so with probability more than

$$1 - 10^{-82},$$

a database will be Zip-incompressible, a symptom of randomness. All its correlations will be spurious!

The deluge of spurious correlations: A more general result

For every real number α in $(0, 1)$,

> *the number of strings x of length n having the complexity $K(x)$ smaller than $n - \alpha \cdot n$ is smaller than $2^{(1-\alpha)n} - 1$.*

Thus,

> *the probability that a string x of length n has $K(x) < n - \alpha \cdot n$ is smaller than $2^{-\alpha \cdot n}$,*

a quantity which converges exponentially to 0 when $n \to \infty$.

Grand Challenge (Research Question)

Find another way (based also on the difference in size between compressible and incompressible strings) to argue that spurious correlations outnumber the meaningful ones.

References

[1] C. Bernhardt.
Turing's Vision The Birth of Computer Science.
The MIT Press, Cambridge, Mass, 2016.

[2] L. Bienvenu, D. Desfontaines, and A. Shen.
What percentage of programs halt?
In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *Automata, Languages, and Programming I*, volume LNCS 9134, pages 219–230.
Springer Berlin, Heidelberg, 2015.

[3] C. Calude.
Information and Randomness—An Algorithmic Perspective.
Springer, Berlin, 2002 (2nd ed.).

[4] C. S. Calude.
Unconventional computing: A brief subjective history.
In *Advances in Unconventional Computing*, pages 855–864. Springer, 2017.

[5] C. S. Calude and D. Desfontaines.
Anytime algorithms for non-ending computations.
International Journal of Foundations of Computer Science, 26(4):465–475, 2015.

References (cont.)

[6] C. S. Calude and M. Dumitrescu.
A probabilistic anytime algorithm for the halting problem.
Computability, 7(2-3):259–271, 2018.

[7] C. S. Calude and M. Dumitrescu.
A statistical anytime algorithm for the halting problem.
Computability, 9(2):155–166, 2020.

[8] C. S. Calude and G. Longo.
The deluge of spurious correlations in big data.
Foundations of Science, 22(3):595–612, 2017.

[9] C. S. Calude and M. A. Stay.
Most programs stop quickly or never halt.
Advances in Applied Mathematics, 40(3):295–308, 2008.

[10] G. J. Chaitin.
The Unknowable.
Springer, Singapore, 1999.

[11] M. Davis.
Computability and Unsolvability.
McGraw-Hill, New York, 1958.

References (cont.)

[12] M. Davis.
What is a computation?
In C. S. Calude, editor, *Randomness and Complexity. From Leibniz to Chaitin*, pages 89–113. World Scientific Publishing, Singapore, 2007.

[13] M. Davis.
The Universal Computer: the Road from Leibniz to Turing.
W. W. Norton Company, New York, 2017.

[14] L. De Mol.
Turing Machines.
In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

[15] T. Franzén.
Gödel's Theorem: An Incomplete Guide to Its Use and Abuse.
AK Peter, Wellesley, Massachusetts, 2005.

[16] K. Gödel.
Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I.
Monatshefte für Mathematik, 38:173–198, 1931.

References (cont.)

[17] J. Grass.
Reasoning about computational resource allocation. An introduction to anytime algorithms.
Magazine Crossroads, 3(1):16–20, 1996.

[18] S. Köhler, C. Schindelhauer, and M. Ziegler.
On approximating real-world halting problems.
In M. Liskiewicz and R. Reischuk, editors, *Fundamentals of Computation Theory, 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005, Proceedings*, volume 3623 of *Lecture Notes in Computer Science*, pages 454–466. Springer, 2005.

[19] L. Kronecker.
Zur theorie der elimination einer variablen aus zwei algebraischen gleichungen.
Monatsberichte der Königlich Preussischen Akademie der Wissenschaften, pages 535–600, 1881.

[20] G. Longo.
Interfaces of incompleteness.
In G. Minati, M. R., and A. E. Pessa, editors, *Systemics of Incompleteness and Quasi-Systems*, pages 3–55. Springer Nature, June 2019.

References (cont.)

[21] S. Lucas.
The origins of the halting problem.
Journal of Logical and Algebraic Methods in Programming, 121:100687, 2021.

[22] N. Lynch.
Approximations to the halting problem.
Journal of Computer and System Sciences, 9(2):143 – 150, 1974.

[23] Y. I. Manin.
Renormalisation and computation II: time cut-off and the Halting Problem.
Mathematical Structures in Computer Science, (22):729–751, 2012.

[24] H. Putnam.
Reason, Truth and History.
Cambridge University Press, Cambridge, 1981.

[25] B. Russell and A. N. Whitehead.
Principia Mathematica.
Cambridge University Press (three volumes), 1910, 1912, 1913.

[26] G. Smith.
The AI Delusion.
Oxford University Press, 2018.

References (cont.)

[27] P. Smith.
An Introduction to Gödel's Theorems.
Cambridge University Press, Cambridge, UK, 2013.

[28] E. Snapper.
The three crises in mathematics: Logicism, intuitionism and formalism the three crises in mathematics: Logicism, intuitionism and formalism.
Mathematics Magazine, 54(4):207–216, 1979.

[29] F. Soler-Toscano, H. Zenil, J.-P. Delahaye, and N. Gauvrit.
Calculating Kolmogorov complexity from the output frequency distributions of small turing machines.
PLoS ONE, 9(5):e96223, 2014.

[30] S. Succi and P. V. Coveney.
Big data: the end of ther scientific method?
Phil. Trans. R. Soc. A, 377(2142):20180145, 2019.

[31] E. Topol.
Deep Medicine.
Basic Books, 2019.

References (cont.)

[32] A. M. Turing.
On computable numbers, with an application to the Entscheidungsproblem.
Proceedings of the London Mathematical Society, Series 2, 42, 43:230–265, 544–546, 1936-7 and 1937.

[33] V. Velickovic.
What everyone should know about statistical correlation.
Scientific American, 103(1):26, 2015.

[34] H. Zenil.
Computer runtimes and the length of proofs.
In M. J. Dinneen, B. Khoussainov, and A. Nies, editors, *Computation, Physics and Beyond*, volume LNCS 7160, pages 224–240. Springer Berlin, Heidelberg, 2012.

[35] H. Zenil and J.-P. Delahaye.
On the algorithmic nature of the world.
In G. Dodig-Crnkovic and M. Burgin, editors, *Information and Computation. Essays on Scientific and Philosophical Understanding of Foundations of Information and Computation*, pages 477–499. World Scientific, 2010.