Chapter 4
Unpredictability and Randomness

Rade Vuckovac

Abstract Randomness is to a certain degree opposite to determinism. This essay tries to put those two on the same page. It argues the premise where randomness is a consequence of a deterministic process. It also provides yet another viewpoint on the hidden variable theory.

4.1 Introduction

Any discussion about randomness should include a description of unpredictability. A quoted part from the explanation on true randomness makes a good starting point for our inquiry (bold emphasis added):

> If outcomes can be determined (by hidden variables or whatever), then any experiment will have a result. More importantly, any experiment will have a result whether or not you choose to do that experiment, because the result is written into the hidden variables before the experiment is even done. Like the dice, if you know all the variables in advance, then you don't need to do the experiment (roll the dice, turn on the accelerator, etc.). The idea that every experiment has an outcome, regardless of whether or not you choose to do that experiment is called "the reality assumption", and it should make a lot of sense. If you flip a coin, but don't look at it, then it'll land either heads or tails (this is an unobserved result) and it doesn't make any difference if you look at it or not. In this case the hidden variable is "heads" or "tails", and it's only hidden because you haven't looked at it.
>
> It took a while, but hidden variable theory was eventually disproved by John Bell, who showed that there are lots of experiments that cannot have unmeasured results. Thus the results cannot be determined ahead of time, so there are no hidden variables, and the results are truly random. **That is, if it is physically and mathematically impossible to predict the results, then the results are truly, fundamentally random.** [1]

Some deterministic systems which show unpredictable behaviour are in:

R. Vuckovac (✉)
Department of Information and Communication Technologies, Engineering School, Universitat Pompeu Fabra, Barcelona, Spain
e-mail: rade.vuckovac@gmail.com

A. Aguirre et al. (eds.), *Undecidability, Uncomputability, and Unpredictability*,
The Frontiers Collection, https://doi.org/10.1007/978-3-030-70354-7_4

Chaos Theory; It shows one exciting feature not usually found in classical systems. Predicting a long-term state of a system combined with its approximate initial conditions is a difficult if not impossible task. Edward Lorenz sums it as:

> Chaos: When the present determines the future, but the approximate present does not approximately determine the future.

Then following details show unpredictability in a little bit more detail:

- *Butterfly Effect;* An interesting and not widely emphasised point about Butterfly Effect is [2, 3]:

  > Even a tiny change, like the flap of a butterfly's wings, can making a big difference for the weather next Sunday. This is the butterfly effect as you have probably heard of it. But Edward Lorenz actually meant something much more radical when he spoke of the butterfly effect. He meant that for some non-linear systems you can only make predictions for a limited amount of time, even if you can measure the tiniest perturbations to arbitrary accuracy.

- *n-body problem;* Even in Newton times, the motions of more than two orbiting bodies were considered as a problem (Fig. 4.1). Currently, we are left with numerical methods and simulations. The former is an approximation and butterfly effect prone. The later are basically computational experiments, which are the preferable option for *n-body* system investigation [4, 5].

Cellular Automation (CA); CA is a deterministic model containing grids, populated by cells. The grids are arranged in one or two-dimensional space. An evolution rule governs how the initial state of cells evolve to the next generation. Figure 4.2 shows one dimensional CA rules and an evolution history. One of the interesting CA features is the concept of Computational Irreducibility (CI). It proposes that the only way to determine the future state of CA is to run it, which is very similar to "the results cannot be determined ahead of time" principle in hidden variable argumentation.

4.2 CA a Closer Look

While Chaos Theory provides us with dynamical systems showing unpredictability behaviour, the CI (computational irreducibility) principle is probably more accessible for the discussion.

Wolfram's rule 30 CA is a good starting point. Figure 4.2 shows the transition rules on the top. Every case prescribes how a cell (black or white) is transformed depending on the previous cell state and its neighbouring cells. Row 1 is the initial state of CA. Rows 2, 3, 4 ... are consecutive evolved generations. A next-generation cell is derived from a previous cell and its neighbours. For example, the cell (row 4, column 13) is derived from case 7. When a cell does not have an above row neighbour, the cell from other end is considered. So, for example, the cell (16; 1) uses case 7 (again) because the top left neighbour is the cell (15; 31).

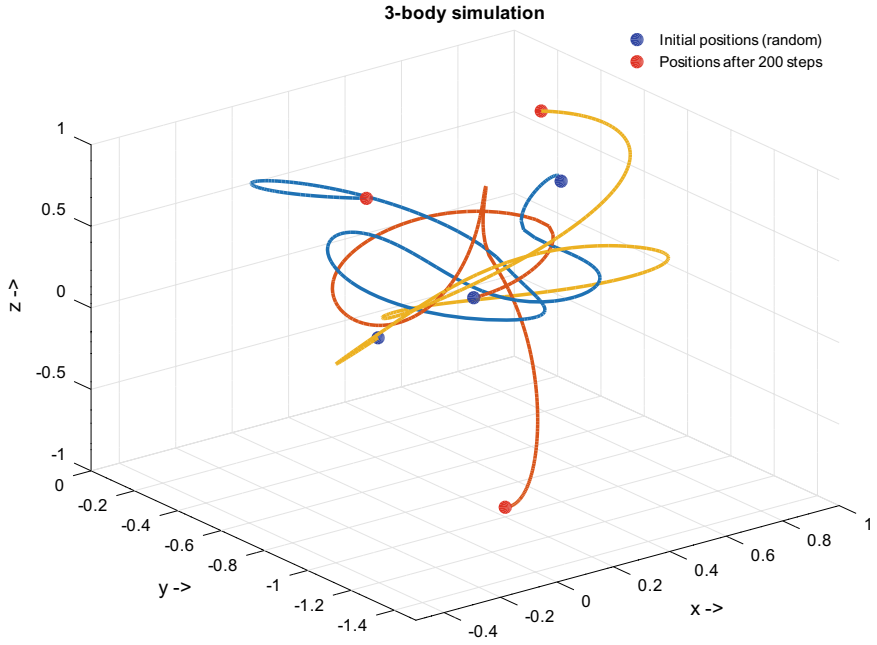

Fig. 4.1 An example of chaotic behaviour when the system has more than two bodies

There are a contest and prize for solving three CA rule 30 problems [6]. All three of them are relevant for the randomness discussion:

Problem 1: *Does the centre column always remain non-periodic?* The centre column is column 16 outlined red in Fig. 4.2. The period has been checked for the $\approx$ first billion steps, and the centre column does not show periodicity. In effect, it has very similar properties to π.

Problem 2: *Does each colour of cell occur on average equally often in the centre column?* The ratio of black and white cells approaches 1:1 when the step iterations increase. After one billion steps, the centre column has 500025038 black and 499974962 white cells. In a sense, it mimics a fair coin flipping exercise.

Problem 3: *Does computing the nth cell of the centre column require at least $O(n)$ computational effort?* Stephen Wolfram strongly suspects that rule 30 is computationally irreducible (CI). In that case, even if the initial state and rules of transformation are known, the quickest way to see the future of the state is to run CA (to do the experiment).

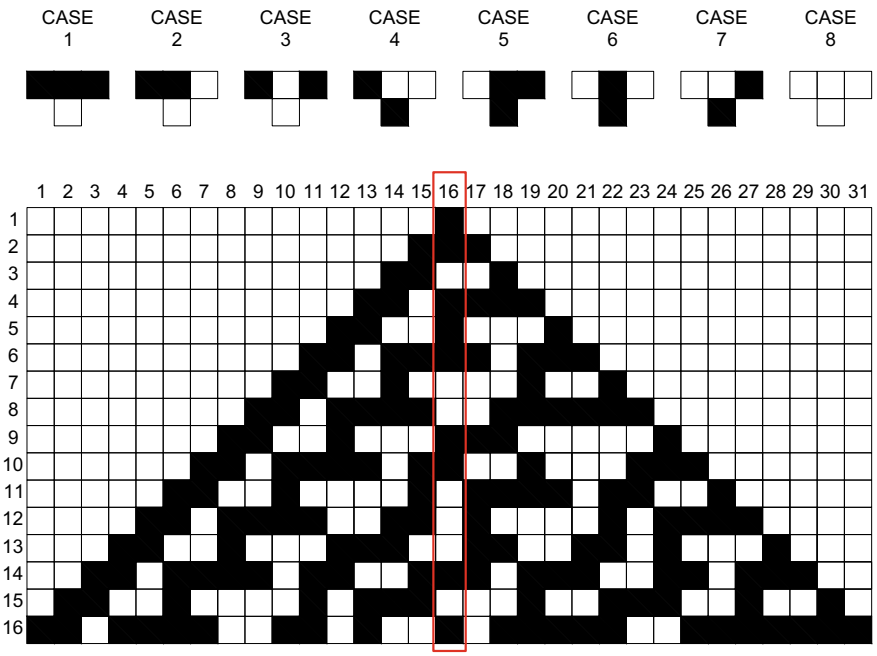

Fig. 4.2 CA rule 30. Transformation rules and evolution history. Column 16 acts as a random sequence

4.3 Conditional Branching

While chaos theory and CA provide evidence of future inaccessibility, there is an even more persuasive argument where we can not acquire the future state of some systems without experimenting.

Conditional branching, the underlying algorithmic construct is the essential argumentation ingredient. Conditional branching with other two primitives, sequence and iteration, provides all necessary blocks to build any algorithm imaginable [7]. It is an if-else statement in a program. When the program reaches if-else command, it evaluates some state and depending on evaluation continues with an if or an else path. Usually, when the program is made, all the possible inputs and they eventual execution paths (EP) are thoroughly defined. Figure 4.4 shows domain partition, where inputs are partitioned according to the joint EP.

For example, the $3x + 1$ problem one step is:

$$F(x) = \begin{cases} f(x) \; if & x \equiv 0 \bmod 2 \\ g(x) \; if & x \equiv 1 \bmod 1 \end{cases} \tag{4.1}$$

where $f(x) = x/2$, and $g(x) = (3x + 1)/2$. There we know that even input will be executed by $f(x)$ and odd with $g(x)$.

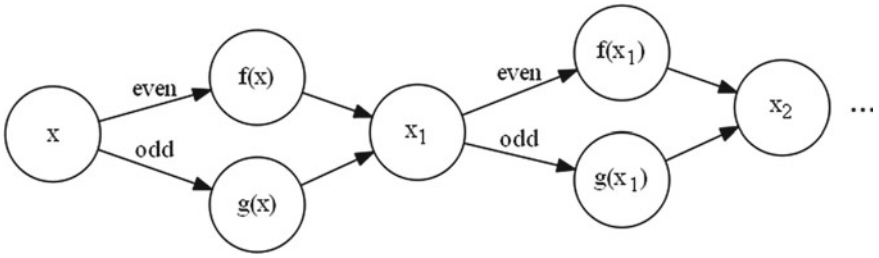

Fig. 4.3 The $3x + 1$ composite function; $f(x) = x/2$ and $g(x) = (3x + 1)/2$

Table 4.1 For two iterations of $3x + 1$ (Fig. 4.3), we have four partitions of natural numbers depending on execution path and corresponding composite

Domain ($\mathbb{N}_1$)	Execution path (EP)	Composite function
Partition A	Even-even	$f(f(x))$
Partition B	Even-odd	$g(f(x))$
Partition C	Odd-odd	$g(g(x))$
Partition D	Odd-even	$f(g(x))$

The problem starts when this procedure is iterated (Fig. 4.3 and Table 4.1). Every iteration doubles the amount of unique EP. For example if our inputs are 64-*bit* integers and we iterate Eq. 4.1 64 times. The number of possible paths is $EP \leq |2^{64}|$. In that case domain partition by execution paths (Fig. 4.4) seems an impossible task and that is probably the cause of why this problem is still not solved despite dealing with very basic arithmetic [8].

Now we can ask: Is every program with multiple execution paths domain partition-able?

Informal Theorem 1. *There exists at least one algorithm with multiple execution paths where the knowledge of which way execution goes is known only after input evaluation* [9].

Proof We can assume the opposite: every input partitioning can be performed efficiently before execution. On first glance, that is a reasonable statement because when a program is specified correctly, every case behaviour is fully defined in advance. On the other hand, there are at least two problems with this:

- Programs are not always made with a purpose. For example, multiple branching statements could be written haphazardly throughout a program. If it compiles, it will run, but the output behaviour will be unpredictable.
- If every program is partition-able, then branching algorithmic structure is redundant. Practically, we will know which path to execute for some input in advance without the need to test the branching statement.

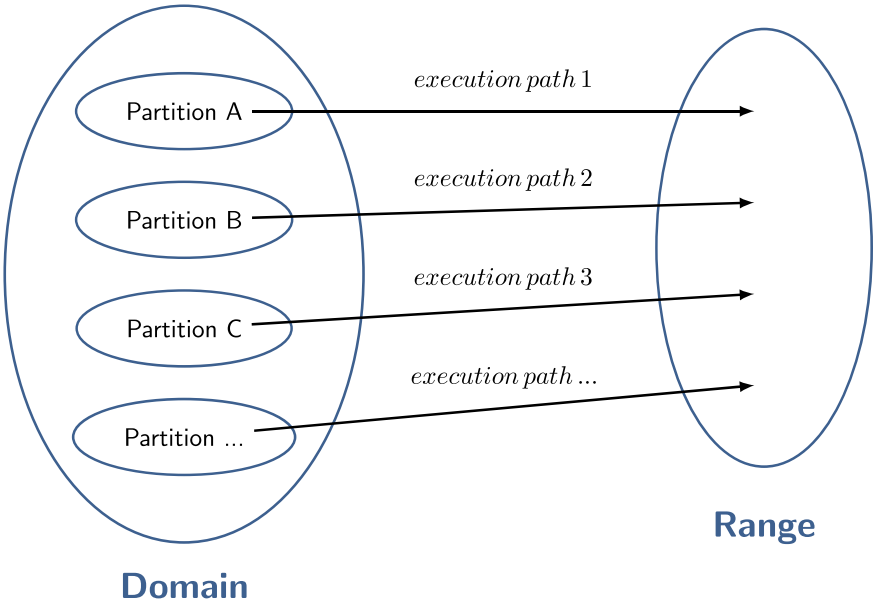

Fig. 4.4 Partitions mappings, when a program has multiple paths of execution

Having algorithms without conditional branching is not what we presently believe:

– Encyclopedia Britannica; Conditional branching entry:

> In Analytical Engine ··· control transfer, also known as conditional branching, whereby it would be able to jump to a different instruction depending on the value of some data. This extremely powerful feature was missing in many of the early computers of the 20th century. [10]

– Wikipedia; Turing Completeness:

> To show that something is Turing complete, it is enough to show that it can be used to simulate some Turing complete system. For example, an imperative language is Turing complete if it has conditional branching. [11] □

4.3.1 Conditional Branching Candidates

While the systems where non-partitionable inputs exist, it is hard to identify one. Some candidates are:

CA rule 30; English description of the rule is in the exact form as Eq. 4.1 (bold added):

> Look at each cell and its right-hand neighbor. **If** both of these where white on the previous step, then take the new color of the cell to be whatever the previous color of its left-hand neighbor was. **Otherwise**, take the new color to be opposite of that. [12]

Rule 30 used to be a random number generator in Wolfram's Mathematica software.

Collatz conjecture; Equation 4.1 is one step of the procedure. The conjecture asserts that every natural number after some steps reaches 1. Some reflections on the problem:

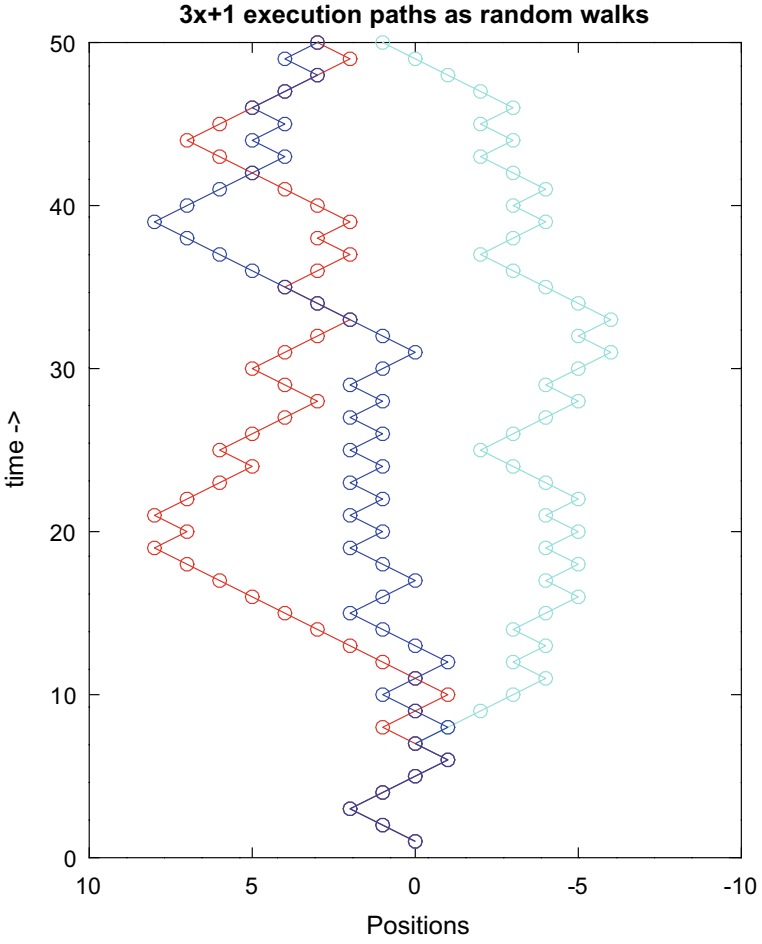

Fig. 4.5 Random walk using 3x + 1 execution paths. When the algorithm executes the odd branch $((3x + 1)/2)$ the left step is taken; otherwise, the walk takes the right step $(x/2)$

> The iterates of the shift function are completely unpredictable in the ergodic theory sense. Given a random starting point, predicting the parity of the n-th iterate for any n is a "coin flip" random variable. ... Empirical evidence seems to indicate that the 3x + 1 function on the domain Z retains the "pseudorandomness" property on its initial iterates until the iterates enter a periodic orbit. This supports the 3x + 1 conjecture and at the same time deprives us of any obvious mechanism to prove it, since mathematical arguments exploit the existence of structure, rather than its absence. [13, p. 18]

Figure 4.5 shows random walks behaviour using branching $3x + 1$ parity.

Möbius function; It is another mathematical object with a branching structure and randomness emergence (Fig. 4.6). It appears that the Riemann hypothesis, random walks and Möbius function are closely related. A Math StackExchange discussion on this topic is here [14]. Equation 4.2 and Fig. 4.6 show the Möbius function, its branching structure and a random behaviour emergence.

$$\mu(n) = \begin{cases} 0 & \text{if 0 if n has a squared prime factor} \\ 1 & \text{if n } = 1 \\ (-1)^k & \text{if n is a product of k distinct primes} \end{cases} \tag{4.2}$$

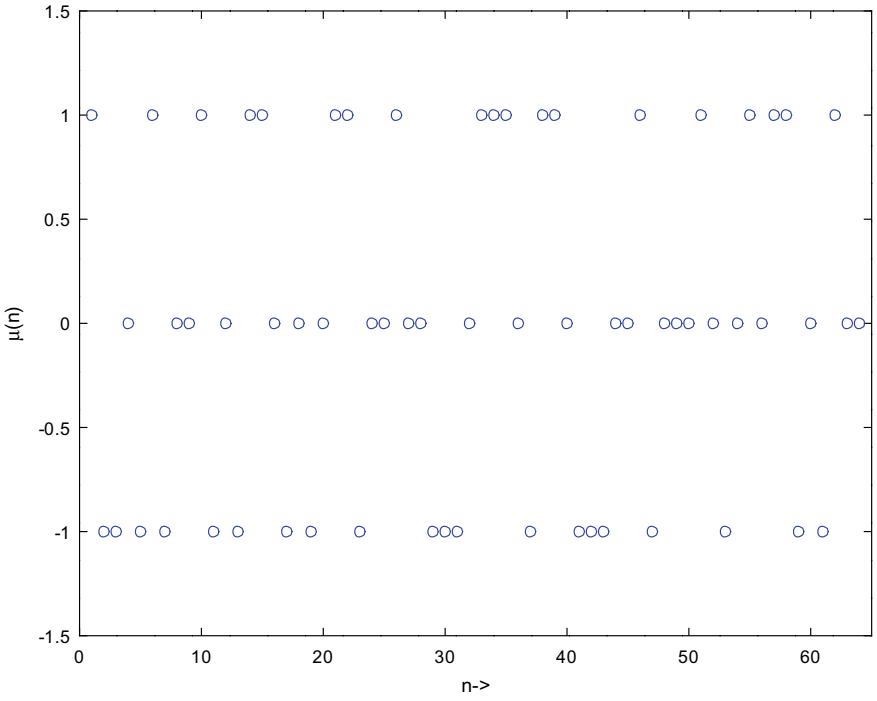

Fig. 4.6 Möbius function and its randomly looking mappings

Branching CA (BCA); This CA was at first designed as a toy model to investigate the if-else structure and its impact on the state transitions. The CA evolution step has now a familiar structure:

$$c' = \begin{cases} c \oplus A_{i+1}, & \text{if } A_{i+2} > A_{i+3} \\ c \oplus \overline{A}_{i+1}, & \text{otherwise} \end{cases} \tag{4.3}$$

where $\oplus$ is exclusive or and $\overline{A}_{i+1}$ is one complement of the cell A_{i+1}. This particular CA is used as a cryptographic primitive for cipher design [15]. Figure 4.7 using *branching CA* illustrates all chaos theory features including the butterfly effect.

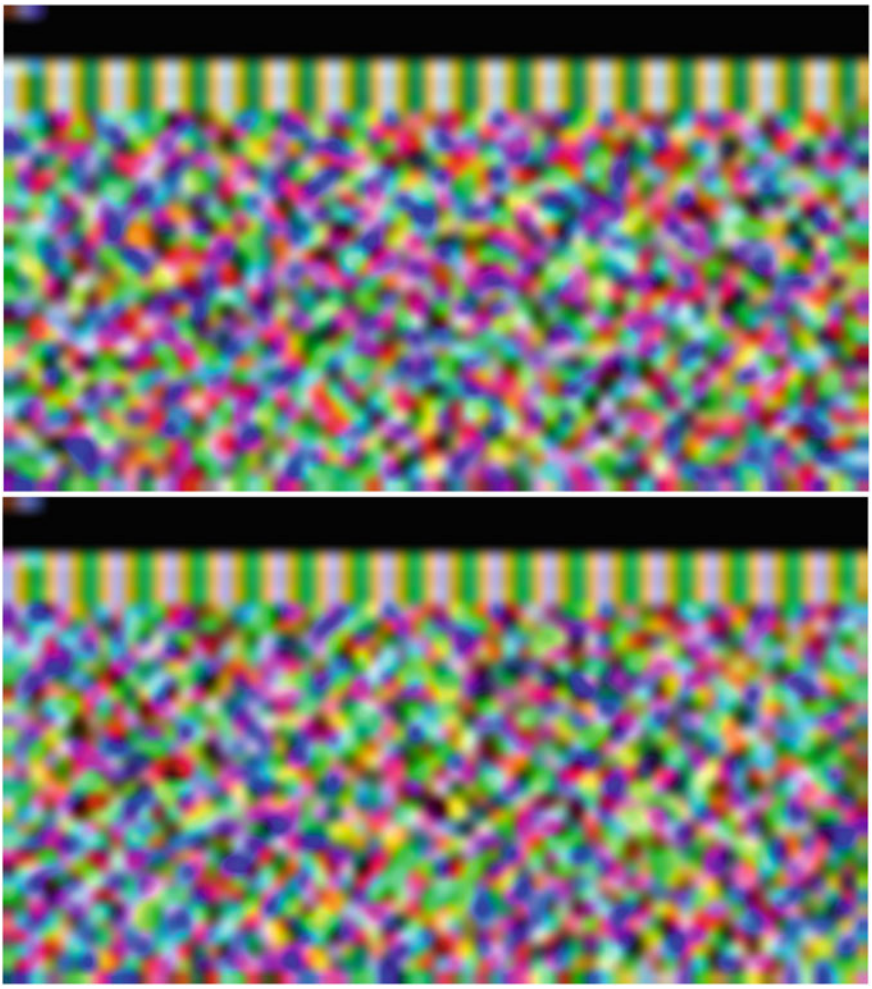

Fig. 4.7 Two cellular automata evolutions where the initial state (top left three pixels plus top black region) differs by just one bit (one in 4096-*bit* state)

4.4 Measuring Randomness

While all mentioned candidates show randomness, the measure of it is a seemingly tricky endeavour. The Turing test for intelligence [16] may provide the base for randomness evaluation. A similar idea was explored in cryptography as well [17].

We have three entities in Turing AI test: a human, a machine and an interrogator. They are in three separate rooms, and their communication is in the questions/answer form where interrogator asks, and the other two parties respond. The evaluation of answers should determine who the human is. If a distinction is inconclusive, the machine could be considered intelligent.

For randomness game we have true-randomness side represented by Random Oracle (RO), the pseudo-randomness side is BCA and an interrogator (IN). As in the AI test, IN communicate with RO and BCA via questions/answer (input/output) format. IN task is to decide which party produces truly random output. If the distinction could not be made, the true and pseudo qualifiers are redundant.

4.4.1 Random Oracle

RO concept provides a specific tabular representation which we can categorise as a true random mapping:

> The following model describes a random oracle:
>
> - There is a black box. In the box lives a gnome, with a big book and some dice.
> - We can input some data into the box (an arbitrary sequence of bits).
> - Given some input that he did not see beforehand, the gnome uses his dice to generate a new output, uniformly and randomly, in some conventional space (the space of oracle outputs). The gnome also writes down the input and the newly generated output in his book.
> - If given an already seen input, the gnome uses his book to recover the output he returned the last time, and returns it again. [18]

The RO model is believed to be an imaginary construct useful in a cryptography secure protocol argumentation. However, it cannot be realised in practice. The use of dices in the description is assumed to be an entirely random process. In other words, if we need RO in practice, we have to use a random function. Its main feature is that every output is an independent, random string. That implies the use of a gigantic input/output table (Table 4.2). That table cannot be compressed, rendering the exercise impractical. More details about RO can be found here [19].

In the finding pseudo game, IN (interrogator) asks a question and RO (the gnome) replies with a truly random answer with the stipulation that same question is answered with the same response.

Table 4.2 Random function

Domain (N)	Range (N)
Input 1	Random string 1
Input 2	Random string 2
Input 3	Random string 3
...	...

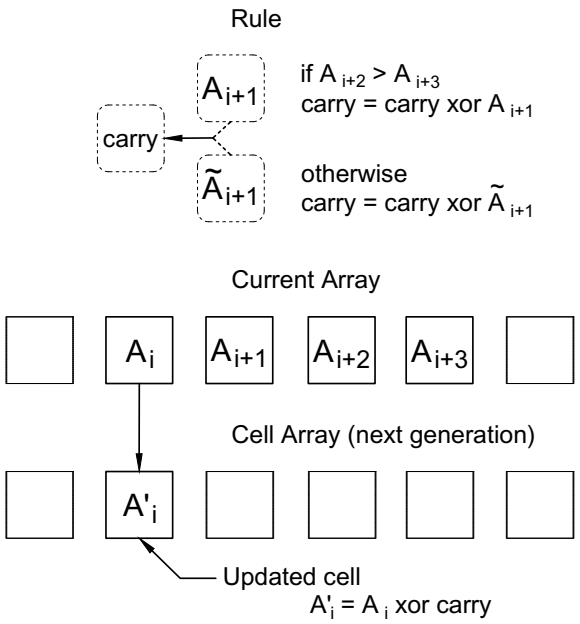

Fig. 4.8 BCA transformation rule; depending on the neighbours relation, third right-hand neighbour or its one complement (all bits are flipped) is XORed (logical exclusive or) with the carry. Updated carry is XORed with the current cell to form a new cell

4.4.2 BCA

BCA candidate (mentioned earlier) represents a pseudo-random oracle. Transformation details are shown in Fig. 4.8. When IN (interrogator) submits the question the following happens: firstly an array is initialised. The input x (question) is copied on the array first cells. The array is evolved for some time, and the part of the last state serves as output y (response). Figure 4.9 red and blue are input and output used in IN and BCA communications.

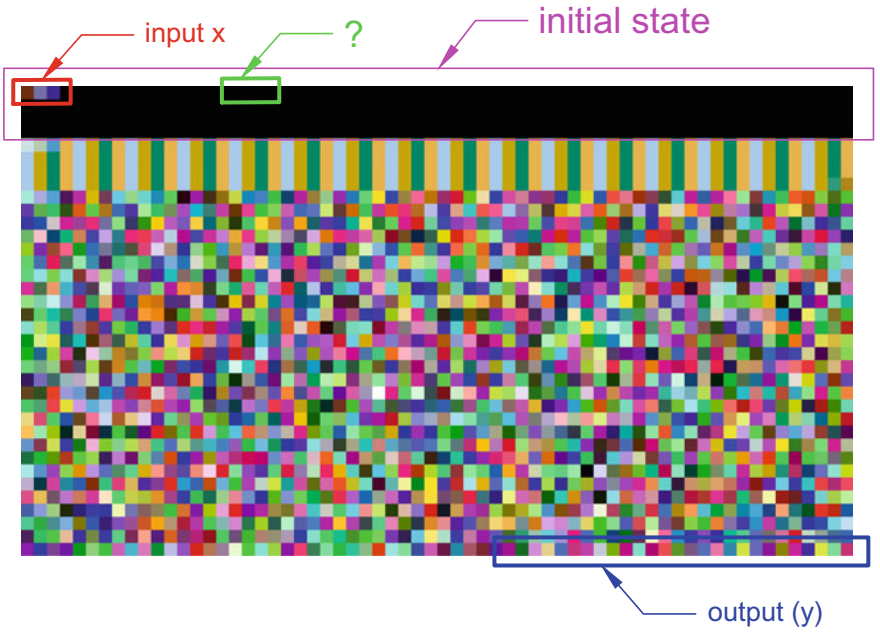

Fig. 4.9 Branching cellular automata (BCA) pretending to be a random oracle (RO)

4.4.3 Distinguishing Chances

The finding pseudo can be played by two set of rules:

- 1st **variant**: IN (interrogator) knows inputs and corresponding outputs only (red and blue Fig. 4.9).
  When IN examines BCA input/output pairs, a couple of exciting things are revealed. Similar inputs, produce very different outputs (Fig. 4.7). When numerous input/output pairs are put in a tabular form, the table starts to reassemble random structure from random function mappings (Table 4.2). This set-up is reminiscent to the Alan Turing challenge scenario:

  > I have set up on the Manchester computer a small programme using only 1,000 units of storage, whereby the machine supplied with one sixteen-figure number replies with another within two seconds. I would defy anyone to learn from these replies sufficient about the programme to be able to predict any replies to untried values. [16]

- 2nd **variant**; we give IN some hints: IN can have full knowledge of BCA algorithm and only partial knowledge of the initial state. Meaning, knowing the magenta part excluding the green part Fig. 4.9. If IN wants to check whether the observed pair comes from BCA by running it, the green part has to be replaced with some provisional part. To find a matching green part IN has to perform an exhaustive search. The same happens if the IN goes backward from the output. Complete

knowledge of the state is needed (provisional plus output) to be able to proceed with steeping. That situation leaves IN with brute force option only.

In both variants, the IN (interrogator) inability to correlate input/output pairs stays. The 2nd variant also prevents IN to check if observed data comes from BCA even the inner working of the input/output transformation is known.[1]

4.5 Speculations

The main point raised in this essay is the computation barrier imposed by conditional branching. That wall forces input evaluation during program execution. Even with known input and algorithm knowledge, predictability of output is impossible. The only way to discover output is to run the algorithm (to evaluate input). The absence of patterns, when input/output pairs of such algorithm were analysed, is caused by that barrier.

If we assume that the conditional barrier is relevant to physical reality and in the context of:

> Thus the results cannot be determined ahead of time, so there are no hidden variables, and the results are truly random.

We can speculate; yes, the results can not be determined ahead of time but, paradoxically, *that does not necessary exclude determinism*.

The misunderstandings might come from determinism definitions because we have different meanings which usually are not noted in argumentations. There are at least two sets of determinism with very other properties. The differentiation occurs when we need to access state details at a particular point of time. For example, imagine, a special kind of video player which can play two movies only: 2-*body* digital simulation and the sequel *n-body* digital simulation. While watching the 2-*body* movie, we can fast forward it if we are bored. While watching *n-body* movie, we notice that fast forward is not working any more, and if we want to see what happens on end, we have to watch the whole movie.

From there, we can identify potential inconsistencies when dealing with determinism. In the accepted hidden variable theories (HVT) narrative, a cursory look at the Bell's inequality [20] and HVT indicates that we have determinism acting locally with the correspondent distribution of probable outcomes which is never confirmed by experiment. Since quantum distribution is matched with observed, we conclude that quantum phenomena are non-deterministic.

On the other hand, we have determinism from the *n-body* movie. Desired state details are known only when they happen. Any imaginable distribution information

[1] The partially knowing BCA initial state case might be analogous to the true randomness cause. Generally, initial state ignorance comes from practical reasons. Fidelities of real state versus assumed state might be tiny, and misreadings can be easily made. We can see what one-bit difference in 4096-*bit* state can do for BCA evolution (Fig. 4.7). There is a good chance that RO (true) and BCA (pseudo) randomness have the same origin.

is obtained by observation only. It is not evident how this kind of determinism fits in HVT context and how its distributions can be violated by experiment.

For the end, there is a parallel with apparent quantum weirdness in the algorithmic world:

$3x + 1$ *problem story*; We have a state (a natural number). The state will go through the transition, as shown in Fig. 4.1. Before the measurement, the state is in a superposition of a number of possible execution paths. Only after running the algorithm (measurement), the superposition state collapses, and we know the execution path and the transition result.

The $3x + 1$ problem is still unsolved. The conjecture support comes from experimental evidence where all numbers between 1 and $2^{\approx 59}$ are reaching 1 as postulated. Another support comes from a heuristic probability which shows that every multiplication (with 3) has two divisions (with 2) on average, which indicates a starting number shrinkage on the long run [8]. Both of the supports or what we know about the phenomena comes from experiments and probability distributions; even it is a deterministic process.

References

1. Ask a Physicist. Do physicists really believe in true randomness? https://www.askamathematician.com/2009/12/q-do-physicists-really-believe-in-true-randomness/
2. S. Hossenfelder, The 10 most important physics effects, http://backreaction.blogspot.com/2020/02/the-10-most-important-physics-effects.html
3. E.N. Lorenz, The predictability of a flow which possesses many scales of motion. Tellus **21**(3), 289–307 (1969)
4. K.T. Alligood, T.D. Sauer, J.A. Yorke, Chaos. Springer (1996)
5. M. Trenti, P. Hut, N-body simulations (gravitational). Scholarpedia **3**(5), 3930 (2008)
6. The wolfram rule 30 prizes, https://www.rule30prize.org/
7. C. Böhm, G. Jacopini, Flow diagrams, turing machines and languages with only two formation rules. Commun. ACM **9**(5), 366–371 (1966)
8. Collatz conjecture, https://en.wikipedia.org/wiki/Collatz_conjecture
9. R. Vuckovac, On function description (2020), arXiv:2003.05269
10. Conditional branching, https://www.britannica.com/technology/conditional-branching
11. Turing completeness, https://en.wikipedia.org/wiki/Turing_completeness
12. S. Wolfram, A new kind of science (rule 30 alternative description p 27), https://www.wolframscience.com/nks/p27--how-do-simple-programs-behave/?firstview=1/
13. J.C. Lagarias, The 3x + 1 problem: an overview, http://bookstore.ams.org/mbk-78/FreeAttachments/mbk-78-prev.pdf
14. Riemann hypothesis, random walks and möbius function, https://math.stackexchange.com/questions/2350052/riemann-hypothesis-random-walks-and-m
15. R. Vuckovac, Secure and computationally efficient cryptographic primitive based on cellular automaton. Complex Syst. **28**(4), 457–474 (2019)
16. A.M. Turing, Computing machinery and intelligence (1950), in *The Essential Turing: the Ideas That Gave Birth to the Computer Age*, ed. by B.J. Copeland (Oxford University Press, Oxford, 2004), pp. 433–464
17. O. Goldreich, Randomness, interactive proofs, and zero-knowledge–a survey, *The Universal Turing Machine, A Half Century Survey* (Citeseer, 1988)

18. What is the “random oracle model” and why is it controversial? https://crypto.stackexchange.com/questions/879/what-is-the-random-oracle-model-and-why-is-it-controversial
19. What is the random oracle model and why should you care? https://blog.cryptographyengineering.com/2011/09/29/what-is-random-oracle-model-and-why-3/
20. J.S. Bell, On the Einstein Podolsky Rosen paradox. Phys. Phys. Fiz. **1**(3), 195 (1964)