A computational complexity approach to the definition of empirical equivalence.

Doriano Brogioli

January 17, 2019

Abstract

I propose to investigate the problem of empirical equivalence by performing numerical calculations, simulating hypothetical physical systems, with known evolution rules, which include a robot performing an experiment. The aim of the experiments of the robot is to discover the rules governing the system in which it is simulated. The proposed numerical calculation is actually a thought experiment: I discuss the principles of how the discussion on the empirical equivalence should be performed; the discussion is based on the evaluation of the complexity classes of problems connected to the numerical calculation. Based on this discussion, I prove a sufficient condition for empirical equivalence, which is based on the existence of a transformation belonging to a given complexity class.

1 Introduction

Scientists check their theories by comparing the predictions with observations and experimental results. This operation is routinely performed by scientists, but subtleties arise when it is discussed in depth. The philosophical background of this discussion relies on the concept of "data equivalence".

Duhem has pointd out that, for any theory T, a data equivalent rival theory T' can be found. The same author also discussed the difficulties in the comparison between theories and experiments: the experiments always rely on additional hypothesis, making a direct comparison unfeasible [1]. An extremely critical point of view is expressed by the Quine-Duhem thesis, radically questioning the possibilty of comparing an experiment with a single theory or with a small set of theories. Such difficulties lead to the development of the deeper concept of "empirical equivalence" [2, 3, 4, 5].

Data equivalence and empirical equivalence have been discussed in particular in relation with realism [6, 7]; shortly, it is discussed if a realistic point of view can be kept also when there are multiple rival theories which are either data equivalent or emipirically equivalent.

Roughly speaking, the difficulties in comparing experiments to theories arise due to the intricate relations among the theories describing the various aspects of the experiment and the hypothesis on the behavior of the instruments;

the situation becomes even more complex when possible non-deterministic phenomena are taken into account (including experimental errors, quantum non-determinism and non-determinism due to inherently statistical theories such as statistical mechanics and thermodynamics). In this paper, I tackle the problem from an alternative point of view. I hypothetically consider a world that is determined by known rules, and I discuss what its inhabitants would be able to discover about the rules. Although this situation is different from real science (we will start from a privileged situation, in which we already know the "real" theory), the study can give hints about significant real problems.

The approach is inspired by the widespread use of numerical simulations in every field of science, from fundamental physics (e.g. numerical calculations of fundamental particle interactions) to biology (e.g. flock dynamics). The target of the simulation is not necessarily limited to an accurate modelling of a real system, but is sometimes used to get a general feeling of a concept. An example is the theory of the so-called "sand piles" [8]: although the similarity to real piles of sand is limited, they have been used as a prototype for the concept of self-organized criticality, which is common in nature.

The usual application of numerical modelling is to calculate results of a model and compare them with experiments. This can be seen as an "external" point of view. I underline again that the target of the present study is different, i.e. inspecting the "internal" point of view: try to understand what the inhabitants of a hypothetical world could understand about the physical laws, by simulating not only the phenomena but also the experimental process itself. Despite the widespread use of numerical simulations, to my knowledge this approach has never been attempted in the field of philosophy of science, at least in the form that I am proposing here.

The idea of simulating a world, and studying the limits to the knowledge of its inhabitants, is apparently unfeasible in practice: the simulation of a whole experimental setup, and, in principle, of the scientist performing the experiment, including his brain, is beyond the capacity of computers. I will limit the discussion to very simplified worlds (described as cellular automata) and to very simplified entities performing experiments (robots rather than humans). The very limited target is only a first attempt in this direction, which can be expanded in the future.

Moreover, I will not present results of numerical experiments, but rather I will focus on the definition of the problem and I will derive generally valid sentences (theorems), based on computational complexity theory [9]. It has been already noticed that this theory provides a deep characterisation of physical concepts [10]. In other words, I will present "thought experiments" and discuss their mathematical constraints. This way of proceeding has been already proved fruitfully, for example in the case of the theorems obtained on the Turing machines: they were actually "thought experiments" discussed in advance with the real development of computers; they gave interesting results independent on the practical realisation of the machines.

As an example of the power of the approach, I will mathematically prove a sufficient condition for empirical equivalence, which is valid at least in the

described framework. The sufficient condition is connected to the existence of a transformation between two models, belonging to a given computational complexity class.

2 Statement of the problem and definitions

2.1 Cellular automata as models.

I will consider mathematical models expressed as cellular automata. Such a model is based on a set of cell, distributed on a lattice that is either 1-, 2- or 3-dimensional (or, possibly, with higher dimensionality). Each cell, at a given time, has a state, expressed by a number. The states evolve with time, according to an "evolution rule": given the lattice at time-step t, the evolution at time-step $t+1$ of each cell is calculated based on a given neighborehood of the cell itself and the evolution rule. A famous example is the "game of life" [11].

Cellular automata are often used to model natural phenomena: the evolution rule is chosen so that it mimicks some phenomenon; the evolution of a given configuration of the cellular automaton is calculated and results are compared with the experiments. In the case considered in the present paper, the target is completely different: I aim at describing inside the cellular automaton the process of performing an experiment, and that experiment will be aimed at discovering some property of the evolution rule.

In order to reach the target, the model (the cellular automaton) must be powerful enough to enable the description of the experimental apparatus and of its control logic, at least. The cellular automata are actually quite powerful. Indeed it is known that many of them are Turing-complete; roughly speaking, it is possible to give a suitable initial lattice which evolves similarly to a Turing machine (or any real computer) [12]. Examples of Turing-complete cellular automata are the "game of life" [13] and "Rule 110" [14]. In specific cellular automata, it was possible to simulate a self-replicating robot [15]. We thus see that the cellular automata are able to model two of the features connected with a complex bahaviour, i.e. the ability to perform operations following a logic and self-replicating.

2.2 Notation for the representation of cellular automata

The state of each cell can be represented by a finite sequence of bits, usually a few bits, e.g. for the "game of life" one bit suffices (0 represents an empty cell and 1 an occupied cell). The instantaneous state of the whole cell lattice is coded by concatenating the bits representing each cell, taken a given ordering. I call s a coding of the lattice state, with $s \in [0,1]^*$, i.e. s is a string of 0 and 1 (see Ref. [9] for the notation); s_j is the jth bit of the representation. I call $|s|$ the length of the string s, i.e. the number of bits representing a given lattice.

The evolution rule is a function $f : \{0,1\}^* \to \{0,1\}^*$ which maps a lattice state in the lattice state at a subsequent (discrete) time. Since the lattice is

fixed during the evolution, $|f(s)| = |s|$, i.e. a lattice state s is mapped to a lattice state with the same number of bits.

I represent the (discrete) time-sequence of lattice states (or "history") with h and the state s at a discrete time t will be written as $h(t)$. The range of t is between 0 and $T-1$, i.e. it is finite. Moreover I assume that the number of time-steps, T, is equal to the number of bits of the lattice state s, $N = |s|$. This means that the state-sequences h are square matrices of binary digits. This limitation will not affect the discussion performed in this paper.

The time-sequences h are formally elements of a set $\mathcal{H}$, $h \in \mathcal{H}$, where the set $\mathcal{H}$ is defined as:

$$\mathcal{H} = \bigcup_{N \in \mathbb{N}} [0 \dots N-1] \times [0,1]^N \tag{1}$$

where $[0 \dots N-1] \subset \mathbb{N}$ is the finite interval of integers from 0 to $N-1$ representing the time range. All the states of a state-sequence have the same length N, i.e. $|h(t)| = N$.

In terms of h, the evolution rule is written as:

$$h(t+1) = f\left[h(t)\right] \tag{2}$$

The "initial state" of the cellular automaton is the state $h(t=0)$, which I will call $\tilde{s}$. Given the initial state $\tilde{s}$, its evolution up to time $T-1$ can be calculated by sequentially applying f, thus obtaining a full time-sequence h. I will call $F : [0,1]^* \to \mathcal{H}$ the function which maps an initial state $\tilde{s}$ into the full time-sequence $h = F(\tilde{s})$. More formally, F is defined so that:

$$\left[F(\tilde{s})\right](t=0) = \tilde{s} \tag{3}$$

$$\left[F(\tilde{s})\right](t+1) = f\left\{\left[F(\tilde{s})\right](t)\right\} \tag{4}$$

The two equations univocally define the function F.

2.3 Inspecting the "internal point of view"

Giving an initial state of the cellular automaton $\tilde{s}$ and observing its time-evolution of course enables to infere the evolution rule. However, what we would like to inspect is the "internal point of view": which details of the evolution rule can be discovered by an entity which is simulated inside the cellular automaton itself.

In order to clarify this point, I make an example of a detail of the evolution rule that cannot be discovered from the internal point of view.

Example 1. Given a cellular automaton α and evolution rule f_α, it is possible to obtain an alternative representation β of the same cellular automaton by flipping (negating) all the bits. The evolution rule f_β is $f_\beta(s) = \neg f(\neg s)$, where the operator $\neg$ flips (negates) all the bits of its argument. By giving an initial state $\tilde{s}$ and observing the resulting time-sequence $F(\tilde{s})$, a researcher can easily find if the rule of the cellular automaton is f_α or f_β; this represents the "external point of view". However, it is clear that f and f' represent the same cellular

automaton, simply obtained by representing each bit either as it is or flipped: the difference is in the representation and not in the cellular automaton itself. We thus expect that an experiment, simulated inside the cellular automaton, cannot distinguish between f_α and f_β.

I argue that the two evolution rules f_α and f_β cannot be distinguished from the "internal point of view", while it is possible that other couples of evolution rules can. The main target of this paper is to formally define the internal point of view and characterize it mathematically.

2.4 A different perspective on empirical equivalence

Usually, the numerical simulation is used to calculate results of a model; the results expected from the model are then compared with the results of real experiments. The researcher has an external point of view, i.e. it sees from outside both the experimental system and the numerical simulation. If two simulations of models α and β give the same experimentally testable results, then the models are judged empirically equivalent.

The present approach is radically different. I imagine that the two models α and β are simulated numerically, including the experimental apparatus, and I wonder if it is possible to perform an experiment which gives different results in α and β. If not, the two cellular automata are deemed empirically equivalent. This represents the new perspective on empirical equivalence (an internal point of view) that I propose in this paper.

2.5 Robots performing experiments

The approach for inspecting the "internal point of view" consists in implementing a robot, inside the cellular automaton, that performs experiments. It is already known that a self-replicating robot can be built in specific cellular automata, and in general an "universal constructor" [15]: so it should be possible to discuss this possibility at least for some cellular automata. Practically, the implementation of the robot is performed by designing an initial state $\tilde{s}$, which evolves mimicking a robot performing a specific task.

In particular, the task will be the execution of a sequence of experiments. It is important to emphasize here that I do not expect that the robot itself performs a heuristic process to discover the rule f: this would require an investigation of artificial intelligence. Instead, the robot will make a set of experiments aimed at distinguishing between two known possible cellular automata α and β. The operation is pre-programmed and the logics behind it is decided by the human operator who is studying the cellular automata.

An ingenuous way of inspecting the behaviour of the robot could be the following. The reseacher Φ, which is investigating the cellular automaton, gives an initial state $\tilde{s} = h(t = 0)$ describing the initial state of the robot, then Φ calculates the time-sequence $h(t)$ and from it Φ infers the governing rule f. Of course, by this approach it is not clearly evident wether Φ is inspecting the

"internal" or the "external" point of view; for example, the researcher Φ could easily distinguish f_α from f_β of Example 1, having access to the time-sequence $h(t)$, and pretend that the distinction comes from the experiment itself. In order to avoid this difficulty, it is necessary to better distinguish the information available to the robot itself (internal point of view) from the information on the representation of the cellular automaton (external point of view). This distinction can be done by means of the following assumption.

I will assume that the robot is "programmable". I will give a formal definition below; in less formal way, I say that the robot accepts a program, written in a given language (similar to C or Pascal), with some additional built-in function for controlling the actuators and sensors. An example of such a function is the movement of an arm; in this case, the arguments of the function contain the parameters of the movement (e.g. speed and final position) and the results include the forces measured by the haptic sensors.

I suggest that the genuine "internal point of view" is represented by the information on the governing rule f that the researcher can get by programming the robot and observing the result of the program (a number, a string, or, in the following, even a single bit) without having access to the whole state-sequence h.

A formal definition of the operation of implementing the robot in the cellular automaton, and of the related difficulties, is reported in Sect. 3.3.

3 Implementing Turing machines and robots in cellular automata

3.1 Notations about Turing machines and universal Turing machines

I assume that the general notions about Turing machines are known to the reader. I will use the notation of Ref. [9].

When a Turing machine M operates on an input i it returns a single bit of output at the end of the program (either "accept" or "reject"). This output bit is denoted $M(i)$.

An universal Turing machine U is a Turing machine able to simulate any other Turing machine (it is the analogous of a computer, which can be programmed to execute any task). Given a Turing machine M and an input i, a universal Turing machine U takes as inputs a coding of M (as a string) and the input i, and returns as output $M(i)$.

3.2 Implementing Turing machines in cellular automata

In literature, it has been often discusses if (and how) it is possible to implement an arbitrary Turing machine in a given cellular automaton. As already discussed, this is possible e.g. for the "game of life" [13] and for "Rule 110" [14].

This possibility is usually expressed in terms of "Turing-universality" of the cellular automaton: if it is possible to implement any arbitrary Turing machine in it, then the automaton itself behaves as an universal Turing machine.

The concept of "universal machine" was developed by Turing, who showed the existence of a machine (nowadays called "universal Turing machine") that can be programmed in order to behave as any other desired Turing machine [16]. In that case, it was clear from the definition that the "universal machine" was able to simulate the behaviour of any specific Turing machine. When we want to discuss the Turing-universality of a given cellular automaton, however, subtleties arise [17].

For showing that a given cellular automaton is Turing-universal, a coding $C(M, i)$ and a decoding $D(s)$ algoritm are provided, operating as follows. Given a Turing machine M (or better its coding as a string) and an input i, the coding algorithm generates an initial state, $\tilde{s} = C(M, i)$. The evolution rule is repeatedly applied, $s' = f(s)$, starting from the initial state $\tilde{s}$. At each repetition, the deconding algorithm checks if the state s represents an halting state; if so, the decoding algorithm returns the halting state, "accept" or "reject". It is possible that the halting state is never reached; but if it is, then the decoding algorithm gives the result of the Turing machine, $D(s) = M(i)$.

The two algorithms C and D can be imagined as roughly analogus to a compiler and a debugger.

A related definition of "circuit-universality" has been given [18]. It refers to the ability of a cellular automaton to calculate the output of a circuit c, given the input values i. Also in this case there is a coding operation $C(c, i)$, which codes the circuit c and the input i into an initial state of the cellular automaton, and a decoding algorithm $D(h)$ which retrieves the output. In the definition [18], the decoding algorithm is specified: it trivially takes the value of a given cell. The overall operation is thus:

$$D\left\{F\left[C(c, i)\right]\right\} = c(i) \tag{5}$$

This can be qualitatively explained as:

- the circuit c and the input i are coded into $\tilde{s} = C(c, i)$;
- the state-sequence is calculated, $F\left[C(c, i)\right]$;
- the state-sequence is decoded, $D\left\{F\left[C(c, i)\right]\right\}$

The result corresponds to $M(i)$.

The two definitions are however not complete. Indeed, without a proper definition, it would be possible to conceal the calculation of $M(i)$ or $c(i)$ inside the operations of coding or decoding, thus deceiving into believing that some very simple cellular automaton (likely not able to perform any calculation) is Turing-universal or circuit-universal. Since this point is important, I make an example.

Example 2. A trivial cellular automaton is defined by the rule $f(s) = s$. It is likely not circuit-universal. The coding algorithm $C(c, i)$ calculates the result

of the output of a circuit c on the input i, i.e. it is defined as a single bit $C(c,i) = c(i)$. This bit is encoded as the first bit of the initial state $\tilde{s}$ (having a given, fixed, length): $\tilde{s}_{j=0} = c(i)$. The decoding algorithm returns the same bit: $D(s) = s_{j=0}$. In turn, $D(s) = c(i)$: the coding and decoding algorithms return the result of the calculation of the output of the circuit c on the input i, mimicking a circuit-universality.

In this example, the calculation is clearly not performed by the cellular automaton, but it is concealed into the coding algorithm. It is also possible to conceal the calculation in the decoding algorithm, as in the following example.

Example 3. As in the previous example, the cellular automaton is defined by the rule $f(s) = s$. The coding algorithm $C(c,i)$ returns the binary representation of c and i: $\tilde{s} = (c,i)$. The decoding algorithm first converts s into (c,i), then calculates $c(i)$ and returns it: $D(s) = c(i)$.

From the two examples, it is clear than any cellular automaton could be declared as Turing-universal or circuit-universal, unless additional conditions are imposed on the nature of the coding and decoding algorithms. No syntactic property has been found useful for improving the definition; likely, a property such "the algorithm calculates the result of $M(i)$" cannot be defined at syntactic level. The approach is thus different: the additional condition is defined based on the computational complexity of the coding and decoding algorithms, compared to the calculation of $M(i)$ (respectively, $c(i)$). It is required that the complexity of C and D is less than the complexity of calculating $M(i)$ (respectively, $c(i)$), so that C and D cannot conceal the operation of calculating $M(i)$ (respectively, $c(i)$).

- In Turing-universality, the coding and decoding algorithms are required to be (total) computable functions [19, 20] (roughly, they can be computed in a given finite time for every input). The implemented Turing machine instead calculates partial functions (roughly, functions for which it is not guaranteed that the calculation halts)

- In circuit-universality [18], the coding and decoding algorithms are required to be in the complexity class **NC** (for the definition of the complexity classes see Ref. [9]). Instead, the calculation of the output $c(i)$ of the implemented circuit is a **P-complete** problem. It is conjectured that **P-complete** problems are more complex than **NC**, thus C and D cannot conceal the operation of calculating $c(i)$.

These requirements prevent to conceal the calculation performed by the implemented Turing machine inside the coding and decoding operations: we can say that the calculation of the output of the Turing machine or of the circuit is genuinely performed by the cellular automaton.

We can describe the discussion as a dispute on the circuit universality of a given cellular automaton α, with evolution F_α. Let us imagine that a researcher Φ thinks that α is circuit-universal, while another researcher Ψ is not convinced.

At first, Φ provides the coding C and decoding D algorithms. The researcher Ψ checks that C and D actually work as expected, i.e. they encode any circuit and any input i in the cellular automaton according to Eq. 5.

If Ψ is not jet convinced, Ψ can ask for a proof that the operation of calculating $M(i)$ is not concealed into C or D. This proof can be given by Φ, as described above, based on the computational complexity of the algorithms: if Φ is able to show that the calculation of $M(i)$ asymptotically requires more resources (time or space) than C and D, this should be considered as a valid proof. Of course, the proof could be based on conjectures on the separation of complexity classes (such as the conjecture that **NP** contains problems that are more complex than **P**); in such cases, the result on the circuit-universality of the given automaton is not conclusive but only related to a conjecture.

3.3 Formal definition of the robot

The programmability of the robot is expressed by saying that it includes an universal Turing machine U. In order to allow us to interact with the robot, U will take as input a robot program P and i, the robot program P representing an extended coding of a Turing machine. Here "extended" means that it enables to associate states (of the Turing machine) to a given set of operations of the robot. Entering such states during the execution of the program P will force the robot to perform an operation; parameters of the operation are possibly read from a special tape and results are written on another special tape. These states correspond to the "functions" described above in the informal description. It is worth noting that the robot operations are fixed, once the robot itself and the controlling universal Turing machine U are defined; the program P can only define if and when to execute them.

Now it is necessary to describe the meaning of "implementing" the robot in the cellular automaton. I follow the analogous concept of "implementing a Turing machine" or "implementing a circuit" described in Sect. 3.2 (see Refs [16, 17, 18]. Thus this operation is performed by giving a couple of algorithms: the coding C and decoding D algorithms. The coding algorithm maps the robot program P, the time-steps required for the execution T (expressed in unary) and the input i into the initial state of the cellular automaton, $\tilde{s} = C(P, T, i)$. The decoding algorithm reads the resulting state-sequence $h = F\left[C(P, T, i)\right]$ and evaluates the state of the Turing machine, "accept" or "reject", at the time-step T; the result is called r:

$$r = D\left\{F\left[C(P, T, i)\right]\right\} \tag{6}$$

If the robot program P is not yet ended at time T, on input i, then the result can be arbitrarily set to "reject". It is worth noting that, at variance with the case of Eq. 5, r cannot be related only to the robot program P, the time T and the input i: it contains information on the cellular automaton in which it is implemented, through the evolution function F.

After defining the coding C and decoding D algorithms, Φ starts interacting with the system, now seen as a black box able to execute programs. First, Φ can

check if the implementation is working propertly, i.e. if it correctly responds as the desired Turing machine when no robot operation is involved. Then, Φ can start operating the robot and perform experiments. If the robot was properly designed, Φ will be eventually able to perform experiments distinguishing α from β.

4 How to properly dispute about the empirical equivalence of two cellular automata

4.1 Dispute on the empirical equivalence of two cellular automata

Let us consider two cellular automata α and β. The researcher Φ believes that an entity (the above-described programmable robot), built inside one of the two, is able to decide if it is α or β, while Ψ believes the opposite, i.e. that the two cellular automata are empirical equivalent. It is worth noticing that the data equivalence here refers to the "internal point of view" as expressed in Sect. 2.4.

In order to convince Ψ, Φ proposes the coding and decoding algorithms C_α and D_α which apply to the cellular automaton α; moreover, Φ provides a robot program P which performs suitable experiments; the experiments (according to the belief of Φ) enable to prove that the correct model is α. The experiments are conducted considering $r = D\left\{F\left[C(P,T,i)\right]\right\}$ as a black box: the program P, the time T and the input i are given and the result r is inspected; Φ argues that, from this black-box view, it is possible to deduce that the cellular automaton is α rather then β.

The target of the present paper is to discuss how to rigorously decide if the procedure used by a researche Φ to compare his experiments with theories is valid, thus I proceed by considering that the approach declared by Φ is not convincing for Ψ, who requires a more solid proof: Ψ does not believe that the experiments proposed by Φ are able to distinguish among the cellular automata α and β.

In order to support the position, Ψ provides the coding and decoding algorithms C_β and D_β, for the cellular automaton β, aiming to show that the same results would be obtained in β.

If Ψ is not able to provide coding and decoding algorithms C_β and D_β, then the dispute is resolved in favour of Φ; likely, the cellular automaton β is extremely different from α, since it is not possible to implement a robot in it, while in α it is. In the following, I assume that Ψ is able to provide the algorithms.

The aim of Ψ is now to show that, for any program M, the execution gives the same result in α and β:

$$D_\alpha\left\{F_\alpha\left[C_\alpha\left(P,T,i\right)\right]\right\} = D_\beta\left\{F_\beta\left[C_\beta\left(P,T,i\right)\right]\right\} \tag{7}$$

The validity of Eq. 7 can be checked by means of logical arguments, eventually leading to an agreement on them among Φ and Ψ. If Eq. 7 holds, it is a point

in favour of Ψ (the two cellular automata α and β are empirically equivalent); instead, if there is a program M for which Eq. 7 is false, it is a point in favour of Φ (the two cellular automata α and β are not empirically equivalent).

It is worth noting that, in general, it is not possible to require that the algorithms C and D are the same for the two cellular automata, because the implementation of the Turing machine in the different cellular automata is likely different; this is why we assume that there are two sets, C_α, D_α and C_β, D_β. Instead, the program P that is run is the same, in both α and β.

At this point of the dispute, Φ could argue that Ψ could have cheated, by using a trick analogous to what reported in Sect. 3.2. The trick would consist in concealing the operation $D_\alpha \{F_\alpha [C_\alpha (P, T, i)]\}$ in either C_β or D_β. This can be done as in the following example:

Example 4. The coding algorithm $C_\beta(P, T, i)$ first calculates r (a single bit):

$$r = D_\alpha \{F_\alpha [C_\alpha (P, T, i)]\} \tag{8}$$

Then C encodes r into the first bit of the initial state:

$$\tilde{s}_{j=0} = r \tag{9}$$

The decoding function returns the first bit of the initial state:

$$D_\beta (h) = h_{j=0}(t = 0) = \tilde{s}_{j=0} = r \tag{10}$$

We thus find that $D_\beta \{F_\beta [C_\beta (P, T, i)]\}$ is equal to r, so Eq. 7 holds. It is here clear that the calculation of $D_\alpha \{F_\alpha [C_\alpha (P, T, i)]\}$ is concealed into C_β.

A similar result, obtained by concealing the operation $D_\alpha \{F_\alpha [C_\alpha (M)]\}$ into D_β, can be obtained in analogy with the example reported in Sect. 3.2.

4.2 Discussion based on the computational complexity

In analogy with the case of Sect. 3.2, also in this case the problem expressed at the end of the previous subsection can be resolved by imposing conditions on the computational complexity of the coding and decoding operations. The problem is that it is possible that Ψ conceals the operation $D_\alpha \{F_\alpha [C_\alpha (P, T, i)]\}$ in C_β or in D_β. In order to avoid this, in analogy with the case of Sect. 3.2, the computational complexity of the operations C_β and D_β is required to be in a given class; then it is checked that $D_\alpha \{F_\alpha [C_\alpha (P, T, i, i)]\}$ can execute algorithms with a larger complexity.

It is possible to impose that the coding and decoding algorithms operate in logarithmic space in the length of the representation of P, T and i. Indeed, the robot contains an universal Turing machine U, which is implemented into the cellular automaton; C simply stores P, T and i as input data (this approach is similar to embedding an "interpreter" in the cellular automaton together with a representation of the program and of its input). This requires to only keep a finite number of indices (memory addresses), with a length that is logarithmic

in the length of the representation of P, T and i (I remind here that T is represented in unary) [9]. This can be expressed saying that the calculation of a bit of the output of C_β or the output of D_β are problems in **L**, i.e. calculations that can be performed using an amount of memory that is logarithmic in the size of the input.

It is usually conjectured [9] that **L** is strictly contained in **P** (although, up to now, there are no rigorous proofs that **L** is not equal to **P**, or even **NP**). I will assume this conjecture in the following.

If the robot program P does not make use of functions of the robot, it simply describes a Turing machine M. Thus $D_\alpha \{F_\alpha [C_\alpha (P, T, i)]\}$ calculates $M(i)$ at time T. The problem of taking M and i as inputs, and calculating $M(i)$ at time T, is a well-known **P**-complete problem.

Thus it has been shown that the condition that C_β and D_β work in log-space prevents them to contain the calculation of $D_\alpha \{F_\alpha [C_\alpha (M, i)]\}$.

In particular, in order to prove that Ψ has cheated by doing so, Φ can operate as follows. Φ writes a program P that takes an input split into two parts, M and i (the first part is the representation of a Turing machine). First, P checks if it is running in α or β. In the first case, P executes the machine M on the input i; this operation calculates the result of a **P**-complete problem. In the second case, it rejects. If the calculation of $D_\alpha \{F_\alpha [C_\alpha (P, T, M, i)]\}$ is concealed in C_β or D_β, then the two algorithms should require more than log-space to run.

5 Formalization of the dispute

I formalize the procedure for resolving the dispute between Φ and Ψ about the empirical equivalence of α and β. Φ believes that there is an experiment able to distinguish α from β; instead Ψ thinks that they are empirically equivalent.

The first step is to declare the two cellular automata. Φ chooses one of the cellular automata; I assume that it is α. Φ provides the coding and decoding algorithms C_α and D_α. Ψ provides the coding and decoding algorithms C_β and D_β.

The inability of either Φ or Ψ to do so is regarded as the inability of implementing a robot or a Turing machine in the corresponding cellular automaton. If Φ succeeds but Ψ fails, then it is possible to conclude that the two cellular automata are empirically different, since one of them is Turing-complete and the other is not; if both fail, then it is concluded that the two cellular automata are not complex enough to host a Turing machine, thus the problem of their empirical equivalence from the internal point of view is not relevant.

The procedure for a fair dispute is described in the following as a game between two players. The empirical equivalence of α and β is concluded if Ψ has a winning strategy; the opposite is concluded if Φ has a winning strategy. It must be noticed that this procedure refers to the given C_α, D_α, C_β and D_β.

The first step consists in checking that C_β and D_β can be calculated in log-space. If not, the dispute is resolved in favour of Φ because Ψ failed in providing a valid couple of algorithms C_β and D_β.

The empirical equivalence is expressed as the Eq. 7. It corresponds to making an experiment, represented by the couple P, i, in α and β and comparing the results.

In order to support the opinion that α and β are not empirically equivalent, Φ looks for a couple P, i for which the equation is not satisfied. If Φ can provide such a couple P, i, then the dispute is resolved in favour of Φ. Else, it is assumed that the equation holds and the dispute is resolved in favour of Ψ.

6 Transformations

Transformations play an important role in physical theories. The same theory can be represented in different coordinate systems, resulting in different mathematical models; a change of coordinates is a transformation, connecting the different mathematical models. It is often implicitly assumed that a change of coordinates transforms a model into an empirically equivalent one. However, the existence of a (more general) transformation between two models in general does not ensure that they are empirically equivalent. This has been discussed in particular for quantum theories: unitary transformations do not ensure the empirical equivalence but isometric transformations do [21]. I give examples relevant for this discussion in Sect. 6.2.

6.1 Definition of transformation

A transformation T is a function which transforms time-sequences of a cellular automaton α into time-sequences of another cellular automaton β, and *vice-versa*.

A transformation T is defined as follows.

Definition 1 (Transformation)**.** A function $T : [0,1]^* \to [0,1]^*$ is a transformation between the cellular automata α and β if T is invertible (i.e. one-to-one and onto) and

$$T\left[f_\alpha\left(s\right)\right] = f_\beta\left[T\left(s\right)\right] \tag{11}$$

It must be noticed that here the symbol T^{-1} refers to the inverse function of T, so that $T\left[T^{-1}(s)\right] = s$, rather than the algebraic inverse $1/T$.

Given two cellular automata α and β, and two state-sequences h_α and h_β belonging to them, they are connected by the transformation of the lattice states at each time:

$$h_\beta(t) = T\left[h_\alpha(t)\right] \tag{12}$$

I also define "partial transformations", as follows.

Definition 2 (Partial transformation)**.** Two sets $\mathcal{S}_\alpha \subset [0,1]^*$ and $\mathcal{S}_\beta \subset [0,1]^*$ are given. A function $T : \mathcal{S}_\alpha \to \mathcal{S}_\beta$ is a partial transformation between the cellular automata α and β if T is invertible (i.e. one-to-one and onto) and

$$T\left[f_\alpha\left(s\right)\right] = f_\beta\left[T\left(s\right)\right] \tag{13}$$

$\forall\ s \in \mathcal{S}_\alpha$.

Partial transformations are particularly relevant when:

$$\{F_\alpha\left[C_\alpha(P,T,i)\right](t) \forall P \in [0,1]^*, T \in \mathbb{N}, i \in [0,1]^*, t \in [0 \dots N-1]\} \subset \mathcal{S}_\alpha \quad (14)$$

i.e. when the domain of T covers at least the initial states given by the coding algorithm and their evolutions in the time-sequences. Since these are the lattice states that are subject to the analysis and to the experiments performed by the robot, they are the only ones that are relevant for the discussion. For this reason, I will call such transformations "partial", rather than "local", since they cover enough space and time to describe the whole world that is seen by the robot performing the experiments.

6.2 Examples of transformations

Here I give two exemples of transformations. The first is inspired by the Example 1.

Example 5. A cellular automaton α is given, with evolution rule f_α. A derived cellular automaton β is defined by an the evolution rule f_β:

$$f_\beta(s) = \neg f_\alpha(\neg s) \quad (15)$$

The transformation T connecting them is thus:

$$T(s) = \neg s \quad (16)$$

T is a transformation between α and β according to Def. 1.

Proof. T it is clearly invertible and $T^{-1} = T$; moreover, it satisfies Eq. 11, by substituting $s' = \neg s$:

$$\neg f_\alpha(s') = f_\beta(\neg s') \quad (17)$$

□

As already noticed for Example 1, the two cellular automata α and β are likely empirically equivalent (from the internal point of view). As already noticed, in general, the existence of a transformation between two cellular automata does not ensures that they are empirically equivalent. The following is an example of transformation between two cellular automata that are lilely not empirically equivalent.

Example 6. Two reversible cellular automata α and β are given. Reversibility means that the evolution rules f_α and f_β are invertible (one-to-one and onto). I define a partial transformation on a domain $\mathcal{S}_\alpha$ and co-domain $\mathcal{S}_\beta$.

I define a function $\tau_\alpha(s)$, with the following property:

$$\tau_\alpha[f_\alpha(s)] = \tau_\alpha[s] + 1 \quad (18)$$

Roughly, the function τ_α represents a clock, associating an absolute time to a state s (number of time steps elapsed since a given initial time) in the cellular

automaton α. The existence of such a function depends on the cellular automata α and β and of the sets $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$; a discussion of this point would require to give a specific example with specific α and β and is outside the scope of the present paper.

The transformation T is defined as follows:

$$T(s) = f_\beta^{\tau_\alpha(s)} \left[f_\alpha^{-\tau_\alpha(s)}(s) \right] \tag{19}$$

where $f^n(s)$ is the result of composing n times the function f (not the algebraic power).

In words, T first evaluates the absolute time of s in the cellular automaton α; evolves s back in time to the origin of the absolute time; finally evolves forward in time for the same number of steps.

It must be discussed if T is a transformation between α and β according to Def. 1. T can be constructed as surjective (onto) by definition, taking $\mathcal{S}_\beta$ as the co-domain of T, $\mathcal{S}_\beta = T[\mathcal{S}_\beta]$. The injectivity depends on the presence and structure of cycles; it can be ensured by giving suitable sets $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$; however, also in this case, a discussion of this point would require to give a specific example with specific α and β and is outside the scope of the present paper. The last property required by Def. 1 is to show that Eq. 13 holds, as follows.

Proof. I start calculating $T[f_\alpha(s)]$ by using the definition of T, Eq. 19:

$$T[f_\alpha(s)] = f_\beta^{\tau_\alpha[f_\alpha(s)]} \left\{ f_\alpha^{-\tau_\alpha[f_\alpha(s)]} [f_\alpha(s)] \right\} \tag{20}$$

By using Eq. 18:

$$T[f_\alpha(s)] = f_\beta^{\tau_\alpha(s)+1} \left\{ f_\alpha^{-\tau_\alpha(s)-1} [f_\alpha(s)] \right\} \tag{21}$$

Calculating the composition of the function f_α:

$$T[f_\alpha(s)] = f_\beta^{\tau_\alpha(s)+1} \left[f_\alpha^{-\tau_\alpha(s)}(s) \right] \tag{22}$$

This equation can be rewritten as:

$$T[f_\alpha(s)] = f_\beta \left\{ f_\beta^{\tau_\alpha(s)} \left[f_\alpha^{-\tau_\alpha(s)+1}(s) \right] \right\} \tag{23}$$

In this expression it is possible to recognise the definition of T, so that Eq. 13 is obtained. □

The trasformation of this last example can be built between quite generic couples of cellular automata; assuming that it is possible to find two reversible cellular automata that are not empirically equivalent, then it is possible to build the corresponding T also for them. This emphasizes that the existence of a transformation between two cellular automata does not prove their empirical equivalente (from the internal point of view). In the next section, I will show that there is a sufficient condition which ensures that, in the presence of a transformation, the two cellular automata are empirically equivalent (in the sense defined in Sect. 4).

6.3 Transformations ensuring empirical equivalence

In this section I show that it is possible to select a subset of the set of trasformation (as defined in the previous section), by giving an additional condition, so that the presence of such a transformation between two cellular automata ensures that they are empirically equivalent.

Definition 3 (Log-space transformation)**.** A transformation T is a log-space transformation if each bit of $T(s)$ and of $T^{-1}(s)$ can be calculated using an amount of memory that is logarithmic in the number of bits of s.

The following theorem can be proved.

Theorem 1. *The presence of a log-space transformation T between two cellular automata α and β is a sufficient condition for their empirical equivalence (as defined in Sect. 4).*

Proof. The two cellular automata α and β are given, together with the coding C_α and decoding D_α algorithms for the cellular automaton α.

The coding algorithm C_β for the cellular automaton β is defined as follows:

$$C_\beta(P,T,i) = T\left[C_\alpha(P,T,i)\right] \tag{24}$$

In words: the algorithm C_β first uses the coding algorithm C_α for coding P,T,i into an initial state $\tilde{s}$ of α; then it transforms it into an initial state of β.

The decoding algorithm D_β if *formally* defined as:

$$D_\beta(h) = D_\alpha\left[T^{-1}(h)\right] \tag{25}$$

i.e., given a state-sequence h of β, the inverse of T is applied to obtain the corresponding state-sequence of α.

The algorithms C_α, D_α and T operate in log-space, thus also C_β and D_β can operate in log-space, as requested in Sect. 5.

Finally, Eq. 7 is evaluated. By using the definitions of C_β and D_β given above:

$$D_\beta\left\{F_\beta\left[C_\beta\left(P,T,i\right)\right]\right\} = D_\alpha\left[T^{-1}\left(F_\beta\left\{T\left[C_\alpha\left(P,T,i\right)\right]\right\}\right)\right] \tag{26}$$

By using Eq. 11, $T^{-1}\left\{F_\beta\left[T(\tilde{s})\right]\right\} = \tilde{s}$, and thus Eq. 7 is obtained.

□

It can be noticed that the transformation T of Example 5 can be clearly calculated in log-space, and indeed it was already argued that it connected empirically equivalent cellular automata.

In the case of the transformation T of the Example 6, it was argued that it could connect cellular automata that are not empirically equivalent. Actually, the definition of T (Eq. 18) contains the calculation of $f^n(\tilde{s})$: this operation is equivalent to calculating the evolute of an initial state of a cellular automaton, that is a **P-complete** problem in the case of circuit-complete cellular automata. Under the (widely believed) conjecture that $\mathbf{L}\neq\mathbf{P}$, it is thus not possible to calculate T in log-space.

7 Conclusion

I introduced the idea that the empirical equivalence can be studied from an "internal point of view" by numerically simulating a system, including a robot which performs experiments. I showed that a discussion about the empirical equivalence of two models, from this "internal point of view", requires some subtleties and I suggested how to correctly perform it. The distinction between empirically equivalent and non-equivalent models is based on the theory of computational complexity.

The presence of a transformation between two cellular automata is not enough to ensure that they are empirically equivalent; I discuss two examples of transformations, one connecting empirically equivalent cellular automata and one connecting cellular automata that are (likely) not empirically equivalent.

However, from the suggested procedure for discussing the empirical equivalence, I obtained that an additional condition that can be imposed on a transformation; if such a condition is met, then the presence of the transformation between two cellular automata is a sufficient condition for empirical equivalence. This condition is defined in terms of computational complexity: it is required that the transformation can be calculated using an amount of memory (space) that is at most logarithmic in the input length. If such a log-space transformation exists between two models, then the models are empirically equivalent.

The above-mentioned examples of transformations (one connecting empirically equivalent models and one connecting models that can be not empirically equivalent) are actually different from the point of view of computational complexity, i.e. one can be calculated in logarithmic space and the other does not, emphasizing the importance of the complexity class of the transformation for determining the empirical equivalence.

Although the discussion only covers systems that are discrete (in space, time and states of cell), various properties of complexity classes have been extended to continuous systems. Wether this is possible in the present case will be the goal of future work.

References

[1] P. Duhem. *The Aim and Structure of Physical Theory*. Princeton University Press, Princeton NJ, 1954.

[2] C. Hoefer and A. Rosenberg. Empirical equivalence, underdetermination, and systems of the world. *Philosophy of Science*, 61:592–607, 1994.

[3] L. Laudan and J. Leplin. Empirical equivalence and underdetermination. *The Journal of Philosophy*, 88:449–472, 1991.

[4] P. Acuña and D. Dieks. Another look at empirical equivalence and underdetermination of theory choice. *European Journal for Philosophy of Science*, 4(2):153–180, 2013.

[5] P. Acuña and D. Dieks. *Artificial examples of empirical equivalence*. Springer, 2014.

[6] R. Boyd. Realism, underdetermination, and a causal theory of evidence. *Noûs*, 7:1–12, 1973.

[7] J. Worrall. Underdetermination, realism and empirical equivalence, 2011.

[8] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: an explanation of $1/f$ noise. *Physical Review Letters*, 59(4), 1987.

[9] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, 2009.

[10] S. Aaronson. Why philosophers should care about computational complexity, 2011. manuscript Pre-print.

[11] Martin Gardner. The fantastic combinations of John Conway's new solitaire game of life. *Scientific American*, 223:120–123, 1970.

[12] S. Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1–2):1–35, 1984.

[13] B. Durand and Zs. Róka. The game of life: universality revisited, 1999. Cellular automata, (Saissac, 1996) (M. Delorme and J. Mazoyer, eds.), Kluwer Acad. Publ., Dordrecht.

[14] M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40.

[15] J. von Neumann and A. W. Burks. *Theory of self-reproducing automata*. University of Illinois Press, Urbana and London, 1966.

[16] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc. Ser.*, 2(42):230–265, 1937.

[17] Nicolas Ollinger. Universalities in cellular automata; a (short) survey, 2008. Bruno Durand. JAC 2008, Uzès, France, Regular paper track. ¡hal-00274563¿ HAL Id: hal-00274563 https://hal.archives-ouvertes.fr/hal-00274563.

[18] A. Gajardo and E. Goles. Circuit universality of two dimensional cellular automata: a review. In C. S. Calude, editor, *Randomness and Complexity: From Leibniz to Chaitin*, chapter 7. World Scientific Publishing, Singapore, 2007.

[19] Y. Rogozhin. Small universal turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996.

[20] M. D. Davis and M. Davis. A note on universal turing machines. *Journal of Symbolic Logic*, 35(4):590–590, 1970.

[21] A. Anderson. Quantum canonical transformations. physical equivalence of quantum theories. *Phys. Lett. B*, 305(1–2):67–70, 1993.